Detecting neural assemblies in calcium imaging data

In the following, we give a brief overview over the main functions of the repository, to generate datasets of surrogate calcium imaging data and to analyse such datasets for assembly activity. Some of functions for preprocessing of the datasets as well as most of the algorithms have been developed by other authors. Therefore, wherever third-party code was used, the corresponding functions can be found in a separate directory together with a README file which specifies the origin of these functions together with a list of functions which have been changed from the original. Within the functions, these changes were consistently marked with the string " % # ".

A complete list of third-party packages of functions is a follows in the order in which they are used:

- 'fitF0smoother' package by K. Podgorski, included with permission
- 'KalmanAll' package by K. Murphy (distributed with the 'fitF0smoother' package), included with permission
- 'peakfinder' package by N. C. Yoder, published under the FreeBSD License
- 'oopsi' package by J. Vogelstein, published under the Apache License 2.0
- 'Toolbox-Romano-et-al' package by S. Romano et al., published under the GNU General Public License v3.0
- 'toolbox' package by V. Lopes-dos-Santos, included with permission
- 'SVDEnsemble' package by S. Han, published under the GNU General Public License v3.0
- 'PyFIM' & 'psf+psr' package by C. Borgelt, published under the MIT License

Reference: J. Mölter, L. Avitan, and G. J. Goodhill. "Detecting neural assemblies in calcium imaging data". BMC Biology **16**:143 (2018) doi: 10.1186/s12915-018-0606-4.

All datasets analysed in this work can be regenerated using the script `GENERATE_ALL_DATASETS`. However, on a single desktop computer this may take several days or weeks and will require about 350 GB of disk space.

In the directory `examples/` we provide a fully analysed dataset of surrogate calcium imaging data as well as the fully analysed dataset of stimulus-evoked calcium imaging data from the optic tectum of a larval zebrafish studied in the work mentioned above.

Version: 30. April 2019

Simulation of population calcium fluorescence activity

```
ARTIFICIAL_CALCIUM_FLUORESCENCE_GENERATION( '~/' ,
    ↪ 'HEX5_K5_medium__T1800s_dT500ms_cT1s_sKInf_eF10.0mHz_eM6_nS0' , 0 ,
    ↪ 1800 , 0.5 , 'HEX5_K5_medium' , 1 , 1 , 0.01 , 6 , 0 , Inf )
```

Prerequisits of third-party packages: 'fitF0smoother', 'KalmanAll'

Output: `*_CALCIUM-FLUORESCENCE.mat`

- `calcium_fluorescence.F`: raw fluorescence signal [T $\times$ N matrix]
- `calcium_fluorescence.F0`: baseline fluorescence signal [T $\times$ N matrix]
- `calcium_fluorescence.dF_F`**: fluorescence signal (ΔF/F) [T $\times$ N matrix]**
- `topology`: neuron coordinates [N $\times$ 1 cell : 1 $\times$ d matrix]
- `parameter.units`**: number of neurons**
- `parameter.dT_step`**: temporal resolution / lenght of a time step (s)**
- `parameter.time_steps`**: number of time steps**
- `parameter.assembly_configuration`: underlying ground truth assembly configuration
- `parameter.rate_range`: background firing rate range (1/s)
- `parameter.eventDuration`: event duration (s)
- `parameter.eventFreq`: event frequency (1/s)
- `parameter.eventMult`: event firing rate multiplier
- `parameter.calcium_T1_2`**: calcium indicator half-life (s)**
- `parameter.saturation_K`: calcium fluorescence saturation constant
- `parameter.noiseSTD`: Gaussian noise standard deviation
- `meta_information`: meta-information about the dataset – optional

Note: For all subsequent analysis a structure as decribed above is expected for the modules to run smoothly. The minimal set of fields required is marked in bold.

Conversion from existing calcium fluorescence activity

```
CALCIUM_FLUORESCENCE_CONVERSION( '~/' ,
    ↪ 'HEX5_K5_medium__T1800s_dT500ms_cT1s_sKInf_eF10.0mHz_eM6_nS0' ,
    ↪ dF_F_field , 0.5 , 1 )
```

Output: `*_CALCIUM-FLUORESCENCE.mat`

Preprocessing of the calcium fluorescence activity

```
CALCIUM_FLUORESCENCE_PROCESSING( [ '~/'
    ↪ 'HEX5_K5_medium__T1800s_dT500ms_cT1s_sKInf_eF10.0mHz_eM6_nS0' '/'
    ↪ 'HEX5_K5_medium__T1800s_dT500ms_cT1s_sKInf_eF10.0mHz_eM6_nS0'
    ↪ '_CALCIUM-FLUORESCENCE.mat' ] )
```

Prerequisits of third-party packages: 'peakfinder', 'oopsi' , 'Toolbox-Romano-et-al'

Output: `*_ACTIVITY-RASTER.mat`

- `activity_raster`: binary rastered fluorescence signal [T $\times$ N matrix]
- `activity_raster_peak_threshold`: threshold for high population fluorescence activity
- `activity_raster_peaks`: time points of high population fluorescence activity [T $\times$ 1 matrix]

Output: `*_ACTIVITY-FIM-RASTER.dat`

Output: `*_SPIKE-PROBABILITY-RASTER.mat`

- `spike_probability_raster`: binary rastered spike probabilities [T $\times$ N matrix]
- `spike_probability_raster_thresholds`: threshold for high spike probability
- `oopsi_deconvolution`: fast nonnegative matrix deconvolution (OOPSI) results

Output: `*_RASTER.mat`

- `raster`: fluorescence signal (ΔF/F) binary mask [T $\times$ N matrix]
- `params`
- `deltaFoF`: fluorescence signal (ΔF/F) [T $\times$ N matrix]
- `deletedCells`
- `movements`
- `mu`
- `sigma`
- `imageAvg`
- `F0`
- `dataAllCells`

Detection of neural assemblies

Using the ICA-CS / ICA-MP algorithm

```
ICA_ASSEMBLY_DETECTION( [ '~/'
    ↪ 'HEX5_K5_medium__T1800s_dT500ms_cT1s_sKInf_eF10.0mHz_eM6_nS0' '/'
    ↪ 'HEX5_K5_medium__T1800s_dT500ms_cT1s_sKInf_eF10.0mHz_eM6_nS0'
    ↪ '_CALCIUM-FLUORESCENCE.mat' ] )
```

Prerequisits of third-party packages: 'toolbox'

Output: `*_ICA-ASSEMBLIES.mat`

- `cs_assembly_vectors`: assembly vectors from CS null model [1 × k cell]
- `cs_assemblies`: assemblies from CS null model [k × 1 cell]
- `ks_alpha`: KS-significance-level
- `mp_assembly_vectors`: assembly vectors from MP null model [1 × k cell]
- `mp_assemblies`: assemblies from MP null model [k × 1 cell]

Using the Promax-MP algorithm

```
PROMAX\_MP_ASSEMBLY_DETECTION( [ '~/'
    ↪ 'HEX5_K5_medium__T1800s_dT500ms_cT1s_sKInf_eF10.0mHz_eM6_nS0' '/'
    ↪ 'HEX5_K5_medium__T1800s_dT500ms_cT1s_sKInf_eF10.0mHz_eM6_nS0'
    ↪ '_RASTER.mat' ] )
```

Prerequisits of third-party packages: 'Toolbox-Romano-et-al'

Output: `*_PROMAX-MP-ASSEMBLIES.mat`

- `threshSignifMatchIndex`
- `matchIndexTimeSeries`
- `assembliesCells`: assemblies [1 × k cell]
- `matchIndexTimeSeriesSignificant`
- `matchIndexTimeSeriesSignificantPeaks`
- `matchIndexTimeSeriesSignificance`
- `clustering`
- `assembliesVectors`: assembly vectors [N × k matrix]
- `PCsRot`: assembly vectors (rotated principal components) [N × k matrix]
- `confSynchBinary`
- `zMaxDensity.x`: zMax value [1 × ? matrix]

- `zMaxDensity.densityNorm`: zMax density function [1 × ? matrix]
- `zMaxDensity.normCutOff`: zMax cut-off value

Using the Promax-CS algorithm

```
PROMAX\_CS_ASSEMBLY_DETECTION( [ '~/'
    ↪ 'HEX5_K5_medium__T1800s_dT500ms_cT1s_sKInf_eF10.0mHz_eM6_nS0' '/'
    ↪ 'HEX5_K5_medium__T1800s_dT500ms_cT1s_sKInf_eF10.0mHz_eM6_nS0'
    ↪ '_RASTER.mat' ] )
```

Prerequisits of third-party packages: 'toolbox', 'Toolbox-Romano-et-al'

Output: `*_PROMAX-MP-ASSEMBLIES.mat`

- `threshSignifMatchIndex`
- `matchIndexTimeSeries`
- `assembliesCells`: assemblies [1 × k cell]
- `matchIndexTimeSeriesSignificant`
- `matchIndexTimeSeriesSignificantPeaks`
- `matchIndexTimeSeriesSignificance`
- `clustering`
- `assembliesVectors`: assembly vectors [N × k matrix]
- `PCsRot`: assembly vectors (rotated principal components) [N × k matrix]
- `confSynchBinary`
- `zMaxDensity.x`: zMax value [1 × ? matrix]
- `zMaxDensity.densityNorm`: zMax density function [1 × ? matrix]
- `zMaxDensity.normCutOff`: zMax cut-off value

Using the CORE algorithm

```
CORE_ASSEMBLY_DETECTION( [ '~/'
    ↪ 'HEX5_K5_medium__T1800s_dT500ms_cT1s_sKInf_eF10.0mHz_eM6_nS0' '/'
    ↪ 'HEX5_K5_medium__T1800s_dT500ms_cT1s_sKInf_eF10.0mHz_eM6_nS0'
    ↪ '_SPIKE-PROBABILITY-RASTER.mat' ] )
```

Output: `*_CORE-ASSEMBLIES.mat`

- `ensemble_detection`
- `assemblies`: assemblies [1 × k cell]

Using the SVD algorithm

```
SVD_ASSEMBLY_DETECTION( [ '~/'
    ↪ 'HEX5_K5_medium__T1800s_dT500ms_cT1s_sKInf_eF10.0mHz_eM6_nS0' '/'
    ↪ 'HEX5_K5_medium__T1800s_dT500ms_cT1s_sKInf_eF10.0mHz_eM6_nS0'
    ↪ '_SPIKE-PROBABILITY-RASTER.mat' ] )
```

Prerequisits of third-party packages: 'SVDEnsemble'

Output: `*_SVD-ASSEMBLIES.mat`

- `assemblies`: assemblies [1 × k cell]
- `assemblies_activation`: assembly activation time series [T × 1 matrix]

Using the SGC algorithm

```
SGC_ASSEMBLY_DETECTION( [ '~/'
    ↪ 'HEX5_K5_medium__T1800s_dT500ms_cT1s_sKInf_eF10.0mHz_eM6_nS0' '/'
    ↪ 'HEX5_K5_medium__T1800s_dT500ms_cT1s_sKInf_eF10.0mHz_eM6_nS0'
    ↪ '_ACTIVITY-RASTER.mat' ] )
```

Output: `*_SGC-ASSEMBLIES.mat`

- `assembly_pattern_detection.activityPatterns`: activity patterns of high population fluorescence activity [? × 1 cell]
- `assembly_pattern_detection.patternSimilarityAnalysis.graph`: activity pattern similarity graph [1 × 1 graph]
- `assembly_pattern_detection.patternSimilarityAnalysis.communityStructure`: similarity graph community-structure analysis results
- `assembly_pattern_detection.assemblyActivityPatterns`: assembly activity patterns [k × 1 cell]
- `assembly_pattern_detection.assemblyIActivityPatterns`: activity patterns indices for every assembly activity pattern [k × 1 cell]
- `assemblies`: assemblies [k × 1 cell]

Using the FIM-X algorithm (Python!)

```
! ( cd ./assembly-detection/FIM_PSF_PSR_ASSEMBLY_DETECTION/ &&
   ↪ ./py__FIM_PSF_PSR_ASSEMBLY_DETECTION.sh
   ↪ ~/HEX5_K5_medium__T1800s_dT500ms_cT1s_sKInf_eF10.0mHz_eM6_nS0/HEX5_K5_medium__T1
   ↪ )
```

Prerequisits of third-party packages: 'PyFIM', 'psf+psr'

Output: `*_FIM-PSF-PSR-ASSEMBLIES.dat`

Collection of the results from the different algorithm for postprocessing

```
ASSEMBLIES_COLLECTION( [ '~/'
   ↪ 'HEX5_K5_medium__T1800s_dT500ms_cT1s_sKInf_eF10.0mHz_eM6_nS0' '/'
   ↪ 'HEX5_K5_medium__T1800s_dT500ms_cT1s_sKInf_eF10.0mHz_eM6_nS0'
   ↪ '_CALCIUM-FLUORESCENCE.mat' ] )
```

Output: `*_ASSEMBLIES-COLLECTION.mat`

- `parameter`: simulation parameters
- `meta_information`: meta-information about the dataset – optional
- `topology`: neuron coordinates [N $\times$ 1 cell : 1 $\times$ d matrix]
- `ICA_CS_assemblies`: assemblies as inferred from the ICA-CS algorithm [1 $\times$ k cell]
- `ICA_MP_assemblies`: assemblies as inferred from the ICA-MP algorithm [1 $\times$ k cell]
- `PROMAX_MP_assemblies`: assemblies as inferred from the Promax-MP algorithm [1 $\times$ k cell]
- `PROMAX_CS_assemblies`: assemblies as inferred from the Promax-CS algorithm [1 $\times$ k cell]
- `CORE_assemblies`: assemblies as inferred from the CORE algorithm [1 $\times$ k cell]
- `SVD_assemblies`: assemblies as inferred from the SVD algorithm [1 $\times$ k cell]
- `SGC_assemblies`: assemblies as inferred from the SGC algorithm [1 $\times$ k cell]
- `FIM_PSF_PSR_assemblies`: assemblies as inferred from the FIM-X algorithm [1 $\times$ k cell]