UFARSA User Guide

Author: Vahid Rahmati, TU Dresden (March, 2018)

The provided Matlab codes implement the UFARSA method (Ultra-fast Accurate Reconstruction of Spiking Activity). UFARSA is a novel, heuristic model-free-type method which enables an ultra-fast, accurate, near-automatic reconstruction of spiking activities from noisy calcium imaging data. UFARSA has been introduced in this paper:

"**Ultra-Fast Accurate Reconstruction of Spiking Activity from Calcium Imaging Data**"; Vahid Rahmati, Knut Kirmse, Knut Holthoff, Stefan J. Kiebel (Feb, 2018), Journal of Neurophysiology. Full Text: https://doi.org/10.1152/jn.00934.2017.

In the following, we first explain how you can simply setup the UFARSA. We then describe two examples, as well as how you can easily apply UFARSA to your recorded data. We then suggest a simple approach for setting the central parameter of UFARSA. Finally, we provide a number of useful hints, a description for the available plotting and saving options, and a description for the main function of UFARSA. For questions, comments, suggestions, or bug reports, please contact Vahid Rahmati (vahidrahmati92@gmail.com).

Note that in the following as well as the comments included in our codes, the term "frame" mainly refers to "time point", as UFARSA processes the fluorescence traces extracted from different regions of interest (ROIs; e.g. different cells in the recorded field of view), separately; i.e. it processes them as a batch job.

Setup

You can download UFARSA from the GitHub link provided in the paper (i.e. from https://github.com/VahidRahmati/UFARSA), where it will be downloaded as a zip file: "UFARSA-master.zip". To use UFARSA, first its directory should be added to the search path of MATLAB. This can be simply done as follows. First unzip the downloaded UFARSA-master.zip file, by extracting (or copying) its files into a directory in your system (preferably into a new folder in any Drive except the main Drive). Then in this directory, open the "setup_UFARSA.m" file in MATLAB and run it by clicking the green Triangle button on the Editor Toolbar of Matlab. That's it.

Examples

Two example demo scripts have been provided.

1) demo_UFARSA_simData.m: this script applies UFARSA to a synthetic fluorescence trace encoding a Poisson spike train with user-determined characteristics, including the underlying mean firing rate and recording sampling frequency.

2) demo_UFARSA_realData.m: this script applies UFARSA to a sample data file ("sample_data.mat") of fluorescence traces assumed to be extracted from eleven regions of interest (ROIs) at a sampling

frequency of 30 Hz; you can simply adapt this script to apply UFARSA to your recorded data (please see the "Applying UFARSA to "new" recorded data" section below).

To execute each of these scripts you need to first open its m-file (e.g. demo_UFARSA_simData.m) in Matlab Editor. A quick way to open these scripts is to type e.g. " *edit* demo_UFARSA_simData " in the Matlab command window prompt and pressing Enter. After opening the script, then run it: by either clicking on the green Triangle button on the Editor Toolbar of Matlab, or simply typing its name (without the '.m' extension; e.g. demo_UFARSA_simData) at the Matlab command window prompt and then pressing Enter. Before executing these examples, please first read the descriptions at the top of their scripts. All lines of codes have been fully commented.

After running each of the demo scripts, the main outputs of the UFARSA are stored in a structure variable called *output_UFARSA* (see also *ROIs_all* variable in the "Main function" section, and the "Useful hints" section below). These include the reconstructed event times (*output_UFARSA.eTrain* or *output_UFARSA.eTrain_dem*) and the "relative" spike-count per fluorescence transient (or event; *output_UFARSA.cTrain* or *output_UFARSA.cTrain_dem*), as well as the estimated "scaled" firing rate vector (see the Hint (vi) in the "Useful hints" section below) based on convolving those reconstructed spike-counts with a Gaussian kernel (*output_UFARSA.cFR* or *output_UFARSA.cFR_dem*). These outputs can be generated in both cases of using or skipping the demerging step of UFARSA. The results in the paper were generated by using the demerging step; i.e. we set "*opt.demerging = 1*" and reported the outputs with "_dem" suffix in their names. In addition, different versions of the given fluorescence trace are available as output. These include the normalized trace (*output_UFARSA.fluors.normalized*), the pre-processed before-smoothing trace (e.g. after applying any fast or slowly varying drifts correction algorithms, available in UFARSA; *output_UFARSA.fluors.beforeSmoothing*), and the pre-processed after-smoothing trace which is obtained after applying the smoothing step to the pre-processed before-smoothing trace (*output_UFARSA.fluors.afterSmoothing*). For a detailed description and a list of outputs, please see the "Main function" section below. Besides, for the available options for plotting and saving the results, please see the "Plotting and saving options" section below.

Applying UFARSA to "new" recorded data

To apply UFARSA to your recorded fluorescence traces you can simply adapt the "demo_UFARSA_realData.m" file, by just changing the "*opt.FluorFile_name*" and "*opt.FluorFile_dir*" parameters to the filename and the directory of your fluorescence data, respectively. For setting the directory, please don't forget the "\" at the end, e.g. 'D:\data\'.

Currently, UFARSA accepts two data structures:

i) ".txt": a TXT file exported from "ImageJ" program, enclosing the fluorescence traces (i.e. time-courses) extracted from given regions of interest (ROIs). The structure is as follows (please also see the "image_txt.png" file):

Row 1, column 1: []
Row 1, column 2 … Row 1, column *last*: ROI number
Row 2, column 1 … Row *last*, column 1: Sample number (i.e. frame number)

Row 2, column 2 … Row *last*, column *last*: raw fluorescence values

ii) ".mat": a MAT file with M rows and N columns (i.e. an MxN matrix), where M is the number of recorded frames, and N is the number of fluorescence traces (i.e. the number of ROIs). Each column represents the raw fluorescence values of each trace (or ROI). Each row corresponds to a frame, where Row 1 relates to frame number 1, and Row 2 to frame number 2, and etc. As an example please see "sample_data.mat" file.

To increase the reconstruction accuracy, we strongly recommend that you set the central parameter of UFARSA (i.e. the leading threshold) manually, by using the simple approach described in the "Central parameter" section. Moreover, if you would like, you can also make UFARSA more sensitive in reconstructing within-burst (i.e. interior) events, by decreasing the "*opt.scale_min_interior_thr = 0.75*" to e.g. 0.5 (see the "parameters of reconstruction" block in the "internal_parameters.m" file; a quick way to open the "internal_parameters.m" script is to typing " *edit* internal_parameters " in the Matlab command window prompt and then pressing Enter). However, note that this decrease may yield an increase in the false detection error. Moreover, increasing "*opt.scale_burstAmp= 0.2*" to e.g. 0.4 (see "internal_parameters.m" script) can decrease the sensitivity in reconstructing interior events; i.e. UFARSA may reconstruct less interior events.

If necessary, you can also use the pre-processing steps of UFARSA to remove the <u>fast</u> and <u>slowly</u> varying drifts (see the Hint (v) in the "Useful hints" section below). However, note that UFARSA is in general not very sensitive to slowly varying drifts.

Central parameter

UFARSA has only one central parameter called "leading threshold", which is used to detect the fluorescence transients evoked by leading events: A leading event refers mainly to an isolated spike, the first spike of a burst, or the spike underlying the first fluorescence transient of several superimposed transients. To increase the reconstruction accuracy it is strongly recommended that user sets the leading threshold properly, by using the following simple approach (described by an example):

i) For all fluorescence traces of the same recording, select a couple of fluorescence traces (e.g. of ROIs 1 and 2; out of 100 ROIs) and apply UFARSA to each of them. Note that to do this for example for ROI 2, you need to set "*opt.which_ROIs= 2*" in the "demo_UFARSA_realData.m". After applying UFARSA to each trace (separately), you need to write down the UFARSA-estimated std of noise in that trace, by typing the "*output_UFARSA.std_noise*" at the Matlab command window prompt. For example, suppose you have written *down "std_noise1 = X"* and "*std_noise2 = Y"* for traces 1 and 2, respectively.

ii) Following this, you need to inspect the smoothed version (an output of UFARSA: *output_UFARSA.fluors.afterSmoothing*) of each of these traces to roughly estimate (but not overestimate) the expected <u>minimum</u>, noise-free amplitude of the leading transients. To do this, you can simply inspect the "smth. Fluor" trace in the plotted figures (you may find using the "Data Cursor" button on the ribbon of the plotted figure useful). Note that for plotting this smoothed trace, the following options should have been already set as "*opt.plt = 1*" and "*opt.plt_Ft_afterSmth = 1*" in the "internal_parameters.m" file; see the "Plotting and saving options" section below. Write down your estimations for the minimum leading amplitude in each trace, e.g. "*Amp_min1 = C*" and "*Amp_min2 = D*" for traces 1 and 2, respectively.

iii) Finally, estimate the "leading-threshold scaling constant" for each trace as "*Coeff_leading1 = (C/X)* " and " *Coeff_leading2 = (D/Y)* ". Then set the "*opt.scale_NoiseSTD*" parameter in the "demo_UFARSA_realData.m" file equal to the smallest *Coeff_leading* value between these two. That's it, and you can use this value for the rest of fluorescence traces of the same recording, as well (e.g. in this example, for all 100 ROIs). Following this setting, we also recommend to set the "opt.min_leading_amp = 0 " in the "internal_parameters.m" file, in order to remove the internally determined lower-bound used for the leading-threshold.

iv) Now, you can set " *opt.which_ROIs = []* " in the demo_UFARSA_realData.m and get the results for all ROIs, without the need for changing any setting.

Hint (important): To avoid overestimating the Coeff_leading, you can consider the following two points when performing the abovementioned approach: 1) preferably select the traces with apparently more variable and smaller leading transient amplitudes (to plot all fluorescence traces, set " *opt.which_ROIs=[]* " in the demo_UFARSA_realData.m file), and 2) after estimating the Amp_min, multiply it by e.g. 0.8.

For the rest of reconstruction parameters of UFARSA, please see the "parameters of reconstruction" block in the "internal_parameters.m" file (see also the "Main function" section below). A quick way to open the "internal_parameters.m" script is to type " *edit* internal_parameters " in the Matlab command window prompt and then pressing Enter.

Useful hints

In this section, we would like to draw your attention to a number of useful hints:

i) To save the figures and reconstruction results, a new folder called "UFARSA_results", as well as two sub-folders called "Figures" and "Reconstructions" will be created in the folder of processed data. If these folders already exist, then they won't be created and the already-existed (i.e. saved) files will be replaced by the new ones. For more details, see the "Plotting and saving options" section below.

ii) If there are more than one fluorescence traces (or ROIs) need to be processed, then the variables "*output_UFARSA*" and "*opt_out*", which are outputs of the "run_UFARSA" function, will correspond to the last trace (or ROI). If you would like to store these output for all traces, then please assign the " *ROI_all* " variable as an output to the "run_UFARSA" function; please the "Main function" section below, for more details.

iii) Note that allowing variable "ROIs_all" to be an output of "run_UFARSA" function may significantly reduce the speed of the computer, especially in the case of long-term data. This is because all different versions of all fluorescence traces (i.e. of all ROIs) as well as all reconstruction variables will be stored in the working memory. Instead, we suggest using our implemented saving options to save the outputs of interest (see the "Plotting and saving options" section below).

iv) The reconstruction results based on *with* and *without* using the demerging step can be generated simultaneously. However, in the current version of the codes, for each reconstructed variable, they cannot be shown in the same figure. That is, reconstructed event train and reconstructed demerged event train cannot be shown in the same figure. The same applies to the plots of reconstructed spike-counts and estimated scaled firing rate vectors. If the user enables the plotting options for both *with* and *without* demerging, then only the demerged results will be plotted. Note that for plotting or producing the demerged results, you have to enable the demerging step, by setting "*opt.demerging = 1*" in the main scripts (e.g. in the demo_UFARSA_realData.m file).

v) If necessary, you can use the pre-processing steps of UFARSA to remove the fast and slowly varying drifts, thereby increasing the reconstruction accuracy. In general, UFARSA is not very sensitive to slowly varying drifts. The fast drifts include the large positive impulses as well as large short-lasting negative deflections in the fluorescence trace. Note that to obtain more accurate deflection-removal results, the fluorescence trace should be free of slowly varying drifts or had been corrected for such drifts. To remove slowly varying drifts you can use the BEADS method implemented in UFARSA, where you may need to adjust the normalized cut of frequency of this method properly ("*opt.fc_beads*"). For a complete list of these drift/deflection correction options, please see the "parameters of pre-processing (detrending)" and "parameters of pre-processing (deflection removal)" blocks in the "internal_parameters.m" file (see also the "Main function"section below).

vi) UFARSA has been developed for reconstructing spiking activity trains (e.g. spike train or spike-count train), and the code generates these outputs for the user. However, <u>in addition and just for visualization</u>, the code also generates the "scaled" firing rate vector (i.e. smoothed spike-count train); based on convolving the reconstructed spike-count train with a Gaussian kernel. However, please note that these generated rates are "proportional" to firing rates; i.e. they are not in units of [Hz], because a division of the generated rates by the area-under-kernel has not been performed (I will implement it in next update). Nevertheless, the scaled firing rate vector not only provides a visualization of the rate changes over time, but can also be used for some further analysis; including the calculation of the <u>Pearson correlation index</u> (this index does not depend on the scale).

Plotting and saving options

For recorded data, a bunch of options for plotting and saving the UFARSA's outputs (reconstructions and pre-processed fluorescence traces) have been implemented; please see the "internal_parameters.m" file (a quick way to open the "internal_parameters.m" script is to type " *edit* internal_parameters " in the Matlab command window prompt and then pressing Enter). By enabling the saving options, first a new folder called "UFARSA_results" will be created in the directory where the processed fluorescence data exist. Then, the sub-folders “Figures” and “Reconstructions” will be created for saving figures or output files, respectively. Following this, another sub-folder will be created to save each type of output files. If these folders already exist, then they won't be created and the already- existed (saved) files will be replaced by the new ones.

For saving the figures, different formats are available, where you need to set the “*opt.save_plt_format*” parameter to your desired format. These include “.png”, “.tif”, “.jpg”, and etc. For saving the outputs (reconstructions and pre-processed fluorescence traces), currently there are two formats available: “.txt” and “.mat”. You can select your desired format by using the “*opt.save_output_format*” parameter. The structure of the saved file is same for these two formats, and has been described in this example:

Example: Let’s assume that your recorded fluorescence data are in this directory “D:\data\”. Now suppose that you have enabled the “saving” options of UFSAR, and then have applied UFARSA to these data. Then, “D:\data\Reconstructions\cTrains_dem” sub-folder contains a list of files named as “cTrain_dem_ROI(1).txt”, “cTrain_dem_ROI(1)_fire.txt”, “cTrain_dem_ROI(2).txt”, “cTrain_dem_ROI(2)_fire.txt”, and etc. That is, a separate file has been created for each ROI (note the number within the parentheses). The suffix “dem” denotes that when generating the reconstruction results saved in these files, the “demerging” step was used. The second column in “cTrain_dem_ROI(1).txt” shows the reconstructed demerged spike-count vector from the fluorescence trace extracted from ROI number 1. First column shows the number of recorded frames. Evidently, the reconstructed vector has the same number of rows as its fluorescence trace. “cTrain_dem_ROI(1)_fire.txt” includes only spiking data (note the suffix “fire”). It has two columns, where the first column includes the reconstructed demerged event times, and the second column shows the estimated spike-count per that event (i.e. per its detected fluorescence transient). Similar structures were used for saving the reconstructed event trains, the estimated “scaled” firing rate vectors, and the pre-processed fluorescence traces.

For a full list of plotting and saving options, please see the “internal_parameters.m” file.

Main function

The main function for running UFARSA is the “run_UFARSA.m” m-file. You can see this function at the end of “demo_UFARSA_realData.m” and “demo_UFARSA_simDATA” scripts. Importantly, note that when applying UFARSA to your recorded data, you should “not” assign the inputs “fluor” (simulated fluorescence trace) and “true_spikes” (simulated spike-count train) to this function (e.g. see “demo_UFARSA_realData.m”). These two inputs are just for the simulation demo script (see “demo_UFARSA_realData.m”). In other words, the general form of the run_UFARSA function when applying UFARSA to recorded data is:

[output_UFARSA, opt_out, ROIs_all] = run_UFARSA(opt)

while its complete form is

[output_UFARSA, opt_out, ROIs_all] = run_UFARSA(opt, fluor, true_spikes)

In the following, we provide a detailed description of the input and output variables as well as the parameters, plotting and saving options of “run_UFARSA” function (adapted from the function-description provided at the top of the run_UFARSA.m script). To set the parameters/options of UFARSA to what you desired, you can change their values in “demo_UFARSA_realData.m” (for main parameters) and internal_paramters.m (for more parameters and saving/plotting options) files. When changing the value of any parameter/option in the internal_paramters.m file, first SAVE the script (CTRL+S), and then run again the main script (e.g. “demo_UFARSA_realData.m”). Regarding the output variable “ *ROIs_all* ”, please see the Hints (ii) and (iii) in the “Useful hints” section above.

[output_UFARSA, opt_out, ROIs_all] = run_UFARSA(opt, fluor, true_spikes)

This function applies UFARSA to the given fluorescence trace(s). The trace(s) will be first pre-processed, and then spiking activity trains will be reconstructed, plotted and saved.

INPUT:

fluor: the raw "simulated" fluorescence trace.

true_spikes: the "simulated" train of spiking event times or spike-counts (the time-unit is [frame]).

#**Hint**#: for real data don't assign any of "fluor" and "true_spikes" variables as input to this function; i.e. use instead: ... = run_UFARSA(opt)

opt: an structure variable, where:

Parameters of "Slow drifts removal"

opt.remove_drifts → 1: remove slowly varying drifts, 0: skip the drift removal step.

opt.detrending_method: select the drift removal method, 1: BEADS, 2: Polynomial.

opt.fc_beads: in [cycles/sample], filter's normalized cut-off frequency in BEADS method (0<fc_beads<0.5).

opt.asymmetery_param: asymmetry ratio parameter in BEADS method.

opt.beads_nAppendFrames: number of appended frames to the start and end of fluorescence trace, used for compensating for the potential boarder effects after applying BEADS method.

opt.beads_fast → 1: use our fast implementation of BEADS method, 0: use the original code of BEADS method.

opt.filterOrder_param: filter's order in BEADS method (set it to 1 or 2).

opt.poly_degree: degree of polynomial function used by polynomial detrending method.

Parameters of "Deflections removal"

opt.remove_posDeflections → 1: apply large-impulse (deflection) removal step, 0:skip this step.

opt.posPrctileLevel: positive percentile level, used in the large-impulse removal step.

opt.scale_posDefl: a constant.

opt.remove_negDeflections → 1: remove large short-lasting negative deflections, 0:skip this step.

opt.negPrctileLevel: negative percentile level, used in the negative-deflections removal step.

opt.scaleRep_negDefl: a constant.

Parameters of "smoothing & noise-level estimation"

opt.test_smoothing → 1: check smoothing quality (recommended), 0: skip the check.

opt.test_smallS → 1: check smoothing quality when smoothing parameter is smaller than 1 (recommended for real data), 0: skip the check.

opt.smoothing_param: smoothing parameter; [ ]: automatic, [NUMBER]: the given NUMBER will be used for smoothing (non-automatic).

opt.Svalue_when_fail: the value of smoothing parameter to be used when smoothing was not proper.

opt.samples_randomly → 1: select the frames for computing the STD randomly, 0: select them in a non-random manner.

opt.nSamplesForSTD → [ ]: consider the whole trace for estimating noise STD.

→ [NUMBER]: consider the first, or randomly selected, NUMBER of frames to estimate noise STD.

→ [NUMBER1 NUMBER2]: consider frames from frame NUMBER1 up to frame NUMBER2, to estimate noise STD.

opt.warn_pms → 1: show the potential warning messages in command window, 0: don't show them.

opt.noiseLevel_method: noise-STD estimation method: 1 → Power-spectrum density (PSD), 0 → residual.

Parameters of "reconstruction"

opt.demerging → 1: apply the demerging step, 0: skip it.

opt.gen_FR_count → 1: generate the estimated firing rate vector based on the reconstructed spike-count train, 0: skip it.

opt.gen_FR_count_dem → 1: generate the estimated firing rate vector based on the reconstructed demerged spike-count train, 0: skip it.

opt.min_leading_amp → [ ]: leading threshold will be estimated, and be lower-bounded with the minimum leading threshold (default).

→ 0: minimum leading threshold will not be used to lower-bound the leading threshold (recommended when user estimated leading threshold from data).

→ [scalar]: an estimate of the minimum amplitude of non-within-burst transients (e.g. isolated or leading); see user_guide.pdf file, for more details.

opt.scale_NoiseSTD: Leading-threshold scaling constant (by default 2.25).

% We strongly recommended to estimate this parameter easily from a couple of you fluorescence traces (see user_guide.pdf file). Following this estimation, we recommended to set the " opt.min_leading_amp = 0 " in the "internal_parameters.m" file, in order to remove the internally determined lower-bound used for this threshold.

opt.denominator_prct: Min-leading-threshold scaling constant.

% To determine the lower bound for leading threshold, the 98th percentile of the smoothed fluorescence trace is divided by this number.

opt.scale_burstAmp: Min-interior-amplitude scaling constant.

opt.plateau_interior: Interior-plateau scaling constant.

opt.scale_min_amp: Min-leading-amplitude scaling constant (see the user_guide.pdf file), this is used when "opt.min_leading_amp = [scalar] ".

opt.scale_min_interior_thr: Min-interior-threshold scaling constant (a number within the range [0 1]): decreasing this (e.g. to 0.5) can lead to a better reconstruction of within-burst spikes.

opt.max_nBurstSpikes → [Number]: Maximum spike-count constant; maximum spike-count which can be reconstructed from a detected leading transient.

→ [ ]: no restriction (i.e. maximum bound) will be applied to the raw estimated spike-count (not recommended).

opt.round_border: Border used for the imbalanced rounding of the estimated spike-counts; a scaler between 0.5-1 (default: 0.75).

opt.onset_shift: Onset-shifting parameter (in [frame]); the corrected reconstructed event times will be shifted by "onset_shift" number of frames.

opt.sigma_gauss: in [frame], the STD of Gaussian kernel used to be convolved with the spiking activity train, in order to generate estimated firing rate vector.

Parameters of "figures": plotting and saving

opt.plt → 1: plot the results, 0: skip plotting results.

opt.save_plt → 1: save the plotted figure, 0 --> don't save it.

opt.save_plt_format: the format of saved figures; e.g. '.png' or '.jpg' or '.tif' or etc.

opt.plt_eTrainOnFt → 1: plot the user's selected type of the reconstructed event train on fluorescence trace(s), 0: don't show it.

opt.plt_eTrain → 1: plot the reconstructed event train (E(t)), 0: don't show it.

opt.plt_cTrain → 1: plot the reconstructed spike-count train (C(t)), 0: don't show it.

opt.plt_cFR → 1: plot the estimated firing rate vector based on C(t), 0: don't show it.

opt.plt_eTrain_dem → 1: plot the reconstructed demerged event train (E_dem(t)), 0: don't show it.

opt.plt_cTrain_dem → 1: plot the reconstructed demerged spike-count train (C_dem(t)), 0: don't show it.

opt.plt_cFR_dem → 1: plot the estimated firing rate vector based on C_dem(t), 0: don't show it.

opt.plt_eTrain_sim → 1: plot the simulated event train, 0: don't show it.

opt.plt_cTrain_sim → 1: plot the simulated spike-count train, 0: don't show it.

opt.plt_Ft_orig_norm → 1: plot the normalized version of the raw fluorescence trace, 0: don't show it.

opt.plt_Ft_beforeSmth → 1: plot the fluorescence trace before smoothing, 0: don't show it.

opt.plt_Ft_afterSmth → 1: plot the pre-processed fluorescence trace after smoothing, 0: don't show it.

opt.display_NoiseSTD → 1: display the estimated noise_std for each processed trace in the command window, 0: skip it.

#**Hint**#: To save the figures, first a new folder called "UFARSA_results", and then new sub-folder called "Figures" will be created in the folder of data. If these folders already existed, then they won't be created and the already-existed saved figures will be replaced by the new ones.

Parameters of "saving UFARSA's results": reconstructions and fluorescence traces

opt.save_outputs → 1: enable saving the reconstruction results and different versions of the fluorescence trace, 0: skip it.

opt.save_output_format → '.txt' or '.mat'; the files will be saved in this given format.

opt.save_eTrains → 1: save the reconstructed event train (E(t)), 0: don't save it.

opt.save_cTrains → 1: save the reconstructed spike-count train (C(t)), 0: don't save it.

opt.save_cFR → 1: save the smoothed firing rate trace based on C(t), 0: don't save it.

opt.save_eTrains_dem → 1: save the reconstructed demerged event train (E_dem(t)), 0: don't save it.

opt.save_cTrains_dem → 1: save the reconstructed demerged spike-count train (C_dem(t)), 0: don't save it.

opt.save_cFR_dem → 1: save the estimated firing rate vector based on C_dem(t), 0: don't save it.

opt.save_Ft_orig_norm → 1: save the normalized version of the raw fluorescence trace, 0: don't save it.

opt.save_Ft_beforeSmth → 1: save the fluorescence trace before smoothing, 0: don't save it.

opt.save_Ft_afterSmth → 1: save the pre-processed fluorescence trace after smoothing, 0: don't save it.

opt.save_ROIs_all → 1: save the cell variable containing the UFARSA's outputs for all fluorescence traces (ROIs), 0: don't save it.

#**Hint**#: To save these variables, first a new folder called "UFARSA_results", and then a new sub-folder called "Reconstructions" will be created in the folder of data. If these folders already exist, then they won't be created and the already existed saved files will be replaced by the new ones.

Parameters for "setting the data directory"

opt.FluorFile_name → name of the file containing your recorded fluorescence data; e.g. 'mydata.txt'.

opt.FluorFile_dir → directory where your recorded fluorescence data were saved; e.g. 'D:\data\'.

#**Hint**#: when applying UFARSA to your recorded data, give only "opt" as an input to the run_UFARSA function; i.e. as in the demo_UFARSA_realData.m code use: ...= run_UFARSA(opt).

OUTPUT:

output_UFARSA: an structure variable, where

Reconstructed spiking activities

output_UFARSA.eTrain: reconstructed "Event train" (in paper, denoted by E(t)).

output_UFARSA.eTrain_dem: reconstructed "Demerged event train" (in paper, denoted by E_dem(t)).

output_UFARSA.cTrain: reconstructed "Spike-count train" (in the paper, denoted by C(t)).

output_UFARSA.cTrain_dem: reconstructed "Demerged spike-count train" (in the paper, denoted by C_dem(t)).

output_UFARSA.cFR: estimated firing rate vector based on the reconstructed spike-count train, and the user-determined STD of Gaussian kernel.

output_UFARSA.cFR_dem: estimated firing rate vector based on the reconstructed demerged spike-count train, and the user-determined STD of Gaussian kernel.

Main thresholds and noise parameters

output_UFARSA.leading_thr: Leading threshold.

output_UFARSA.std_noise: estimated std of noise in the (normalized) fluorescence trace (it is computed in pre-processing step).

output_UFARSA.min_leading_thr_est: Minimum leading threshold.

output_UFARSA.interior_thr: Interior threshold.

Different versions of the given fluorescence trace

output_UFARSA.fluors.original: raw (non-normalized) fluorescence trace.

output_UFARSA.fluors.normalized: raw normalized fluorescence trace.

output_UFARSA.fluors.beforeSmoothing: normalized pre-processed before-smoothing fluorescence trace or raw normalized fluorescence trace (to be used for smoothing).

output_UFARSA.fluors.afterSmoothing: fluorescence trace after smoothing (to be used for reconstruction).

opt_out: an structure variable, including all parameters assigned to the "opt" structure variable by user, preprocessing_UFARSA function, and within this current function.

ROIs_all: a cell variable including the UFARSA's outputs (i.e. both output_UFARSA and opt_out) for each of the fluorescence traces; e.g. to access the outputs for ROI number 3, write ROIs_all{3} in the Matlab Command Window prompt. To remove this output (recommended) use "[output_UFARSA,opt_out] = run_UFARSA(...", instead of "[output_UFARSA,opt_out, ROIs_all] = run_UFARSA(...".

#**Hint**#: Note that allowing "ROIs_all" to be an output of UFARSA may significantly reduce the speed of the computer, especially in the case of long-term data. This is because all different versions of all fluorescence traces (i.e. of all ROIs) as well as all reconstruction variables will be stored in the working memory. Instead, we suggest using the abovementioned saving options to save the outputs of interest.

#**Hint**#: If there is more than one fluorescence trace need to be processed, then the output variables "output_UFARSA" and "opt_out" will correspond to the last trace.

Author: Vahid Rahmati (March, 2018)