# c2s Documentation

***Release 0.1.0dev***

**Lucas Theis**

December 23, 2016

Contents

A package for reconstructing spikes from calcium traces.

Contents:

CHAPTER 1

# Installation

Besides Python, NumPy, and Scipy, the c2s package depends on the Conditional Modeling Toolkit. Note that as it currently stands, this toolkit works best under Linux and is hard to get to work under Windows. After installing the toolkit, the c2s package can be installed by running the following on the command line:

```
$ pip install cython
$ pip install git+https://github.com/lucastheis/c2s.git
```

The first line (installing Cython) is optional but will enable more options for the evaluation of spike trains. After installation, the following command should be available and output some help:

```
$ c2s -h
```

CHAPTER 2

# Tutorial

To predict spikes from calcium traces we are going to run commands like the following:

```
$ c2s predict data.mat predictions.mat
```

This tutorial describes how to format and preprocess your data, and how to improve and evaluate predictions.

## 2.1 Data formatting

Inputs and outputs can be stored in either MATLAB or Python format.

### 2.1.1 Using MATLAB

When using MATLAB files, data needs to be stored in a cell array named `data`. Each entry of the cell array should be a `struct` containing at least the fields `calcium` and `fps`. Accessing the 12th entry of the cell array might present you with something like the following output:

```
>> data{12}
ans =
     calcium: [1x71985 double]
      spikes: [1x71985 uint16]
         fps: 99.9998
    cell_num: 8
```

Here, `calcium` is a 1xT vector containing the calcium or fluorescence trace, while `fps` is a `double` referring to the corresponding sampling rate in bins per second. Optionally, each entry may contain the fields `spikes`, `spike_times`, and `cell_num`. The field `spikes` should correspond to a binned spike train at the same sampling rate as the calcium trace. However, the preferred method is to specify the spikes times in milliseconds via `spike_times`, since spikes are typically recorded at much higher sampling rates. In this case the output might look like:

```
>> data{12}
ans =
        calcium: [1x71985 double]
    spike_times: [1x1202 double]
            fps: 99.9998
       cell_num: 8
```

The spikes can be used for training a statistical model to predict spikes. The `cell_num` field is used to group recordings together and will affect the leave-one-out cross-validation. This is useful, for example, if data from one cell was split into two or more sessions. If each entry corresponds to a different cell, this field can be ignored.

### 2.1.2 Using Python

In Python, data should be stored in lists of dictionaries and saved as pickled objects. Each dictionary element should contain at least the entries `calcium` and `fps`. Accessing the 12th entry of the list might present you with something like the following output:

```
>>> print data[12]
{'calcium': array([[ 0.391,  0.490, ...,  0.221,  0.307]]),
 'fps': 99.9998,
 'cell_num': 8,
 'spikes': array([[0, 0, 0, ..., 0, 0, 0]], dtype=uint16)}
```

Here, `calcium` is a 1xT NumPy array containing the calcium or fluorescence trace, while `fps` is a float value referring to the corresponding sampling rate in bins per second. Optionally, each dictionary may contain the entries `spikes`, `spike_times`, and `cell_num`. The entry `spikes` should correspond to a binned spike traini at the same sampling rate as the calcium trace. However, the preferred method is to specify the spikes times in milliseconds via `spike_times`, since spikes are typically recorded at much higher sampling rates. In this case the output might look like:

```
>>> print data[12]
{'calcium': array([[ 0.391,  0.490, ...,  0.221,  0.307]]),
 'fps': 99.9998,
 'cell_num': 8,
 'spike_times': array([[ 5951.34, 6007.95, ..., 719155.46, 719307.52]])}
```

The spikes can be used for training a statistical model to predict spikes. The `cell_num` entry is used to group recordings together and will affect the leave-one-out cross-validation. This is useful, for example, if data from one cell was split into two or more sessions. If each entry corresponds to a different cell, this field can be ignored. To save the data, use `pickle`,

```
>>> from pickle import dump
>>> with open('data.pck') as handle:
>>>     dump(data, handle, protocol=2)
```

## 2.2 Preprocessing

After the data has been brought into the right format, we should preprocess it.

```
$ c2s preprocess data.pck data.preprocessed.pck
```

If your data is stored in MATLAB files, use

```
$ c2s preprocess data.mat data.preprocessed.mat
```

The desired format is automatically inferred from the file ending. The preprocessing tries to remove linear trends from the calcium trace and up- or downsamples the data so that all traces have the same sampling rate. By default, this sampling rate is 100 fps but can be changed with

```
$ c2s preprocess --fps 100 data.mat data.preprocessed.mat
```

to something else if desired. Additionally, the preprocessing computes `spikes` from `spike_times` and *vice versa* if only one of the two is given.

---

**Note:** The default model used for making predictions assumes that the data has been preprocessed with the default parameters. In general, data should undergo the same preprocessing before training and prediction.

---

## 2.3 Predicting spikes

Predicting spikes is as easy as

```
$ c2s predict data.preprocessed.pck predictions.pck
```

As for the preprocessing, inputs and outputs can again be MATLAB files. If the data has not been preprocessed yet, use

```
$ c2s predict --preprocess 1 data.pck predictions.pck
```

The predictions are saved in the same format as the data files, except that the entries `spikes`, `spike_times` and `calcium` are removed to save space. By default, the prediction uses a model which has been trained on several datasets recorded by different labs under different conditions. These datasets combined contained roughly 110,000 spikes. But it is possible to train a model specifically for our data. Once trained, the model can be used for prediction as follows:

```
$ c2s predict -m model.xpck data.preprocessed.pck predictions.pck
```

## 2.4 Training a model

To train a model to fit your needs, use the command:

```
$ c2s train data.preprocessed.pck model.xpck
```

Multiple datasets can be combined as well:

```
$ c2s train data1.pck data2.pck model.xpck
```

To print a list of available parameters to influence the training, please see:

```
$ c2s train -h
```

## 2.5 Evaluation

Different metrics have been used to evaluate how well firing rate predictions agree with observed spike trains. c2s offers estimates of the mutual information, correlation, and area uner the ROC curve (AUC). These can be calculated with calls like the following:

```
$ c2s evaluate -m corr data.preprocessed.mat predictions.mat
$ c2s evaluate -m info data.preprocessed.mat predictions.pck
$ c2s evaluate -m auc  data.preprocessed.pck predictions.pck
```

**Note:** For the evaluation of AUC to work, Cython has to have been installed before installing c2s.

The mutual information interprets the prediction as Poisson firing rates and is the most stringent of the three. For predictions $\lambda_t$ and observed spike counts $k_t$, it is given by

$$I = \frac{1}{T} \sum_t k_t \log_2 \frac{\lambda_t}{\bar{k}} + \bar{k} - \bar{\lambda},$$

where $\bar{k}$ is the average over all $k_t$ and $\bar{\lambda}$ is the average over all $\lambda_t$. While correlation is invariant under affine transformations, i.e., multiplying the predictions by a factor or adding a constant to them does not change the performance, mutual information depends on the absolute predictions. On the other end of the spectrum, AUC is invariant under arbitrary strictly monotone functions, i.e., even tranforming the predictions in a nonlinear way will not change the performance. However, many methods developed for spike reconstruction from calcium images have not been developed with mutual information in mind. This is why by default all predictions are nonlinearly transformed by an optimal piecewise linear monotonically increasing function. I.e., the information is calculated using $\lambda'_t = f(\lambda_t)$ rather than $\lambda_t$. This optimization can be disabled as follows:

```
$ c2s evaluate -z 0 -m info data.preprocessed.pck predictions.pck
```

Since the evaluation is generally sensitive to the sampling at which the performance measure is calculated, the performance is calculated at various sampling rates. For example,

```
$ c2s evaluate -s 1 5 10 -m corr data.preprocessed.pck predictions.pck
```

will downsample the signals by the factors 1, 5, and 10 before performing an evaluation. I.e., if the given spike trains and predictions are sampled at 100 Hz, the evaluation will be performed at 100 Hz, 20 Hz, and 10 Hz.

If no predictions but only a dataset is given to the evaluation, the calcium traces are used as predictions instead. Correlations, for example, are then computed between the calcium trace and the spike train. This can be used as a baseline measure.

```
$ c2s evaluate data.preprocessed.pck
```

Finally, the results can be saved into MATLAB or pickled Python files via:

```
$ c2s evaluate -o correlation.mat  -m corr data.preprocessed.pck predictions.pck
$ c2s evaluate -o correlation.xpck -m corr data.preprocessed.pck predictions.pck
```

---

**Note:** Using the same data for training a model and evaluating the model performance will lead to overly optimistic performance estimates. To avoid bias, use independent datasets for training and evaluation or use *leave-one-out cross-validation* for generating predictions.

---

## 2.6 Leave-one-out cross-validation

Training and evaluating a model on the same dataset leads to biased performance results. On the other hand, naively splitting a dataset in two might leave us with too little data to properly train our model or evaluate it. Leave-one-out cross-validation maximizes the amount of available training data by using all but one cell for training and only the remaining cell for prediction and evaluation. By repeating this process – using a different cell for evaluation each time – we can nevertheless use the entire dataset in the evaluation. A call to

```
$ c2s leave-one-out preprocessed.mat predictions.mat
```

will generate predictions by training a model on $N - 1$ cells to predict the remaining cell. Since this means running the training process $N$ times for $N$ cells, this can take a while.

---

**Note:** If recordings from a single cell are split across multiple sessions, you should use `cell_num` to group sessions together.

---