

Inverse Current Source Density method in two dimensions: Inferring neural activation from multielectrode recordings

Supplementary material

Szymon Łęski, Klas H. Pettersen, Beth Tunstall,
Gaute T. Einevoll, John Gigg, Daniel K. Wójcik

March 4, 2011

The Inverse CSD method in two dimensions is implemented as a set of MATLAB scripts. The scripts were tested on MATLAB 7.4.0 (R2007a).

The scripts are published under the GNU General Public License. You can use, redistribute and/or modify them under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

If you use these scripts, please cite the article “Inverse Current Source Density method in two dimensions: Inferring neural activation from multielectrode recordings” (Neuroinformatics, 2011, doi:10.1007/s12021-011-9111-4).

1 Using the 2D iCSD tool

Together with the MATLAB scripts (described below) we provide a graphical viewer and three datasets (two model datasets — apical and basal excitatory input to a population of pyramidal cells, and one experimental — data recorded in the rat subiculum, see the paper for details). To prepare the datasets and launch the graphical viewer, run `RUNME.m` (later you can use `iCSD2Dtool.m` — this skips some steps which are only needed when the tool is run for the first time). You can choose one of the three provided datasets or create a new one using a workspace variable or an array stored in a MAT file (the format of the array is described below in Section 2). The viewer allows you to view potentials and CSD, choose between traditional CSD and different variants of inverse CSD, to adjust boundary conditions, and to try different values of the h parameter in case of iCSD. The resulting figures can be saved to a PNG file.

2 Using the MATLAB scripts

The first step is to calculate the appropriate matrix F . This is done using the scripts `initstep2d.m`, `initlin2d.m` and `initspline2d.m`, which assume nearest neighbor, linear or spline interpolation between the nodes, respectively. The results are saved to a `.mat` file in subdirectory `data`, hence they must be calculated only once. These scripts are invoked as

```
initlin2d('name', nx, ny, dx, dy, h)
initlin2d('name', nx, ny, dx, dy, h, dsp)
initlin2d('name', nx, ny, dx, dy, h, zprofile)
initlin2d('name', nx, ny, dx, dy, h, zprofile, dsp)
```

similarly for the other two variants. The first input argument, `'name'`, is the name of the file in which the results will be saved. The next four arguments describe the geometry of rectangular grid. If no optional argument `dsp` is given, then this is both the grid of electrodes and the grid spanning the CSD distribution. If the electrode grid is not exactly rectangular (or should be different from CSD grid for any other reason), then the geometry of the electrode grid can be specified in `dsp`. Note that the CSD grid will still be the rectangular grid given by `nx`, `ny`, `dx`, `dy`. The argument `h` is related to the spread of the CSD distribution in z direction. CSD distribution is specified using the optional argument `zprofile`, which may be `'step'` (default) or `'gauss'`. Use `help initlin2d` for more information (and similarly for all other scripts described here).

The “traditional” CSD (finite-difference approximation of two-dimensional Laplacian) is also implemented similarly. Here the syntax is

```
Finv = inittrad2d(nx, ny, dx, dy)
```

where the input arguments specify the geometry of the rectangular electrode grid. The output is not the matrix F (which is not well defined) but the differentiation operator which is analogous to F^{-1} . If the fifth argument `'vaknin'` is given, then the matrix `Finv` implements also the Vaknin procedure for obtaining CSD at the boundary.

The next step is to prepare the input data (potentials) in suitable format. The data should be given as a 3-D array of size `[nt, nx, ny]`, where `nt` is the number of time frames. (The array should be 3-D even if only one time frame is present.) Then the CSD at the nodes of the rectangular grid can be calculated using

```
csd = icsd2d(potentials, F);
csd = icsd2d(potentials, F, boundary);
csd = icsd2d(potentials, Finv, 'trad');
```

The first form (with two arguments) is used when the assumption is that CSD is non-zero only within the CSD grid (in xy -plane). The second form has to be used when one wants special treatment of boundary data, i.e. assumption of an additional layer of nodes with either zeros (`boundary = 'B'`) or duplicated

values (boundary = 'D'). In that case the matrix F has to be for appropriately larger grid (n_x+2 by n_y+2). The third form of invoking `icsd2d.m` is used for traditional CSD (with or without Vaknin procedure).

The last step is to interpolate the CSD values between the nodes. This is achieved most conveniently with the script `interp2d.m`, which takes care of all the subtleties related to special boundary treatment. The syntax is

```
out = interp2d(csd, VX, VY, method);
out = interp2d(csd, VX, VY, method, boundary);
```

The vectors `VX` and `VY` define points at which the interpolated values are calculated. The values are normalized, i.e. the values in `VX` take values from 1 to n_x , in `VY` — from 1 to n_y . The argument `method` specifies which interpolation method should be used and can be 'step' (nearest neighbor), 'lin' (linear), 'splinen' (natural spline) or 'splinem' (not-a-knot spline). The optional argument `boundary` may be 'B' or 'D'. Of course, both the interpolation method and the boundary treatment should be consistent with how the CSD was calculated!

The scripts: `bmatrix2d.m`, `gmatrix.m`, `rmatrix2d.m`, `nsplint2d.m`, `parseinpargs.m` and `espmatrices2d.m` are helper scripts and there should be no need to invoke them directly. Some basic documentation is available as comments in the respective m-files.

3 Examples

As an example we will calculate the CSD reconstruction from a test set of potentials. These are potentials generated by two-dimensional Gaussian sources (times a step function in z direction) described in detail in the next section. The 8×16 array of potentials can be loaded from file `data/example.mat`:

```
load data/example.mat
```

We will go through all the steps required to calculate and plot the interpolated CSD. First we calculate the F matrix for, say, linear interpolation:

```
nx = 8;
ny = 16;
dx = 0.2;
dy = 0.1;
h = 0.1;
initlin2d('example_lin', nx, ny, dx, dy, h);
```

We assumed rectangular 8×16 array of electrodes, with spacing of 0.2mm in x direction and 0.1mm in y direction. The CSD will have product structure: a linear function of x and y times a step function in z . The step function will be 1 for $-0.1 \leq z \leq 0.1$ and zero otherwise. Once the matrix F is calculated, we load it and calculate the CSD:

```
load data/example_lin.mat F
csd1 = icsd2d(potentials, F);
```

Then we interpolate the CSD:

```
VX = 1:0.1:nx;
VY = 1:0.1:ny;
csd2 = interp2d(csd1, VX, VY, 'lin');
```

and plot it:

```
clims = [-1 1]*max(abs(csd2(:)));
imagesc((VX-1)*dx, (VY-1)*dy, squeeze(csd2(1, :, :))', clims);
axis image
```

This example is available in a script `Demo_icsd2d.m`.

Another example: the same data, but now the inverse CSD method uses spline interpolation and different boundary conditions:

```
initspline2d('example_spline_B', nx+2, ny+2, dx, dy, h);
load data/example_spline_B.mat Fm
csd1_B = icsd2d(potentials, Fm, 'B');
csd1_D = icsd2d(potentials, Fm, 'D');
csd2_B = interp2d(csd1, VX, VY, 'spline', 'B');
csd2_D = interp2d(csd1, VX, VY, 'spline', 'D');
```