SciUnit

Unit Tests for Scientific Models

Richard C Gerkin, PhD
Arizona State University, USA

Q: Why make models?

A2:

A2:

What I cannot create, I do not understand.

Know how to solve every problem that has been solved

Why const × Sqrt ...

TO LEARN:
Bethe Ansatz Probs.
Kondo
2-D Hall
accel. Temp
Non Linear Classical Hydro

ⓐ $f = U(r, a)$

$g = 4(r \cdot z) u(r, a)$

ⓑ $f = 2|r \cdot a|(u \cdot a)$

MODEL VALIDATION

MODEL VALIDATION

- A scientific model has high *validity* if:

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.
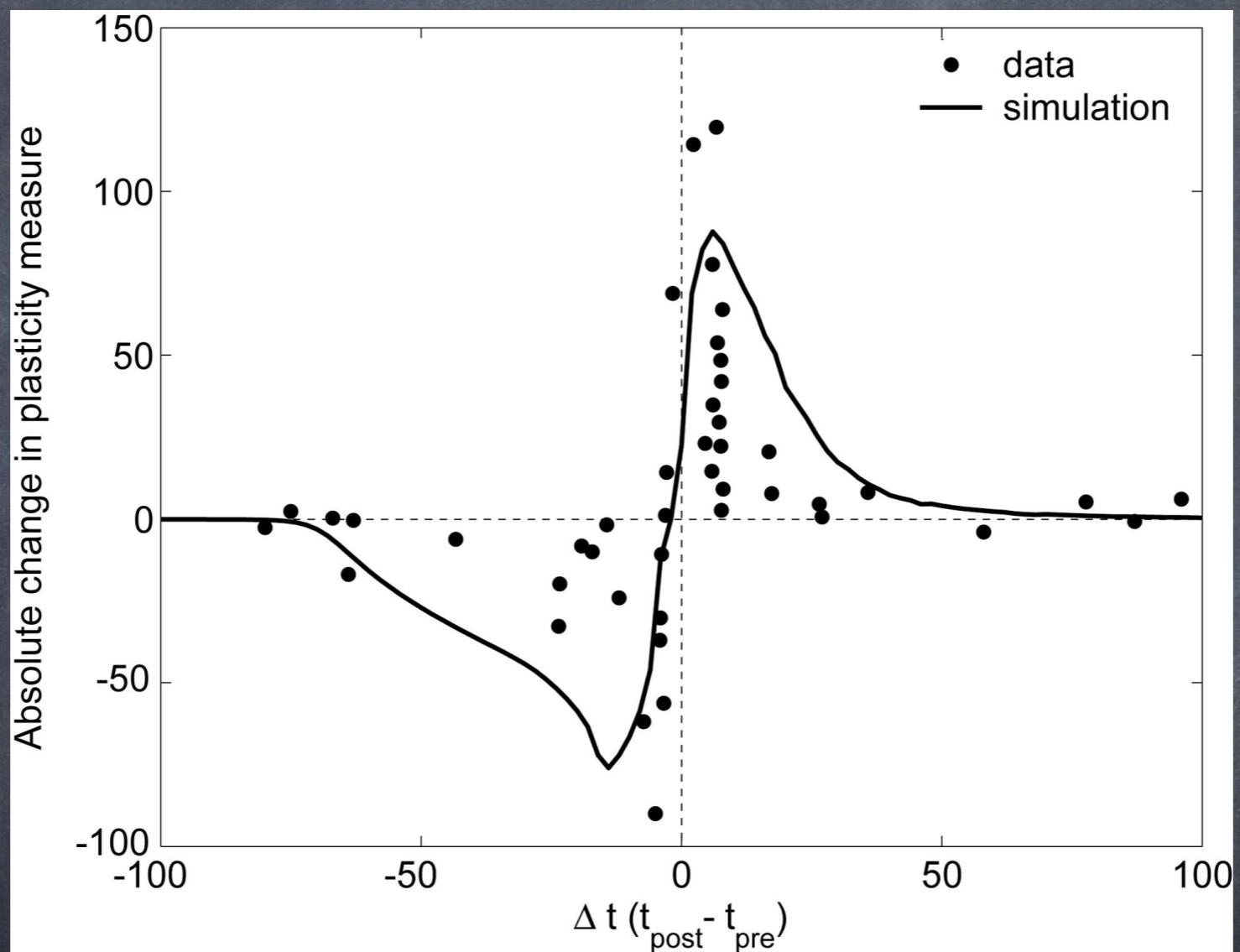
MODEL VALIDATION

- A scientific model has high *validity* if:

  - It is *consistent with* a wide range of previously gathered data.

MODEL VALIDATION
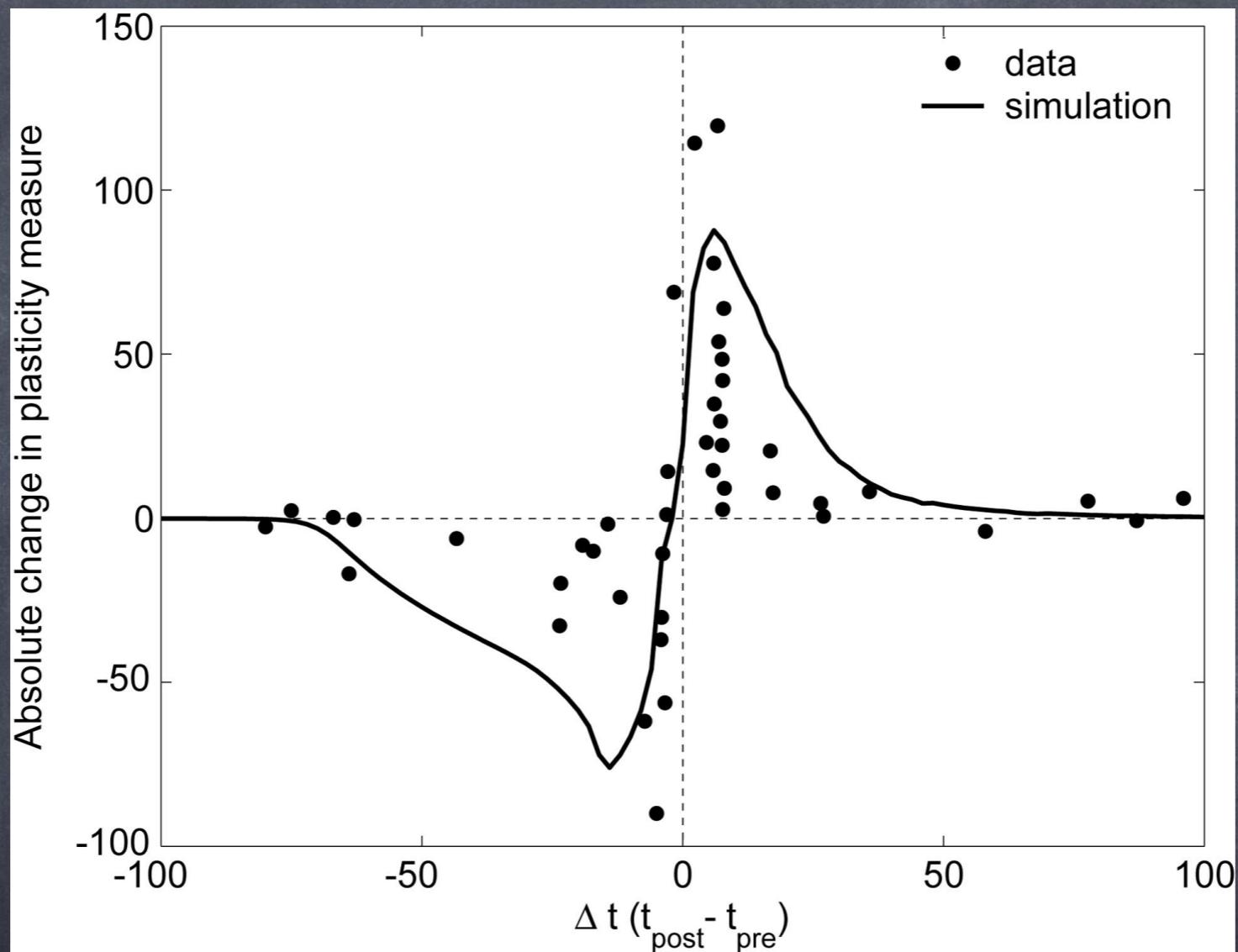
- A scientific model has high *validity* if:

  - 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

Typically, we make only *informal* arguments about a model's explanatory power.

TESTING VALIDITY

TESTING VALIDITY

And there is a ***huge*** amount of data out there now. *Informal validation* is becoming increasingly difficult!

TESTING VALIDITY

TESTING VALIDITY

- This leaves the field with an **unclear** view of:

TESTING VALIDITY

- This leaves the field with an **unclear** view of:

TESTING VALIDITY

- This leaves the field with an **unclear** view of:

  - What has already been explained.

TESTING VALIDITY

- This leaves the field with an **unclear** view of:

  - What has already been explained.

TESTING VALIDITY

- This leaves the field with an **unclear** view of:

  - What has already been explained.

  - What needs explaining.

TESTING VALIDITY

- This leaves the 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

- Takes a function, *f*, that maps inputs of type A to outputs of type B.

THE UNIT TEST

- Takes a function, *f,* that maps inputs of type A to outputs of type B.

THE UNIT TEST

- Takes a function, *f*, that maps inputs of type A to outputs of type B.

THE UNIT TEST

- Takes a function, *f*, that maps inputs of type A to outputs of type B.

- *f* passes the test if and only if f(a) in $[b_{low}, b_{high}]$.

THE UNIT TEST

- Takes a function, *f,* that maps inputs of type A to outputs of type B.

- *f* passes the test if and only if f(a) in [b_{low},b_{high}].

- [b_{low},b_{high}] are the *passing criteria.*

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.

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 cerebellar Purkinje cell should produce spikes that, given a specified description of the stimulus, **match physiologically observed** shapes, rates, interval distributions, etc.

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**?

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

SCIUNIT: CHALLENGES

SCIUNIT: CHALLENGES

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

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?

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?

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?

SCIUNIT: CAPABILITIES

SCIUNIT: CAPABILITIES

Q: How to *interface* with a wide range of model types, languages, and goals?

SCIUNIT: CAPABILITIES

Q: How to *interface* with a wide range of model types, languages, and goals?

A: Enumerate the *capabilities* of models.

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

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.

Test

Let's dance!

SCIUNIT: CAPABILITIES

SCIUNIT: CAPABILITIES

Test

OK, here are some observations.
Do what you said you could do with
them; then let me know your prediction.

SCIUNIT: CAPABILITIES

Test

Model2

OK, here are some observations.
Do what you said you could do with
them; then let me know your prediction.

Got it! Here it goes... CAPABILITY ERROR

SCIUNIT: CAPABILITIES

SCIUNIT: CAPABILITIES

Model2

OK, I'm ready, hit me up again.

SCIUNIT: CAPABILITIES

Model2

Test

OK, I'm ready, hit me up again.

OK, here are those observations again.
I'm calling your methods now.

SCIUNIT: CAPABILITIES

Model2

OK, I'm ready, hit me up again.

Test

OK, here are those observations again.
I'm calling your methods now.

Model2

It's working!   ... here's the
prediction.

SCIUNIT: CAPABILITIES

Model2

OK, I'm ready, hit me up again.

Test

OK, here are those observations again.
I'm calling your methods now.

Model2

It's working!   ... here's the
prediction.

Test

Thanks, hold on ... hmmm, your
prediction needs serious work.

SCIUNIT: CAPABILITIES

SCIUNIT: CAPABILITIES

```python
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

```python
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

```python
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

```python
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: MODELS

SCIUNIT: MODELS

```python
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

```python
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

```python
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

```python
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

```python
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

```python
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

```python
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: CAPABILITIES

SCIUNIT: CAPABILITIES

```python
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()

    def get_action_potential_widths(self):
        action_potentials = self.get_action_potentials()
        widths = [utils.ap_width(x) for x in action_potentials]
        return widths
```

SCIUNIT: CAPABILITIES

```python
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()

    def get_action_potential_widths(self):
        action_potentials = self.get_action_potentials()
        widths = [utils.ap_width(x) for x in action_potentials]
        return widths
```

- Auxiliary methods (relying on the core methods) are available to accelerate the workflow.

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**.

SCIUNIT: TESTS

SCIUNIT: TESTS

```python
class SpikeWidthTest(sciunit.Test):
    """Tests the full widths of spikes at their half-maximum."""

    def __init__(self,
                 observation={'mean':None,'std':None},
                 name="Action potential width"):
        """Takes the mean and standard deviation of observed spike widths"""

        Test.__init__(self,observation,name)

    required_capabilities = (ProducesMembranePotential,ProducesSpikes,)

    description = "A test of the widths of action potentials \
                   at half of their maximum height."

    score_type = sciunit.scores.ZScore

    ...
```

SCIUNIT: TESTS

```python
class SpikeWidthTest(sciunit.Test):
    """Tests the full widths of spikes at their half-maximum."""

    def __init__(self,
                observation={'mean':None,'std':None},
                name="Action potential width"):
        """Takes the mean and standard deviation of observed spike widths"""

        Test.__init__(self,observation,name)

    required_capabilities = (ProducesMembranePotential,ProducesSpikes,)

    description = "A test of the widths of action potentials \
                    at half of their maximum height."

    score_type = sciunit.scores.ZScore

    ...
```

- The constructor and attributes tell us:

SCIUNIT: TESTS

```python
class SpikeWidthTest(sciunit.Test):
    """Tests the full widths of spikes at their half-maximum."""

    def __init__(self,
                    observation={'mean':None,'std':None},
                    name="Action potential width"):
        """Takes the mean and standard deviation of observed spike widths"""

        Test.__init__(self,observation,name)

    required_capabilities = (ProducesMembranePotential,ProducesSpikes,)

    description = "A test of the widths of action potentials \
                    at half of their maximum height."

    score_type = sciunit.scores.ZScore

    ...
```

- The constructor and attributes tell us:
  - What the test needs in order to be parameterized.

SCIUNIT: TESTS

```python
class SpikeWidthTest(sciunit.Test):
    """Tests the full widths of spikes at their half-maximum."""

    def __init__(self,
                 observation={'mean':None,'std':None},
                 name="Action potential width"):
        """Takes the mean and standard deviation of observed spike widths"""

        Test.__init__(self,observation,name)

    required_capabilities = (ProducesMembranePotential,ProducesSpikes,)

    description = "A test of the widths of action potentials \
                   at half of their maximum height."

    score_type = sciunit.scores.ZScore

    ...
```

- The constructor and attributes tell us:
  - What the test needs in order to be parameterized.
  - What capabilities it will make use of.

SCIUNIT: TESTS

```python
class SpikeWidthTest(sciunit.Test):
    """Tests the full widths of spikes at their half-maximum."""

    def __init__(self,
                observation={'mean':None,'std':None},
                name="Action potential width"):
        """Takes the mean and standard deviation of observed spike widths"""

        Test.__init__(self,observation,name)

    required_capabilities = (ProducesMembranePotential,ProducesSpikes,)

    description = "A test of the widths of action potentials \
                    at half of their maximum height."

    score_type = sciunit.scores.ZScore

    ...
```

- The constructor and attributes tell us:
  - What the test needs in order to be parameterized.
  - What capabilities it will make use of.
  - What kind of score it will return.

SCIUNIT: TESTS

SCIUNIT: TESTS

```python
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

```python
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)
```

- Check the observation.

SCIUNIT: TESTS

```python
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)
```

- Check the observation.

SCIUNIT: TESTS

```python
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)
```

- Check the observation.

SCIUNIT: TESTS

```python
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)
```

- Check the observation.

SCIUNIT: TESTS

```python
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)
```

- Check the observation.

SCIUNIT: TESTS

```python
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)
```

- Check the observation.

- Coax a prediction from the model.

SCIUNIT: TESTS

```python
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)
```

- Check the observation.

- Coax a prediction from the model.

SCIUNIT: TESTS

```python
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)
```

- Check the observation.

- Coax a prediction from the model.

SCIUNIT: TESTS

```python
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)
```

- Check the observation.

- Coax a prediction from the model.

SCIUNIT: TESTS

```python
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)
```

- Check the observation.

- Coax a prediction from the model.

- Produce an indicator of model/data agrrement.

SCIUNIT: TESTS

```python
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)
```

- Check the observation.

- Coax a prediction from the model.

- Produce an indicator of model/data agrrement.

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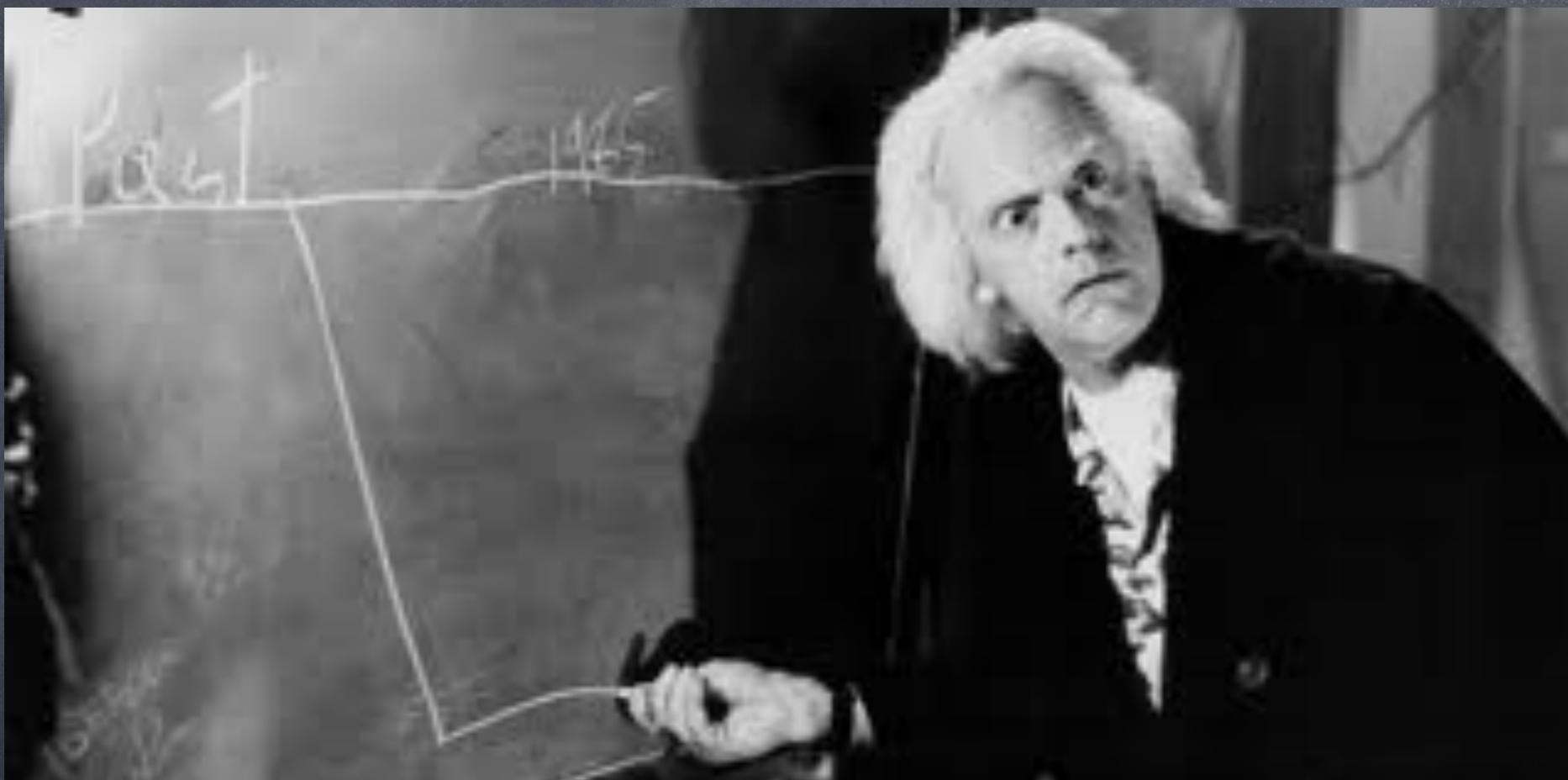
SCIUNIT: TESTS

SCIUNIT: TESTS

"Your test sucks. Why would you think that was a fair comparison to the data? I could totally do better!"

SCIUNIT: TESTS

"Your test sucks. Why would you think that was a fair comparison to the data? I could totally do better!"

Fork it!

SCIUNIT: SCORES

SCIUNIT: SCORES

- BooleanScore [True (Pass) or False (Fail)]

SCIUNIT: SCORES

- BooleanScore [True (Pass) or False (Fail)]

SCIUNIT: SCORES

- BooleanScore [True (Pass) or False (Fail)]

- PercentScore [0 - 100%]

SCIUNIT: SCORES

- BooleanScore [True (Pass) or False (Fail)]

- PercentScore [0 - 100%]

SCIUNIT: SCORES

- BooleanScore [True (Pass) or False (Fail)]

- PercentScore [0 - 100%]

- ZScore (-∞, ∞)

SCIUNIT: SCORES

- BooleanScore [True (Pass) or False (Fail)]

- PercentScore [0 - 100%]

- ZScore (-∞, ∞)

SCIUNIT: SCORES

- BooleanScore [True (Pass) or False (Fail)]

- PercentScore [0 - 100%]

- ZScore ($-\infty$, ∞)

- FloatScore ($-\infty$, ∞)

SCIUNIT: SCORES

- BooleanScore [True (Pass) or False (Fail)]

- PercentScore [0 - 100%]

- ZScore (-∞, ∞)

- FloatScore (-∞, ∞)

SCIUNIT: SCORES

- BooleanScore [True (Pass) or False (Fail)]

- PercentScore [0 - 100%]

- ZScore (-∞, ∞)

- FloatScore (-∞, ∞)

- PositiveIntegerScore [1, ∞)

SCIUNIT: SCORES

- BooleanScore [True (Pass) or False (Fail)]

- PercentScore [0 - 100%]

- ZScore (-∞, ∞)

- FloatScore (-∞, ∞)

- PositiveIntegerScore [1, ∞)

SCIUNIT: SCORES

- BooleanScore [True (Pass) or False (Fail)]

- PercentScore [0 - 100%]

- ZScore (-∞, ∞)

- FloatScore (-∞, ∞)

- PositiveIntegerScore [1, ∞)

- PValueScore [0.0 - 1.0]

SCIUNIT: SCORES

- BooleanScore [True (Pass) or False (Fail)]

- PercentScore [0 - 100%]

- ZScore (-∞, ∞)

- FloatScore (-∞, ∞)

- PositiveIntegerScore [1, ∞)

- PValueScore [0.0 - 1.0]

SCIUNIT: SCORES

- BooleanScore [True (Pass) or False (Fail)]

- PercentScore [0 - 100%]

- ZScore (-∞, ∞)

- FloatScore (-∞, ∞)

- PositiveIntegerScore [1, ∞)

- PValueScore [0.0 - 1.0]

SCIUNIT: SCORES

- BooleanScore [True (Pass) or False (Fail)]

- PercentScore [0 - 100%]

- ZScore $(-\infty, \infty)$

- FloatScore $(-\infty, \infty)$

- PositiveIntegerScore $[1, \infty)$

- PValueScore [0.0 - 1.0]

SCIUNIT: SCORES

- BooleanScore [True (Pass) or False (Fail)]

- PercentScore [0 - 100%]

- ZScore (-∞, ∞)

- FloatScore (-∞, ∞)

- PositiveIntegerScore [1, ∞)

- PValueScore [0.0 - 1.0]

SCIUNIT: SCORES

- BooleanScore [True (Pass) or False (Fail)]

- PercentScore [0 - 100%]

- ZScore (-∞, ∞)

- FloatScore (-∞, ∞)

- PositiveIntegerScore [1, ∞)

- PValueScore [0.0 - 1.0]

SCIUNIT: SCORES

- BooleanScore [True (Pass) or False (Fail)]

- PercentScore [0 - 100%]

- ZScore $(-\infty, \infty)$

- FloatScore $(-\infty, \infty)$

- PositiveIntegerScore $[1, \infty)$

- PValueScore [0.0 - 1.0]

- Conversion between scores is also possible.

SCIUNIT: SCORES

- BooleanScore [True (Pass) or False (Fail)]

- PercentScore [0 - 100%]

- ZScore (-∞, ∞)

- FloatScore (-∞, ∞)

- PositiveIntegerScore [1, ∞)

- PValueScore [0.0 - 1.0]

- Conversion between scores is also possible.
    - $p < 0.01$ → True; else False

SCIUNIT: SCORES

- BooleanScore [True (Pass) or False (Fail)]

- PercentScore [0 - 100%]

- ZScore (-∞, ∞)

- FloatScore (-∞, ∞)

- PositiveIntegerScore [1, ∞)

- PValueScore [0.0 - 1.0]

- Conversion between scores is also possible.
  - $p < 0.01$ → True; else False

SCIUNIT: SCORES

- BooleanScore [True (Pass) or False (Fail)]

- PercentScore [0 - 100%]

- ZScore $(-\infty, \infty)$

- FloatScore $(-\infty, \infty)$

- PositiveIntegerScore $[1, \infty)$

- PValueScore [0.0 - 1.0]

- Conversion between scores is also possible.
    - $p < 0.01 \rightarrow$ True; else False

- Scores have attributes describing their provenance:

SCIUNIT: SCORES

- BooleanScore [True (Pass) or False (Fail)]

- PercentScore [0 - 100%]

- ZScore (-∞, ∞)

- FloatScore (-∞, ∞)

- PositiveIntegerScore [1, ∞)

- PValueScore [0.0 - 1.0]

- Conversion between scores is also possible.
  - $p < 0.01 \rightarrow$ True; else False

- Scores have attributes describing their provenance:
  - Test

SCIUNIT: SCORES

- BooleanScore [True (Pass) or False (Fail)]

- PercentScore [0 - 100%]

- ZScore (-∞, ∞)

- FloatScore (-∞, ∞)

- PositiveIntegerScore [1, ∞)

- PValueScore [0.0 - 1.0]

- Conversion between scores is also possible.
  - $p < 0.01$ → True; else False

- Scores have attributes describing their provenance:
  - Test
  - Model

SCIUNIT: SCORES

- BooleanScore [True (Pass) or False (Fail)]

- PercentScore [0 - 100%]

- ZScore (-∞, ∞)

- FloatScore (-∞, ∞)

- PositiveIntegerScore [1, ∞)

- PValueScore [0.0 - 1.0]

- Conversion between scores is also possible.
  - $p < 0.01$ → True; else False

- Scores have attributes describing their provenance:
  - Test
  - Model
  - Related data (for further analysis)

SCIUNIT: SCORES

- BooleanScore [True (Pass) or False (Fail)]

- PercentScore [0 - 100%]

- ZScore (-∞, ∞)

- FloatScore (-∞, ∞)

- PositiveIntegerScore [1, ∞)

- PValueScore [0.0 - 1.0]

- Conversion between scores is also possible.
  - $p < 0.01$ → True; else False

- Scores have attributes describing their provenance:
  - Test
  - Model
  - Related data (for further analysis)

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.

```python
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.

```python
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."""

        ...
```

```python
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_2 = InjectedCurrentSpikeWidthTest(observation, injection_params(75.0))

width_suite = sciunit.TestSuite([width_test_1,width_test_2,width_test_3])
```

SCIUNIT: WORKFLOW

SCIUNIT: WORKFLOW

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

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

VISUALIZING VALIDITY

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

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

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

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

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

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

VISUALIZING VALIDITY

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

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
ThompsonsMethod	-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
ThompsonsMethod	-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
ThompsonsMethod	-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
ThompsonsMethod	-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
ThompsonsMethod	-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
ThompsonsMethod	-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
ThompsonsMethod	-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

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

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

COMPETITION

```python
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 Berger[1], Brice Bathellier[2], Matteo Carandini[3] and Wulfram Gerstner[1]

[1] Ecole Polytechniqe Federal de Lausanne (EPFL), Switzerland
[2] University of Bern, Switzerland
[3] University College London, United Kingdom

COMPETITION

```python
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 Berger[1], Brice Bathellier[2], Matteo Carandini[3] and Wulfram Gerstner[1]

[1] Ecole Polytechniqe Federal de Lausanne (EPFL), Switzerland
[2] University of Bern, Switzerland
[3] University College London, United Kingdom

https://github.com/incf/qsnmc

COMPETITION

COMPETITION

	This branch is 14 commits ahead of scidash:master		⬆ Pull Request ⬆ Compare
	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   ⬆ Compare

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

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

This enables:

COMPETITION

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

This enables:

- rapid implementation from a consistent skeleton

COMPETITION

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

This enables:

- rapid implementation from a consistent skeleton

- automation of test suite execution

COMPETITION

This branch is 14 commits ahead of scidash:master ⬡ Pull Request ⬆ Compare

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

- automation of test suite execution

- forking for competing visions

COMPETITION

COMPETITION

Competitions in the collaboration queue:

COMPETITION

Competitions in the collaboration queue:

- Quantitative Single Neuron Modeling Competion (*QSNMC*)

COMPETITION

Competitions in the collaboration queue:

- Quantitative Single Neuron Modeling Competion (*QSNMC*)
  - reduced, tractable neuron models

COMPETITION

Competitions in the collaboration queue:

- Quantitative Single Neuron Modeling Competion (*QSNMC*)
  - reduced, tractable neuron models
- Open Source Brain (http://www.opensourcebrain.org)

COMPETITION

Competitions in the collaboration queue:

- Quantitative Single Neuron Modeling Competion (*QSNMC*)
  - reduced, tractable neuron models
- Open Source Brain (http://www.opensourcebrain.org)
  - biophysically detailed neuron/microcircuit models

COMPETITION

Competitions in the collaboration queue:

- Quantitative Single Neuron Modeling Competion (*QSNMC*)
  - reduced, tractable neuron models
- Open Source Brain (http://www.opensourcebrain.org)
  - biophysically detailed neuron/microcircuit models

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

COMPETITION

Competitions in the collaboration queue:

- Quantitative Single Neuron Modeling Competion (*QSNMC*)
  - reduced, tractable neuron models
- Open Source Brain (http://www.opensourcebrain.org)
  - biophysically detailed neuron/microcircuit models

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

- Spike-sorting algorithms

COMPETITION

Competitions in the collaboration queue:

- Quantitative Single Neuron Modeling Competion (*QSNMC*)
  - reduced, tractable neuron models
- Open Source Brain (http://www.opensourcebrain.org)
  - biophysically detailed neuron/microcircuit models

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

- Spike-sorting algorithms
  - formalization and extension of Schmitzer-Torbert et al, 2005.

COMPETITION

Competitions in the collaboration queue:

- Quantitative Single Neuron Modeling Competion (*QSNMC*)
  - reduced, tractable neuron models
- Open Source Brain (http://www.opensourcebrain.org)
  - biophysically detailed neuron/microcircuit models

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

- Spike-sorting algorithms
  - formalization and extension of Schmitzer-Torbert et al, 2005.
- Spike time extraction from fast calcium imaging.

COMPETITION

Competitions in the collaboration queue:

- Quantitative Single Neuron Modeling Competion (*QSNMC*)
  - reduced, tractable neuron models
- Open Source Brain (http://www.opensourcebrain.org)
  - biophysically detailed neuron/microcircuit models

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

- Spike-sorting algorithms
  - formalization and extension of Schmitzer-Torbert et al, 2005.
- Spike time extraction from fast calcium imaging.
  - Jason Kerr has a "ground truth" data set to enable this.

COMPETITION

Competitions in the collaboration queue:

- Quantitative Single Neuron Modeling Competion (*QSNMC*)
  - reduced, tractable neuron models
- Open Source Brain (http://www.opensourcebrain.org)
  - biophysically detailed neuron/microcircuit models

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

- Spike-sorting algorithms
  - formalization and extension of Schmitzer-Torbert et al, 2005.
- Spike time extraction from fast calcium imaging.
  - Jason Kerr has a "ground truth" data set to enable this.
- Neural system prediction and identification challenge (*nuSPIC*)

COMPETITION

Competitions in the collaboration queue:

- Quantitative Single Neuron Modeling Competion (*QSNMC*)
    - reduced, tractable neuron models
- Open Source Brain (http://www.opensourcebrain.org)
    - biophysically detailed neuron/microcircuit models

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

- Spike-sorting algorithms
    - formalization and extension of Schmitzer-Torbert et al, 2005.
- Spike time extraction from fast calcium imaging.
    - Jason Kerr has a "ground truth" data set to enable this.
- Neural system prediction and identification challenge (*nuSPIC*)
    - Can we infer the function of neuronal networks given connectivity?

COMPETITION

Competitions in the collaboration queue:

- Quantitative Single Neuron Modeling Competion (*QSNMC*)
  - reduced, tractable neuron models
- Open Source Brain (http://www.opensourcebrain.org)
  - biophysically detailed neuron/microcircuit models

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

- Spike-sorting algorithms
  - formalization and extension of Schmitzer-Torbert et al, 2005.
- Spike time extraction from fast calcium imaging.
  - Jason Kerr has a "ground truth" data set to enable this.
- Neural system prediction and identification challenge (*nuSPIC*)
  - Can we infer the function of neuronal networks given connectivity?
  - A latter day "Hopfield challenge".

COMPETITION

Competitions in the collaboration queue:

- Quantitative Single Neuron Modeling Competion (*QSNMC*)
  - reduced, tractable neuron models
- Open Source Brain (http://www.opensourcebrain.org)
  - biophysically detailed neuron/microcircuit models

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

- Spike-sorting algorithms
  - formalization and extension of Schmitzer-Torbert et al, 2005.
- Spike time extraction from fast calcium imaging.
  - Jason Kerr has a "ground truth" data set to enable this.
- Neural system prediction and identification challenge (*nuSPIC*)
  - Can we infer the function of neuronal networks given connectivity?
  - A latter day "Hopfield challenge".
- Brain computer interface (BCI, PhyPa (http://www.phypa.org)

COMPETITION

Competitions in the collaboration queue:

- Quantitative Single Neuron Modeling Competion (*QSNMC*)
  - reduced, tractable neuron models
- Open Source Brain (http://www.opensourcebrain.org)
  - biophysically detailed neuron/microcircuit models

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

- Spike-sorting algorithms
  - formalization and extension of Schmitzer-Torbert et al, 2005.
- Spike time extraction from fast calcium imaging.
  - Jason Kerr has a "ground truth" data set to enable this.
- Neural system prediction and identification challenge (*nuSPIC*)
  - Can we infer the function of neuronal networks given connectivity?
  - A latter day "Hopfield challenge".
- Brain computer interface (BCI, PhyPa (http://www.phypa.org)
- Seizure prediction (Sejnowski challenge)

COMPETITION

Competitions in the collaboration queue:

- Quantitative Single Neuron Modeling Competion (*QSNMC*)
  - reduced, tractable neuron models
- Open Source Brain (http://www.opensourcebrain.org)
  - biophysically detailed neuron/microcircuit models

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

- Spike-sorting algorithms
  - formalization and extension of Schmitzer-Torbert et al, 2005.
- Spike time extraction from fast calcium imaging.
  - Jason Kerr has a "ground truth" data set to enable this.
- Neural system prediction and identification challenge (*nuSPIC*)
  - Can we infer the function of neuronal networks given connectivity?
  - A latter day "Hopfield challenge".
- Brain computer interface (BCI, PhyPa (http://www.phypa.org)
- Seizure prediction (Sejnowski challenge)

COMPETITION

Competitions in the collaboration queue:

- Quantitative Single Neuron Modeling Competion (*QSNMC*)
  - reduced, tractable neuron models
- Open Source Brain (http://www.opensourcebrain.org)
  - biophysically detailed neuron/microcircuit models

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

- Spike-sorting algorithms
  - formalization and extension of Schmitzer-Torbert et al, 2005.
- Spike time extraction from fast calcium imaging.
  - Jason Kerr has a "ground truth" data set to enable this.
- Neural system prediction and identification challenge (*nuSPIC*)
  - Can we infer the function of neuronal networks given connectivity?
  - A latter day "Hopfield challenge".
- Brain computer interface (BCI, PhyPa (http://www.phypa.org)
- Seizure prediction (Sejnowki challenge)

CHALLENGES

CHALLENGES

- A big challenge is handling **different implementation languages** with a common system.

CHALLENGES

- A big challenge is handling **different implementation languages** with a common system.

- We must be able to express **complex model capabilities and test requirements**.

CHALLENGES

- A big challenge is handling **different implementation languages** with a common system.

- We must be able to express **complex model capabilities and test requirements**.

- We must be able to handle **large, diverse data sets** and **complex models**.

CHALLENGES

- A big challenge is handling **different implementation languages** with a common system.

- We must be able to express **complex model capabilities and test requirements**.

- We must be able to handle **large, diverse data sets** and **complex models**.

- We aim to solve these with
unit testing philosophy + domain standards

NEURONUNIT

NEURONUNIT

- *SciUnit* is most useful with *domain-specific* libraries of tests, capabilities, and skeleton models.

NEURONUNIT

- *SciUnit* is most useful with *domain-specific* libraries of tests, capabilities, and skeleton models.

- *NeuronUnit* is a such a library for single neuron electrophysiology.

NEURONUNIT

- *SciUnit* is most useful with *domain-specific* libraries of tests, capabilities, and skeleton models.

- *NeuronUnit* is a such a library for single neuron electrophysiology.

- http://github.com/scidash/neuronunit

NEURONUNIT

NEURONUNIT

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

NEURONUNIT

- SciUnit *capability* classes crafted to span the things one would ask single cell neurophyiology 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 neurophyiology 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 neurophyiology 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 neurophyiology 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: ?

NEUROELECTRO

http://www.neuroelectro.org

NEUROELECTRO

http://www.neuroelectro.org

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 explictly refered 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': 2.60208333333333,
 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 explictly refered 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': 2.60208333333333,
 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* to parameterize a test.

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 explictly refered 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': 2.60208333333333,
 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* to parameterize a test.

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)

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*).

```python
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*).

```python
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.

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 GET OUT
(MODELER'S PERSPECTIVE)

WHAT YOU GET OUT
(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.

WHAT YOU GET OUT
(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.

WHAT YOU GET OUT
(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).

WHAT YOU GET OUT
(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.

WHAT YOU GET OUT
(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 who demand that your model pass SciUnit tests.

WHAT YOU GET OUT
(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 who demand that your model pass SciUnit tests.

- Post-publication review of your model, *even as new data come to light*.

WHAT YOU GET OUT
(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 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.

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.

WHAT YOU GET OUT
(EXPERIMENTALIST'S/TESTER'S PERSPECTIVE)

WHAT YOU GET OUT
(EXPERIMENTALIST'S/TESTER'S PERSPECTIVE)

- The ability to put your observations in context.

WHAT YOU GET OUT
(EXPERIMENTALIST'S/TESTER'S PERSPECTIVE)

- The ability to put your observations in context.

  - Is there a model that explains my data?

WHAT YOU GET OUT
(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 YOU GET OUT
(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?

WHAT YOU GET OUT
(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.

WHAT YOU GET OUT
(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 X, and it gives result Y, which model does that support? What about result Z?

WHAT YOU GET OUT
(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 X, and it gives result Y, which model does that support? What about result Z?

- Increased community awareness of the data you collected.

WHAT YOU GET OUT
(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 X, and it gives result Y, which model does that support? What about result Z?

- Increased community awareness of the data you collected.

  - It could become the gold standard by which models are judged!

WHAT YOU GET OUT
(OPENWORM PERSPECTIVE)

WHAT YOU GET OUT
(OPENWORM PERSPECTIVE)

- You are building the first model in history with a realistic chance of faithfully simulating an entire organism.

WHAT YOU GET OUT
(OPENWORM PERSPECTIVE)

- You are building the first model in history with a realistic chance of faithfully simulating an entire organism.

- To faithfully simulate an entire organism, you need to get a lot of things right.

WHAT YOU GET OUT
(OPENWORM PERSPECTIVE)

- You are building the first model in history with a realistic chance of faithfully simulating an entire organism.

- To faithfully simulate an entire organism, you need to get a lot of things right.

- How do you know if you are getting them right?

WHAT YOU GET OUT
(OPENWORM PERSPECTIVE)

- You are building the first model in history with a realistic chance of faithfully simulating an entire organism.

- To faithfully simulate an entire organism, you need to get a lot of things right.

- How do you know if you are getting them right?

  - Do the simulated neurons' membrane potentials evolve in time like the the real C. elegans?

WHAT YOU GET OUT
(OPENWORM PERSPECTIVE)

- You are building the first model in history with a realistic chance of faithfully simulating an entire organism.

- To faithfully simulate an entire organism, you need to get a lot of things right.

- How do you know if you are getting them right?

  - Do the simulated neurons' membrane potentials evolve in time like the the real C. elegans?

  - Do the simulated muscles contract with the same timing and intensity as the real worm?

WHAT YOU GET OUT
(OPENWORM PERSPECTIVE)

- You are building the first model in history with a realistic chance of faithfully simulating an entire organism.

- To faithfully simulate an entire organism, you need to get a lot of things right.

- How do you know if you are getting them right?

  - Do the simulated neurons' membrane potentials evolve in time like the the real C. elegans?

  - Do the simulated muscles contract with the same timing and intensity as the real worm?

  - Does the worm sim swim like the real worm?

OPENWORM AND SCIUNIT

OPENWORM AND SCIUNIT

- You have:

OPENWORM AND SCIUNIT

- You have:

  - Hours and GB's of video of C. elegans swimming, digitized.

OPENWORM AND SCIUNIT

- You have:

  - Hours and GB's of video of C. elegans swimming, digitized.

  - A sim that swims.

OPENWORM AND SCIUNIT

- You have:

  - Hours and GB's of video of C. elegans swimming, digitized.

  - A sim that swims.

- I have:

OPENWORM AND SCIUNIT

- You have:

  - Hours and GB's of video of C. elegans swimming, digitized.

  - A sim that swims.

- I have:

  - A framework to adjudicate goodness-of-fit between the swimming patterns of C. elegant and the worm sim.

OPENWORM AND SCIUNIT

- You have:

  - Hours and GB's of video of C. elegans swimming, digitized.

  - A sim that swims.

- I have:

  - A framework to adjudicate goodness-of-fit between the swimming patterns of C. elegant and the worm sim.

  - An incentive to cut *SciUnit*'s teeth on a new domain (whole organism behavior) and some complex and challenging simulations.

OPENWORM AND SCIUNIT

- You have:

  - Hours and GB's of video of C. elegans swimming, digitized.

  - A sim that swims.

- I have:

  - A framework to adjudicate goodness-of-fit between the swimming patterns of C. elegant and the worm sim.

  - An incentive to cut *SciUnit*'s teeth on a new domain (whole organism behavior) and some complex and challenging simulations.

  - The chance to contribute to an awesome project!

OPENWORM AND SCIUNIT

- You have:

  - Hours and GB's of video of C. elegans swimming, digitized.

  - A sim that swims.

- I have:

  - A framework to adjudicate goodness-of-fit between the swimming patterns of C. elegant and the worm sim.

  - An incentive to cut *SciUnit*'s teeth on a new domain (whole organism behavior) and some complex and challenging simulations.

  - The chance to contribute to an awesome project!

OPENWORM AND SCIUNIT

- You have:

  - Hours and GB's of video of C. elegans swimming, digitized.

  - A sim that swims.

- I have:

  - A framework to adjudicate goodness-of-fit between the swimming patterns of C. elegant and the worm sim.

  - An incentive to cut *SciUnit*'s teeth on a new domain (whole organism behavior) and some complex and challenging simulations.

  - The chance to contribute to an awesome project!

OPENWORM AND SCIUNIT

- You have:

  - Hours and GB's of video of C. elegans swimming, digitized.

  - A sim that swims.

- I have:

  - A framework to adjudicate goodness-of-fit between the swimming patterns of C. elegant and the worm sim.

  - An incentive to cut *SciUnit*'s teeth on a new domain (whole organism behavior) and some complex and challenging simulations.

  - The chance to contribute to an awesome project!

ACKNOWLEDGEMENTS

- **Cyrus Omar**
  Carnegie Mellon University

- **Shreejoy Tripathy**
  Univ. of British Columbia

- **Sharon Crook**
  Arizona State University

- **Padraig Gleeson**
  University College, London