Workflow

The following steps indicate the standard workflow for demixing and deconvolving 2d calcium imaging data, as highlighted in the demo file, *demo_script.m*

1. Read data: Use *bigread2.m* for stack tiff files, or Matlab's *h5read.m* for hdf5 files.
2. Set key parameters: Number of components, size of Gaussian kernel during initialization, order of autoregressive process. Set additional parameters through *CNMFSetParms.m*
3. Pre-process data with *preprocess_data.m*
4. Run Initialization through *initialize_components.m*
5. Manually inspect and add/remove components through *manually_refine_components.m* (optional)
6. Run 1-2 CNMF iterations:
    - Update spatial components *update_spatial_components.m*
    - Update temporal components *update_temporal_components.m*
    - Merge correlated overlapping components *merge_components.m*
7. Sort identified components and extract DF/F values
8. Display components

Handling Big Data

For handling big datasets we recommend splitting the data into partially overlapping spatial patches, process them in parallel and then combine the results. This process is implemented by the function *run_CNMF_patches.m* and is described in more detail at the repository's wiki.

Notation

The table below contains notation and description of the main variables used in the algorithm.

Variable	Dimensions	Description
d1	Positive integer	# of pixels in x dimension
d2	Positive integer	# of pixels in y dimension
d3	Positive integer	# of pixels in z dimension (in not present then d3 = 1)
d	Positive integer	Total # of pixels (d = d1*d2*d3)
T	Positive integer	# of timesteps (frames)
Y	d x T (d1 x d2 x d3x T) double matrix	Input Data
K	Positive integer	# of source components
A	d x K double matrix (sparse)	Spatial components
C	K x T double matrix	Temporal components
S	K x T double matrix	Neural activity matrix (Spikes)
nb	Positive integer	# of background components (usually nb = 1)
b	d x nb double matrix	Spatial profile of background
f	nb x T double matrix	Temporal profile background
P	struct	Structure that stores dataset and component properties
sn	d x 1 double vector	Noise level for each pixel
p	Positive scalar	Order of autoregressive system
g	p x 1 double vector	Discrete time constants
tau	Positive scalar	Standard deviation of Gaussian kernel for initialization

preprocess_data.m

Inputs			Outputs		
Name	**Dimensions**	**Description**	**Name**	**Dimensions**	**Description**
Y	d1 x d2 x T (double matrix)	Input Data	P	Struct	Structure that stores dataset and component properties
p	Positive integer	Order of AR process	Y	d1 x d2 x T (double matrix)	Data with missing entries interpolated
options	Struct	Options structure			

Description: This function performs the following pre-processing operations in the input dataset:

1. Data interpolation: Missing entries from *Y* (identified as NaN values) are stored in *P.mis_entries* and are interpolated. The interpolated values are stored in *P.mis_values* and are also included in *Y*.
2. Saturated pixels are identified. Pixels that are not saturated are stored in *P.pixels*
3. Noise level for each pixel is computed using a power spectral density method and is stored in *P.sn* (see documentation for *constrained_foopsi.m below for more info)*.
4. The AR order *p* is copied into *P.p*
5. Square root of the Power Spectral Density (PSD) of each pixel *P.psdx* restricted to the first 1500 frequency bins.
6. Classification of pixels into active *P.active_pixels* or inactive by clustering the PSD of each pixel.
7. Global discrete time constants (of order *p)* are computed if needed (see documentation for *constrained_foopsi.m below for more info)*.

Called functions: This function calls *interp_missing_data.m* for missing data interpolation, *find_unsaturatedPixels.m* for saturated pixels identification, and *get_noise_fft.m* for noise level estimation.

Tunable parameters: The following option parameters can be set from the user using *CNMFSetParms.m* (check options settings for *constrained_foopsi.m* for settings regarding noise and time constant estimation):

Name	Description	Default value

options.flag_g	Flag for computing global time constants	0
options.split_data	Flag for computing noise values sequentially for memory reasons	0

initialize_components.m

Inputs			Outputs		
Name	**Dimensions**	**Description**	**Name**	**Dimensions**	**Description**
Y	d1 x d2 x d3 x T (double matrix)	Input Data	Ain	d x K (sparse matrix)	Spatial Components
K	Positive integer	Number of components	bin	d x nb (double matrix)	Spatial Background
tau	Positive number	Standard deviation of Gaussian kernel	Cin	K x T (double matrix)	Temporal Components
options	Struct	Options structure	fin	nb X T (double matrix)	Temporal Background
			center	K x 2 (double matrix)	Component centroids

Description: This function produces initial estimates for the spatial and temporal components and background. There are two ways to initialize the components and can be chosen from *options.init_method*:

1. 'greedy': This method after performing spatial filtering with a Gaussian kernel of standard deviation *tau* selects greedily the locations where the spatial estimates explain the largest amount of spatio-temporal variance, and runs a rank-1 local NMF to produce estimates. This is the default option and is recommended for somatic imaging.
2. 'sparse_NMF': This method performs sparse NMF to initialize the components and is recommended for dendritic and axonal imaging data.

These estimates are subsequently refined by using HALS, a hierarchical alternating NMF method. After all the components have been found, a rank *nb* NMF is run on the spatio-temporal residual to initialize the spatial *b* and temporal *f* background. To efficiently process large datasets the algorithm (optionally) down-samples both spatially and temporally before the greedy method and up-samples after HALS. Note that the input data *Y*, is passed as 3-way x-y-t tensor with time appearing in the last dimension.

- The 'greedy' algorithm allows for components with different sizes to be found. To do this, specify tau as a L x 2 matrix with L the number of different types of components, and each row corresponds to the standard deviation for each size, and K as a Lx1 vector with the number of components for each type. Sort K in a decreasing way, i.e., large components to be found first.
- The 'greedy' and 'sparse_NMF' methods can be combined. In this case apply the greedy method first, compute the residual including the background components and then apply the sparse_NMF_method.

Called functions: This function calls *greedyROI.m* for greedy initialization, *sparse_NMF_initialization,* for initializing with a sparse NMF, and *HALS.m* for the final refinement.

Tunable parameters: The following option parameters can be set from the user using *CNMFSetParms.m*:

Name	Description	Default value
options.ssub	Spatial down-sampling factor (scalar >= 1)	1
options.tsub	Temporal down-sampling factor (scalar >= 1)	1
options.nb	Number of background components (positive integer)	1
options.gSig	Size of Gaussian kernel	2*tau+1
options.save_memory	Perform spatial filter in patches sequentially to save memory (binary)	0 (not needed, use subsampling)
options.maxIter	Maximum number of HALS iterations	5
options.bSiz	Expansion factor for HALS localized updates	3
options.snmf_max_iter	Maximum number of sparse NMF iterations	100
options.err_thr	Relative change threshold for stopping sparse_NMF	1e-4
options.eta	Weight on frobenius norm of temporal components * max(Y)^2	1
options.beta	Weight on squared l1 norm of spatial components	0.5

run_CNMF_patches.m

Inputs			Outputs		
Name	**Dimensions**	**Description**	**Name**	**Dimensions**	**Description**
Y	double matrix or .mat file	Input Data	A	d x N (sparse)	Spatial Components
K	Positive integer	Number of components per patch	b	d x nb (double)	Spatial Background
patches	Cell array	Coordinates (start and end points) for each patch	C	N x T (double)	Temporal Components
tau	Positive	Standard deviation of Gaussian kernel	f	nb X T (double)	Temporal Background
p	Positive integer	Order of AR dynamics	S	N x T (double)	Neural Activity (Spikes)
options	Struct	Options structure	P	struct	Neuron Parameters
			RESULTS	struct array	Results in each patch
			YrA	N x T (double)	Traces for each component + noise

Description: This function is designed to process large datasets by processing partially overlapping patches in parallel and then combining the results. The function operates on datasets that have been saved as a matlab .mat file and reads them directly from the hard drive without loading the full dataset on memory. This enables the processing of large datasets. The process is described in more detail in the wiki. For an example on how to use this function consult the file *demo_memmap.m*.

update_spatial_components.m

Inputs			Outputs		
Name	**Dimensions**	**Description**	**Name**	**Dimensions**	**Description**
Y	d x T (double matrix)	Input Data	A	d x K (sparse matrix)	Spatial Components
C	K x T (double matrix)	Temporal Components	b	d x nb (double matrix)	Spatial Background
f	nb x T (double matrix)	Temporal Background	C	K x T (double matrix)	Temporal Components
A_	d x K (sparse matrix)	Current estimate of spatial components			
P	struct	Neuron parameter structure			
options	struct	options structure			

Description: This function provides estimates of the spatial components *A* and background *b*, given the temporal component *C* and background *f,* and the noise values for each pixel stored in the vector *P.sn.* If the data *Y* contain interpolated values from missing entries, then these entries are excluded from the estimation. The problem is solved is solved in a pixel by pixel basis and can be solved in parallel. Each new component is constrained to be zero outsize a component specific component region, that is computed by expanding the spatial support of the same component at the previous iteration (as specified by *A_*). The new components are also post-processed by (i) median filtering, (ii) morphological closing and (iii) energy thresholding.

Big data handling: The input data *Y* can be either a double matrix or a pointer to a matlab .mat file that contains the data.

Called functions:
The search location of each component is determined from the function *determine_search_location.m* There are two method for determining the search location:

1. 'ellipse': In this case an ellipse is constructed around the center of mass and the 2 principal components of the component in the previous iteration, and the search location corresponds to an expanded version of this ellipse.
2. 'dilate': In this case the search location is computed by dilating the support set of the component from the previous iteration with a user defined morphological element.

For somatic imaging data and both methods can be used, whereas in the case of the dendritic or axonal imaging, ‘dilate’ is recommended due to the non-ellipsoid shape of the components.

Component post-processing is implemented with the function *threshold_components.m* The user can define the parameters of the post-processing options. At the end, the algorithm checks and removes any empty spatial components and their temporal counterparts.

Tunable parameters: The following option parameters can be set from the user using *CNMFSetParms.m*:

Name	Description	Default value
options.use_parallel	Flag for solving optimization problem in parallel (binary)	1 (if parallel toolbox exists)
options.search_method	Method for computing search locations (‘ellipse’ or ‘binary’)	‘ellipse’
options.min_size	Minimum size of ellipse axis (positive, real)	3
options.max_size	Maximum size of ellipse axis (positive, real)	8
options.dist	Ellipse expansion factor (positive, real)	3
options.se	Morphological element for method ‘dilate’ (binary image)	strel(‘disk’,4,0)
options.nrgthr	Energy threshold (positive between 0 and 1)	0.99
options.clos_op	Morphological closing operator for post-processing (binary image)	strel(‘square’,3)
options.medw	Size of 2-d median filter (2 x 1 array of positive integers)	[3,3]

update_temporal_components.m

Inputs			Outputs		
Name	**Dimensions**	**Description**	**Name**	**Dimensions**	**Description**
Y	d x T (double matrix)	Input Data	C	K x T (double matrix)	Temporal Components
A	d x K (sparse matrix)	Spatial Components	f	nb x T (double matrix)	Temporal Background
b	d x nb (double matrix)	Spatial Background	P	Struct	Neuron parameters
Cin	K x T (sparse matrix)	Current estimate of temporal components	S	K x T (double matrix)	Neural activity (spiking)
fin	nb x T (sparse matrix)	Current estimate of temporal background	YrA	K x T (double matrix)	Traces for each component + noise
P	struct	Neuron parameters			
options	struct	Parameter structure			

Description: This function estimates the temporal components *C* and background *f,* given the spatial components *A* and background *b.* The function also outputs the deconvolved neural activity matrix *S,* and a parameter struct *P,* that contains information about each neuron. The components are updated using a block-coordinate descent setup where the activity of each component is deconvolved from the residual signal after removing the effect of all the other components. The updating occurs in parallel by default by partitioning the set of components through a series of randomized graph vertex covers. Within each subset of the partition the components can be updated in parallel. It can be switched off by *options.temporal_paralllel.* There are different methods available for deconvolution:

1. '*project'*: In this case the raw trace is just projected on the cone induced by the indicator dynamics.
2. *'constrained_foopsi'*: The algorithm uses the noise constrained deconvolution method, see more details above (default).
3. '*MCEM_foopsi'*: The algorithm iteratively alternates between *constrained_foopsi* and a MCMC method that is used to update the time constant (indicator dynamics).
4. *'MCMC'*: The fully Bayesian deconvolution method is applied to obtain spike estimates in continuous time
5. *'noise_constrained'*: A spatio-temporally noise constrained method is applied (slow, deprecated).

update_temporal_components_parallel.m will be removed in a future release.

Big data handling: The input data *Y* can be either a double matrix or a pointer to a matlab .mat file that contains the data.

Tips:

- The *P* struct in the input must contain the order of the autoregressive system *P.p*. If no order is specified then by default an AR(2) process is estimated.
- The *P* struct in the output contains the following parameters for each neuron: *P.b* (baseline), *P.c1* (initial value), *P.neuron_sn* (noise level at each neuron), and *P.gn* (discrete time constant for each neuron).
- In general the *'constrained_foopsi'* method is recommended. If higher precision and/or uncertainty quantification is required then use *'MCMC'* (at a considerable higher computational cost).
- If the order of the autoregressive process is 0, then the algorithm simply performs thresholding at 0 of the raw traces, reminiscent to the HALS procedure for NMF.
- Any deconvolution method that takes as an input the (averaged) raw traces and outputs the deconvolved activity can be incorporated by the function, by appropriately inserting a function in the switch case command of *update_temporal_components.m*
- Input arguments *Cin* and *fin* correspond to the estimates of temporal components and background from the previous iteration, and should be used if available to speed up convergence of the block-coordinate descent. However, they can be left empty in which case, the algorithm automatically initializes these estimates. This is important in the case when the locations are known (from a previous experiment) and the deconvolution is needed for a new dataset with the same field of view.

Tunable parameters: The following option parameters can be set from the user using *CNMFSetParms.m*:

Name	Description	Default value
options.deconv_method	Deconvolution method (see above for the 5 choices)	'constrained_foopsi'
options.restimate_g	Flag for re-estimating time constants of each neuron (binary)	1
options.temporal_iter	Max number of outer iterations of the block-coordinate descent (positive integer)	2
options.temporal_parallel	Flag for updating the temporal components in parallel	True

constrained_foopsi.m

Note: This function is also maintained at the repository: https://github.com/epnev/constrained-foopsi

Inputs			Outputs		
Name	**Dimensions**	**Description**	**Name**	**Dimensions**	**Description**
y	Tx1 (double vector)	Input Data	c	Tx1 (double vector)	Denoised activity
b	real scalar	baseline	b	real scalar	baseline
c1	Nonnegative scalar	Initial concentration	c1	Nonnegative scalar	Initial concentration
g	p x 1 (double vector)	Discrete time constants	g	p x 1 (double vector)	Discrete time constants
sn	Positive scalar	Noise standard deviation	sn	Positive scalar	Noise standard deviation
options			sp	Tx1 (double vector)	Denoised activity (spikes)

Description: This function extracts the denoised neural activity *c*, and deconvolved activity (spikes) *sp,* from the input data *y* by using a noise constrained deconvolution (CD) approach. The baseline *b,* initial concentration *c1,* discrete time constant *g,* and noise level *sn* can be given as inputs, otherwise they are estimated by the algorithm.

- The noise level *sn* is estimated by averaging the power spectral density (PSD) of *y* over a range of high frequencies.
- The discrete time constants are estimated from the sample autocorrelation function of *y*.
- The solution can be further re-sparsed by using an iterative re-weighted l1 norm approach (see *options.resparse*).
- The order of the AR system is read from *options.p* or from *P.g.* If none are present then is set to the default value p = 2.
- Note the slight abuse of notation here, since *b* here refers to the constant baseline for the trace, whereas *b* in the context of source extraction refers to the spatial background component.

There are 4 available method to solve the constrained deconvolution problem:

1. *'dual'*: uses a dual ascent method, requires the matlab optimization package (slow, not recommended)
2. *'cvx'*: uses the CVX optimization toolbox available from http://www.cvxr.com/ (default). Please follow cvx installation instructions. *cvx_setup.m* must be run (only once) before running CNMF.
3. *'spgl1'*: uses the SPGL1 method and requires its matlab implementation (recommended for low of mid-SNR traces)
4. *'lars'*: uses the LARS regression algorithm (recommended only for short traces with sparse spiking)

Tunable parameters: The following option parameters can be set from the user either manually or using *CNMFSetParms.m*:

Name	Description	Default value
options.p	Order of AR system (positive integer)	2
options.method	Method for solving the CD problem (see above for options)	'cvx'
options.bas_nonneg	Option for setting b nonnegative, otherwise b >= min(y) (binary)	1 (b>=0)
options.noise_range	Range of normalized frequencies over which to average PSD (2 x1 vector)	[0.25,0.5]
options.noise_method	Method to average PSD to reduce variance	'logmexp'
options.lags	Number of autocorrelation lags to be used for estimating *g* (positive integer)	5
options.resparse	Number of times to resparse obtained solution (nonnegative integer)	0 (no re-sparsening)
options.fudge_factor	Multiplicative bias correction for *g* (positive between 0 and 1). Note: Slight changes can have large effects. Typically stay within [0.95-1].	1 (no correction)

merge_components.m

Inputs			Outputs		
Name	**Dimensions**	**Description**	**Name**	**Dimensions**	**Description**
Y	d x T (double matrix)	Input Data	A	d x K (sparse matrix)	Spatial Components
A	d x K (sparse matrix)	Spatial Components	C	K x T (double matrix)	Temporal Components
b	d x nb (double matrix)	Spatial Background	K	Positive integer	Number of components
C	K x T (sparse matrix)	Current estimate of temporal components	merged_ROIs	struct	List of merged components
f	nb x T (sparse matrix)	Current estimate of temporal background	P	struct	Neuron parameters
P	struct	Neuron parameters	S	K x T (double matrix)	Neural activity (spiking)
S	K x T (double matrix)	Neural activity (spiking)			
options	Struct	Parameter structure			

Description: The function merges spatially overlapping components whose temporal components have correlation coefficients above a certain threshold, defined by *options.merge_thr.* When such a pair is identified, the contribution of the to-be-merged is added to the residual, and *update_spatial_components, update_temporal_components* are used to determine the spatial and temporal parts of the new merged component. The matrices *A,C,S* and the parameter struct *P,* are accordingly updated. The struct *merged_ROIs* stores the indeces of the components that were merged.

Tunable parameters: The following option parameters can be set from the user using *CNMFSetParms.m*:

Name	Description	Default value
options.merge_thr	Merging threshold (positive between 0 and 1)	0.85
options.fast.merge	Flag for performing fast merging (boolean)	1

order_ROIs.m

Inputs			Outputs		
Name	**Dimensions**	**Description**	**Name**	**Dimensions**	**Description**
A	d x K (sparse matrix)	Spatial Components	A_or	d x K (sparse matrix)	Spatial Components
C	K x T (double matrix)	Current estimate of temporal components	C_or	K x T (double matrix)	Current estimate of temporal components
S	K x T (double matrix)	Neural activity (spiking)	S_or	K x T (double matrix)	Neural activity (spiking)
P	struct	Neuron parameters	P_or	struct	Neuron parameters
			srt	K x 1 vector	order of components

Description: This function orders the components based on their size and maximum temporal activation. The activity matrix *S* and parameter structure *P* are also ordered.

extract_DF_F.m

Inputs			Outputs		
Name	**Dimensions**	**Description**	**Name**	**Dimensions**	**Description**
Y	d x T (double matrix)	Input data	C_df	(K+nb) x T (double matrix)	Temporal Components in DF/F domain
A	d x (K+nb) (sparse matrix)	Spatial Components (including background)	Df	(K+nb) x 1 (double vector)	Baseline values (F) for each component
C	(K+nb) x T (double matrix)	Temporal Components (including background)	S_df	K x T (double matrix)	Neural activity (spiking) in DF/F domain
S	K x T (double matrix)	Neural activity (spiking)			
i	Integer 1 <= i <= K+nb	Index of background			

Description: The function extracts the DF/F values for all the extracted components. The background component is passed together with the identified components and an index i is provided to indicate the background components. If i is not present, then it is estimated. Currently only nb = 1 is supported.

plot_contours.m

Inputs			Outputs		
Name	**Dimensions**	**Description**	**Name**	**Dimensions**	**Description**
Aor	d x K (sparse matrix)	Spatial Components (ordered)	CC	Struct	Coordinates of each component
Cn	d1 x d2 (double matrix)	Summary image (e.g., mean, correlation)	jsf	Struct	Struct in json format
thr	Positive, 0 < thr <= 1	Contour plot threshold (optional)			
display_numbers	Binary	Show number of component (optional)			
max_number	Positive integer <= K	Maximum number to display			
Coor	Struct array	Coordinates of each component			

Description: The function plots contours of the extracted components *Aor* against a summary statistic, e.g., mean image, or correlation image (which can be obtained through *correlation_image.m*). The level of the contour plot is determined by *thr*: Each contour is determined such that it captures a *thr* fraction of the total component energy. Precomputed coordinates can also be passed as the last argument of the function *Coor* for faster replotting.

manually_refine_components.m

Inputs			Outputs		
Name	**Dimensions**	**Description**	**Name**	**Dimensions**	**Description**
Y	d x T (double matrix)	Input data	A	double matrix	Updated spatial components
A	d x K (double matrix)	Spatial Components	C	double vector	Updated temporal components
C	K x T (double matrix)	Temporal Components (including background)	newcenters	double matrix	Updated centers
center	K x 2 (double matrix)	center of current components (optional)			
img	d1 x d2 (double matrix)	background image (optional)			
sx	scalar (integer)	half size of window			
options	struct	options structure			

Description: The function opens an interactive environment where the user can interactively add new components (by left clicking on the spot where the new component is desired) or remove existing components by right clicking at their center. The new components are estimated using the greedy approach with one component centered at the picked location.

Sources2D.m

Class wrapper around the main functions described above. See *demo_script_class.m* for a usage example

Variables correspondence		**Methods correspondence**	
obj.A	A	*Sources2D*	Object initialization
obj.C	C	*updateParams*	*CNMFSetParms.m*
obj.b	b	*preprocess*	*preprocess_data.m*
obj.f	f	*initComponents.m*	*initialize_components.m*
obj.S	S	*updateSpatial*	*update_spatial_components.m*
obj.P	P	*updateTemporal*	*update_temporal_components.m*
obj.Coor	Coor	*merge*	*merge_components.m*
		extractDF_F	*extract_DF_F.m*
		orderROIs	*order_ROIs.m*
		plotContours	*plot_contours.m*
		plotComponents	*view_components.m*
		plotComponentsGUI	*plot_components_GUI.m*
		refineComponents	*manually_refine_components.m*
		makePatchVideo	*make_patch_video.m*
		snapshot	Obtain results at the current state

Acknowledgements

This Matlab code is primarily developed by Eftychios A. Pnevmatikakis with useful input and comments from

- Josh Merel (MCMC implementation, not presented here)
- Daniel Soudry (PSD analysis)
- Yuanjun Gao (greedyROI2d)
- Pengcheng Zhou (HALS implementation, class wrapper)
- Weijian Yang and Darcy Peterka (bigread2.m, exclusion of saturated pixels, 'dilate' expansion, extensive beta testing)
- Andrea Giovannucci (visualization tools)

Special thanks to a number of early users and beta-testers including: Adam Packer, Lloyd Russell, Ben Deverett, Farzaneh Najafi, Matt Kaufman, Tim Machado, Clay Lacefield, Lucas Theis, David Greenberg, Jeffrey Stirman, Erik Flister, Michael Orger, Selmaan Chettih, Mai Morimoto, Marina Martinez Garcia.

Contact

For questions about the code, please use the public chat room https://gitter.im/epnev/ca_source_extraction
To report a bug, please create an issue on github https://github.com/epnev/ca_source_extraction/issues
Contributions are very welcome using fork and pull requests.

References

Pnevmatikakis, E.A., Soudry, D., Gao, Y., Machado, T., Merel, J., ... & Paninski, L. (2016). Simultaneous denoising, deconvolution, and demixing of calcium imaging data. *Neuron* 89(2):285-299, http://dx.doi.org/10.1016/j.neuron.2015.11.037

Pnevmatikakis, E. A., Gao, Y., Soudry, D., Pfau, D., Lacefield, C., Poskanzer, K., ... & Paninski, L. (2014). A structured matrix factorization framework for large scale calcium imaging data analysis. *arXiv preprint arXiv:1409.2903*.

Friedrich J., Soudry D., Mu Y., Freeman J., Ahrens M., and Paninski L. (2015). Fast constrained non-negative matrix factorization for

whole-brain calcium imaging data. *NIPS workshop on statistical methods for understanding neural systems*.