pykCSD Documentation

Release 0.1.0

Grzegorz Parka

September 22, 2014

Contents

Contents:

CHAPTER 1

pykCSD

Kernel Current Source Density is a recently developed method for estimating the density of trans-membrane current sources, which can be used for a detailed study of neuronal synaptic dynamics.

It can estimate current source density from potentials measured with irregularly placed linear, planar and spatial electrodes.

- Free software: BSD license
- Documentation: http://pykCSD.rtfd.org.

1.1 Features

- Estimation of potentials and CSD in 1D, 2D, 3D case for both static and dynamic recordings
- Visualization of the estimated quantities

1.2 TODO

- tracking the units
- management of big datasets
- GUI

CHAPTER 2

Installation

Make sure you have Scipy and other requirements installed:

```
$ sudo apt-get install build-essential python-dev python-numpy python-setuptools python-scipy libatla
$ sudo pip install -U scikit-learn
```

command to install dependencies:

```
$ pip install -r requirements.txt
```

At the command line:

```
$ easy_install pykCSD
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv pykCSD
$ pip install pykCSD
```

CHAPTER 3

Usage

To use pykCSD in a project:

```
from pykCSD.pykCSD import KCSD
import numpy as np

elec_pos = np.array([[0, 0], [0, 1], [1, 0], [1,1], [0.5, 0.5]])
pots = np.array([[0], [0], [0], [0], [1]])
params = {'gdX': 0.05, 'gdY': 0.05}

k = KCSD(elec_pos, pots, params)

k.estimate_pots()
k.estimate_csd()

k.plot_all()
```

More detailed instructions can be found in the Use Cases section.

CHAPTER 4

Use Cases

With the pykCSD toolbox you can estimate 1D, 2D and 3D potentials and CSD based on your input data. Here are the basic examples for each of the reconstructions.

4.1 Sample 1D reconstruction

You can estimate potentials measured with electrodes placed along a line:

```
from pykCSD.pykCSD import KCSD
import numpy as np

#the most inner list corresponds to a position of one electrode
elec_pos = np.array([[-0.5], [0], [1], [1.5], [3.5], [4.1], [5.0], [7.0], [8.0]])

#the most inner list corresponds to a time recording made with one electrode
pots = np.array([[-0.1], [0.3], [-0.4], [0.2], [0.8], [0.5], [0.2], [0.5], [0.6]])

#you can define model parameters as a dictionary
params = {
        'xmin': -3.0,
        'xmax': 12.0,
        'source_type': 'gauss',
        'n_sources': 30
}

k = KCSD(elec_pos, pots, params)

k.estimate_pots()
k.estimate_csd()

k.plot_all()
```

4.2 Cross validation

Having your kCSD solver set up, you can use cross validation to regularize your results:

```
from pykCSD import cross_validation as cv
from sklearn.cross_validation import LeaveOneOut
```

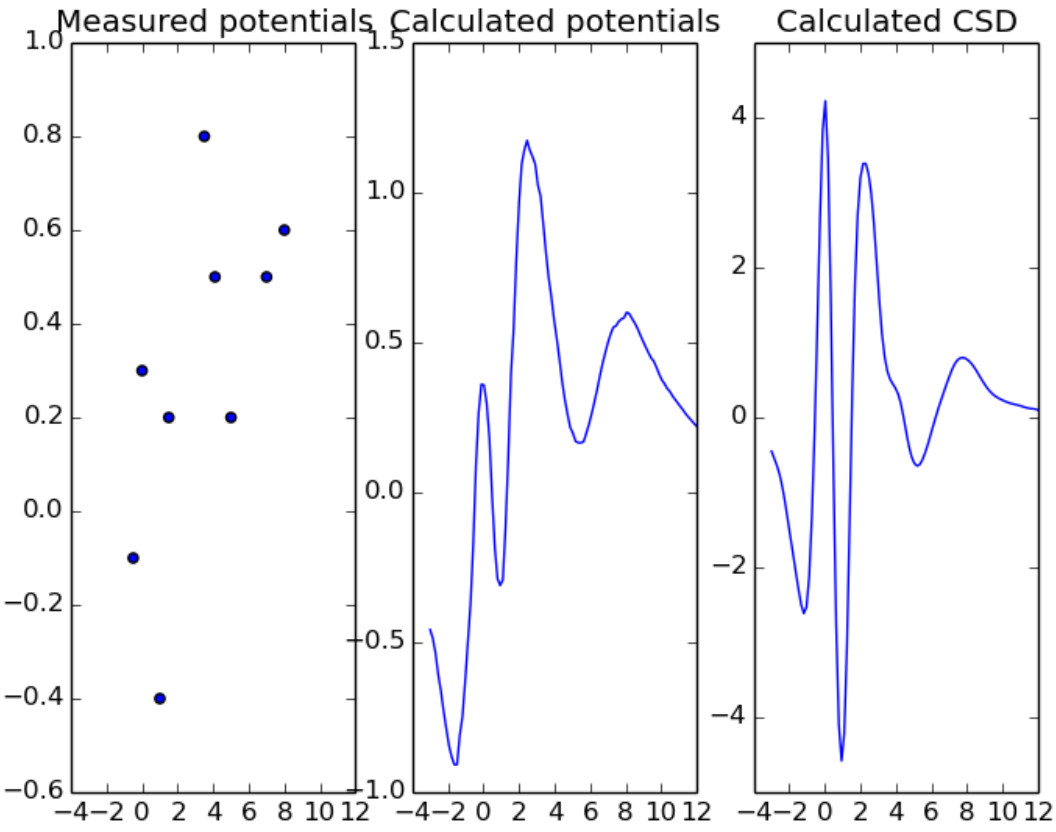

Figure 4.1: The sample reconstruction in 1D

```
index_generator = LeaveOneOut(len(elec_pos), indices=True)
lambdas = np.array([10000./x**2 for x in xrange(1, 50)])

k.solver.lambd = cv.choose_lambda(lambdas, pots, k.solver.k_pot, elec_pos, index_generator)

print k.solver.lambd

k.estimate_pots()
k.estimate_csd()

k.plot_all()

>> 4.16493127863
```

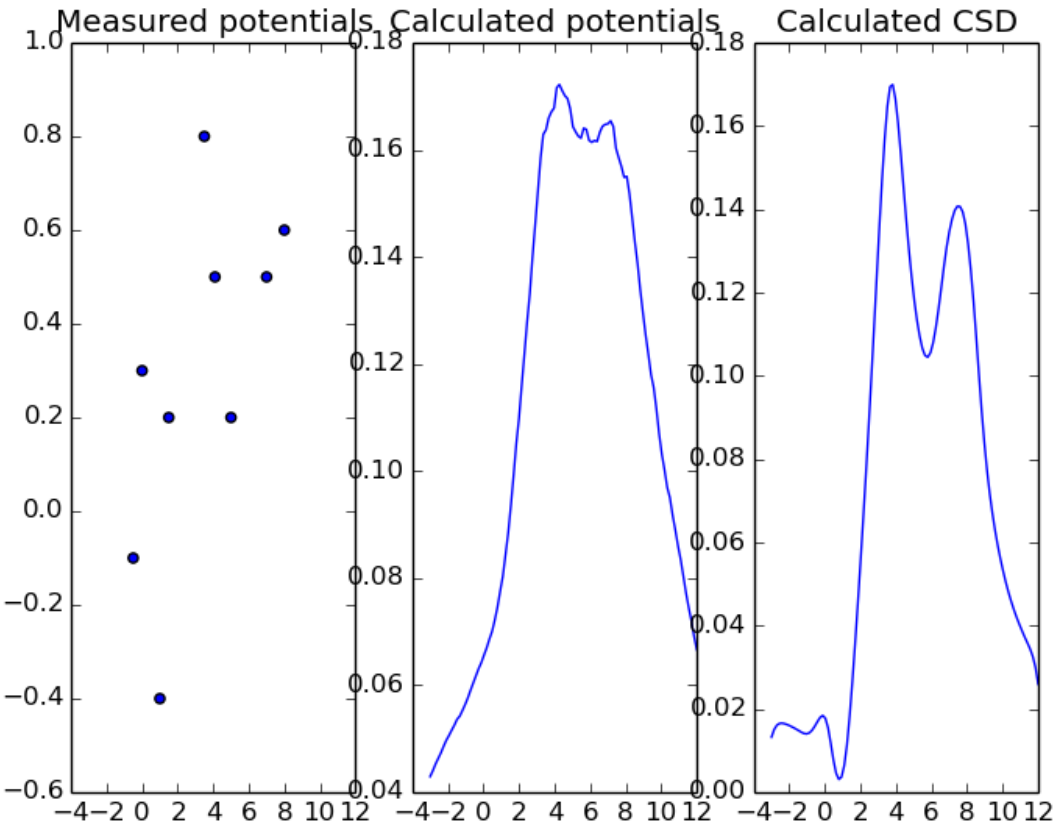

Figure 4.2: The same reconstruction regularized with cross validation

4.3 Sample 2D reconstruction

You can estimate potentials and CSD measured with planar electrodes:

```
from pykCSD.pykCSD import KCSD
import numpy as np
```

```
elec_pos = np.array([[-0.2, -0.2],[0, 0], [0, 1], [1, 0], [1,1], [0.5, 0.5], [1.2, 1.2]])
pots = np.array([[-1], [-1], [-1], [0], [0], [1], [-1.5]])
params = {'gdX': 0.05, 'gdY': 0.05, 'xmin': -2.0, 'xmax': 2.0, 'ymin': -2.0, 'ymax': 2.0}

k = KCSD(elec_pos, pots, params)

k.estimate_pots()
k.estimate_csd()

k.plot_all()
```

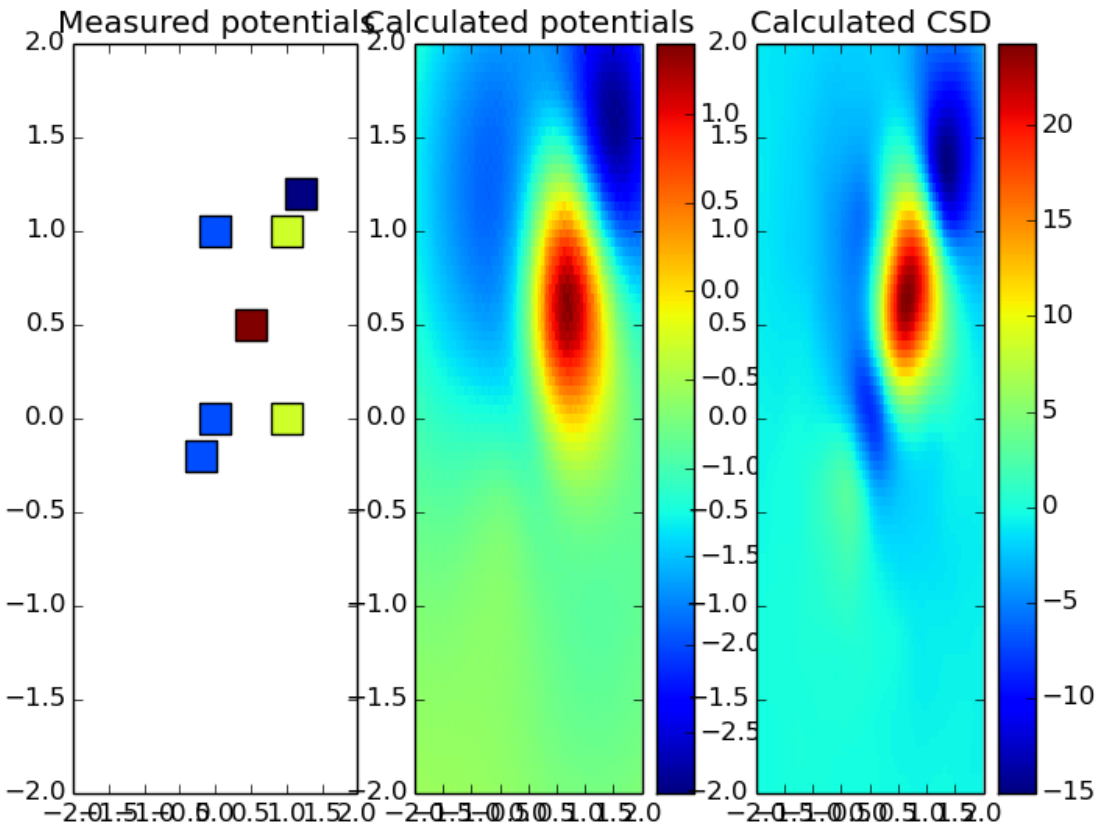

Figure 4.3: The sample reconstruction in 2D

4.4 Sample 3D reconstruction

You can also recostruct CSD and LFP using measurements taken by spatial electrodes:

```
from pykCSD.pykCSD import KCSD
import numpy as np

elec_pos = np.array([(0, 0, 0), (0, 0, 1), (0, 1, 0), (1, 0, 0),
                     (0, 1, 1), (1, 1, 0), (1, 0, 1), (1, 1, 1),
```

```
                          (0.5, 0.5, 0.5)])
pots = np.array([[-0.5], [0], [-0.5], [0], [0], [0.2], [0], [0], [1]])
params = {
        'gdX': 0.05,
        'gdY': 0.05,
        'gdZ': 0.05,
        'n_sources': 64,
}
k = KCSD(elec_pos, pots, params)

k.estimate_pots()
k.estimate_csd()

k.plot_all()
```

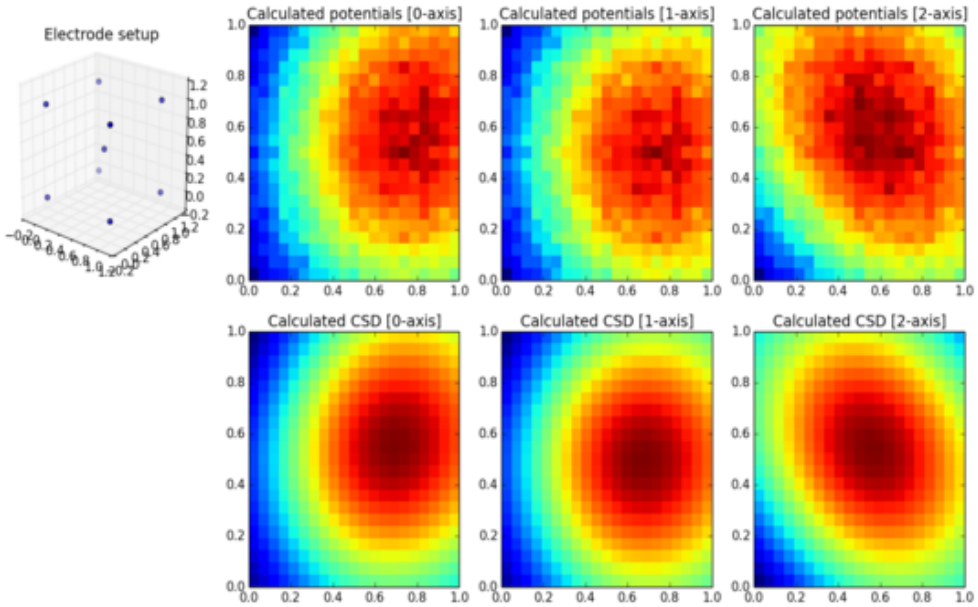

Figure 4.4: The sample reconstruction in 3D

Such a dataset can be also visualized using mayavi:

```
from mayavi import mlab

csd = k.solver.estimated_csd[:,:,:,0]
#setting up a proper gui backend
%gui wx
mlab.pipeline.image_plane_widget(mlab.pipeline.scalar_field(csd),
                         plane_orientation='x_axes',
                         slice_index=10,
                         )
mlab.pipeline.image_plane_widget(mlab.pipeline.scalar_field(csd),
                         plane_orientation='y_axes',
                         slice_index=10,
                         )
mlab.outline()
```

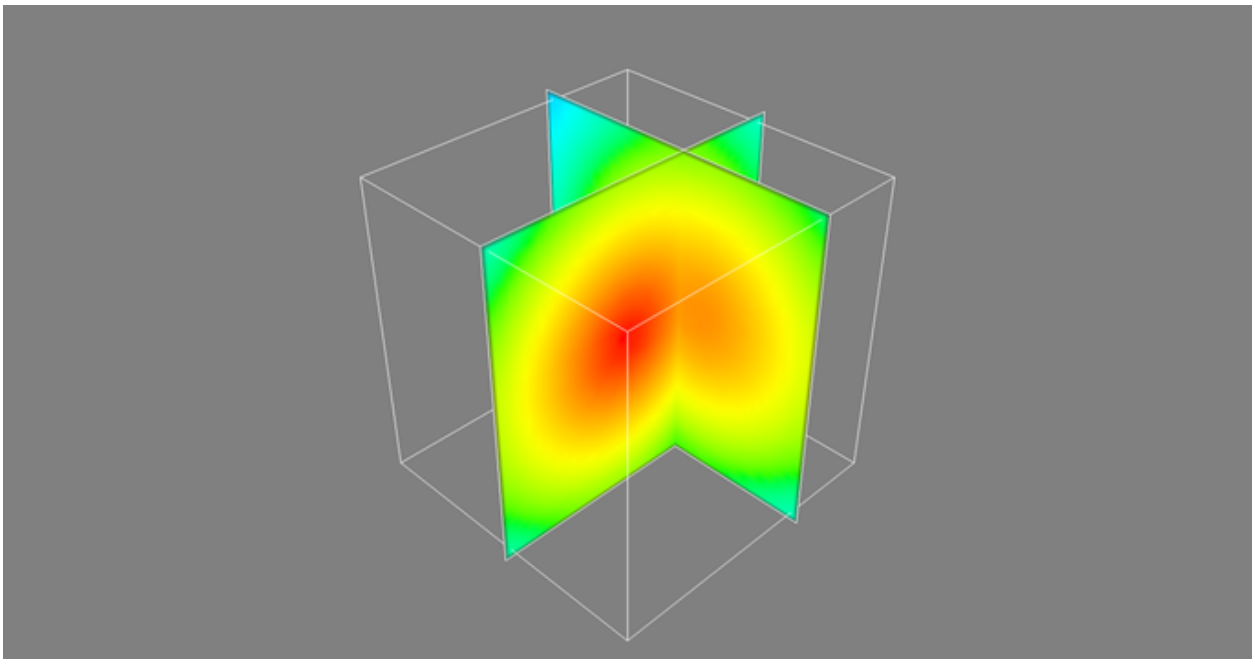

Figure 4.5: The same reconstruction visualized with mayavi

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/INCF/pykCSD/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

5.1.4 Write Documentation

pykCSD could always use more documentation, whether as part of the official pykCSD docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/INCF/pykCSD/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *pykCSD* for local development.

1. Fork the *pykCSD* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/pykCSD.git
   ```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

   ```
   $ mkvirtualenv pykCSD
   $ cd pykCSD/
   $ python setup.py develop
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

   ```
   $ flake8 pykCSD tests
   $ python setup.py test
   $ tox
   ```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, and 3.3. Check https://travis-ci.org/INCF/pykCSD/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_pykCSD
```

CHAPTER 6

Credits

6.1 Scientific Lead

- Daniel Wójcik

6.2 Development Lead

- Grzegorz Parka <grzegorz.parka@gmail.com>

6.3 Contributors

None yet. Why not be the first?

6.4 Base

This project is based on the Matlab implementation developed by Jan Potworowski.

CHAPTER 7

History

CHAPTER 8

Indices and tables

- *genindex*
- *modindex*
- *search*