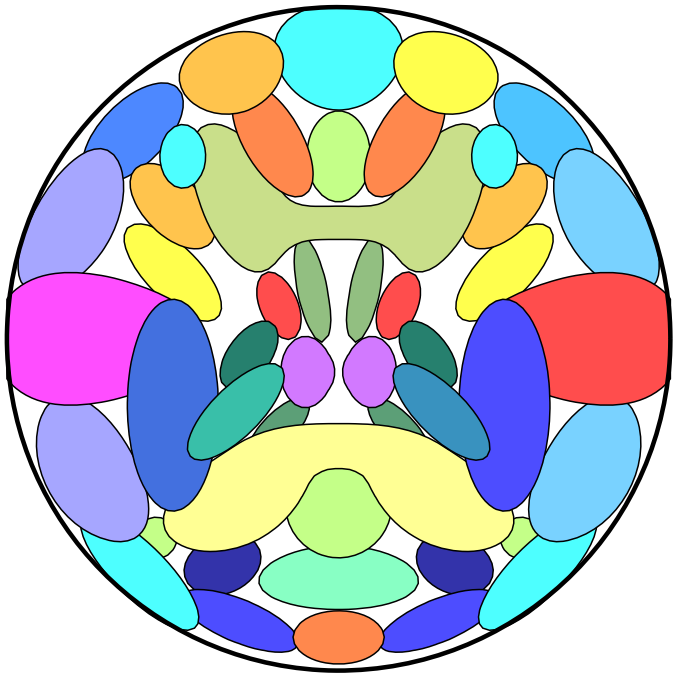
A simulation toolbox for fMRI data: SimTB

Elena A. Allen[1], Erik B. Erhardt[1], Yonghua Wei[2], Tom Eichele[3], and Vince D. Calhoun[1,2]

[1]*The Mind Research Network, Albuquerque, New Mexico, USA*
[2]*University of New Mexico, Albuquerque, New Mexico, USA*
[3]*University of Bergen, Bergen, Norway*

http://mialab.mrn.org
June 24, 2011



Contents

1 Introduction

We have developed a simulation toolbox, SimTB, running under the crossplatform MATLAB environment (The Mathworks, Inc.). The toolbox allows for flexible generation of fMRI datasets under a model of spatiotemporal separability and is designed to facilitate the testing of a variety of analytic methods. We have previously introduced the toolbox in Erhardt et al. (2011) and have used it to explore subject variability within the group ICA framework in Allen et al. (2011a).

Within SimTB users have full control over data generation including the creation and manipulation of spatial sources, implementation of block- and event-related experimental designs, inclusion of tissue-specific baselines, simulated head movement, and more. Beginning MATLAB users can use the SimTB graphical user interface (GUI) to design and execute simulations while experienced users can write batch scripts to automate and customize this process. The toolbox is freely available at http://mialab.mrn.org/software together with sample scripts and tutorials. Please send comments and bug reports to vcalhoun@mrn.org or eallen@mrn.org.

2 Theory

In SimTB, we adopt a data generation model that is consistent with the spatiotemporal separability assumptions of independent component analysis (ICA), that is, data can be expressed as the product of time courses (TCs) and spatial maps (SMs). This process is shown in Figure 1. Specifically, for each subject, $i = 1, \ldots, M$, we assume there are up to C components, each consisting of a SM having both a TC of activation and an amplitude. The no-noise (nn) data is a linear combination of amplitude-scaled and baseline-shifted TC and SM components,

$$\mathbf{Y}_i^{\text{nn}} = [\mathbf{R}_i \text{diag}(\mathbf{g}_i)\mathbf{S}_i + \mathbf{J}_T^V] \odot b_i \mathbf{J}_T^1 \mathbf{u}_i^\top, \tag{1}$$

where $\mathbf{Y}_i^{\text{nn}}$ is the time-by-voxel (T-by-V) no-noise data for subject i, $\mathbf{R}_i$ is a matrix of C column vectors of TCs, $\mathbf{S}_i$ is a matrix of C row vectors of SMs, $\mathbf{g}_i$ is a vector of C component amplitudes defined as a percent signal change of the baseline, b_i is a baseline intensity scalar, $\mathbf{u}_i^\top$ is a vector of voxel baseline intensity modifiers, $\mathbf{J}_T^V$ is a T-by-V matrix of ones, and $\odot$ denotes the Hadamard (element-wise) matrix product. These and other variables are further defined in the following sections, as well as the method for producing the final subject data.

2.1 Spatial maps

A template of the 30 default SMs is shown in Figure 2A on a square image of $V = \sqrt{V} \times \sqrt{V}$ voxels, where side length $\sqrt{V}$ is specified by the user. Users can specify SMs using any 2-D function defined on $x, y \in [-1, 1]$. Default SMs are modeled after components commonly seen in axial slices of real fMRI data and most are created by combinations of simple Gaussian distributions. For example, a component with a single blob of activation can be defined as

$$\mathbf{S}_{ic} = \exp(-[w_x\{(x - x_0)\cos(\theta) - (y - y_0)\sin(\theta)\}]^2) \times \exp(-[w_y\{(x - x_0)\sin(\theta) + (y - y_0)\cos(\theta)\}]^2)$$

where location (x_0, y_0), orientation θ, and width (w_x, w_y) parameterize the activation. Users can vary the location (x_0, y_0) and orientation θ of activation blobs across subjects. The spatial extent of

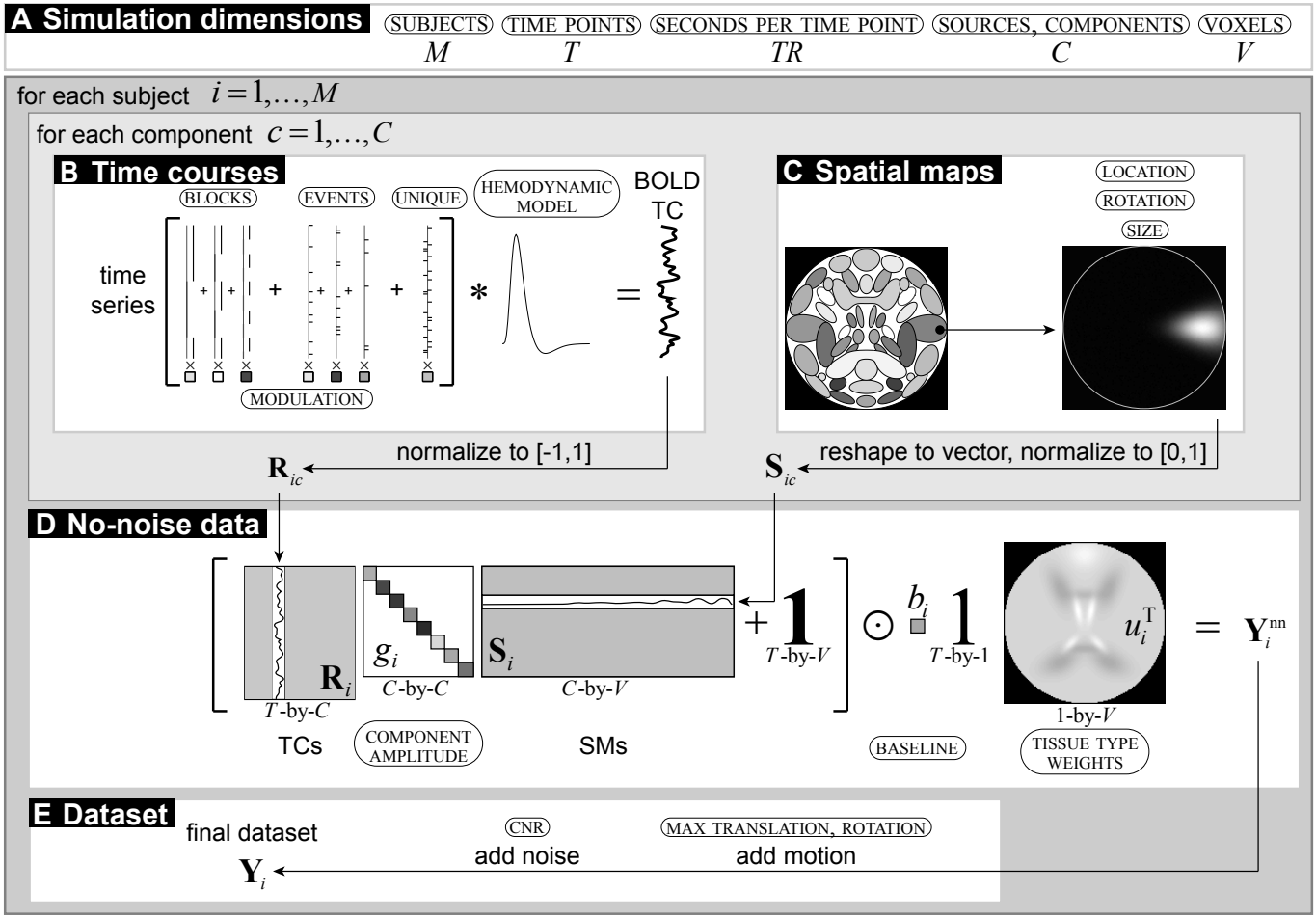


Figure 1: **Flowchart of data generation. (A)** Simulation dimension is determined by the number of subjects, time points (and seconds per time point), and voxels (representing a number of selected sources). **(B)** Time courses are the sum of task block, task event, and unique event time series modeled into a BOLD TC and normalized. **(C)** Spatial maps are selected, translated, rotated, resized, and normalized. **(D)** The no-noise data combines the TCs and SMs scaled by component amplitudes, and scaled to a tissue type weighted baseline. **(E)** The final dataset includes motion and noise.

the whole component can also be varied with the "spread" parameter, ρ. SMs are normalized to have a maximum intensity of 1 and are transformed as $\mathbf{S}'_{ic} = \mathbf{S}_{ic}^{1/\rho}$, where ρ describes the expansion ($\rho > 1$) or contraction ($\rho < 1$) of the component and $\mathbf{S}'_{ic}$ is the modified SM for subject i. Finally, a little Gaussian noise distributed as $\mathcal{N}(0, 2.5 \times 10^{-5})$ is added so that each subject SM is unique.

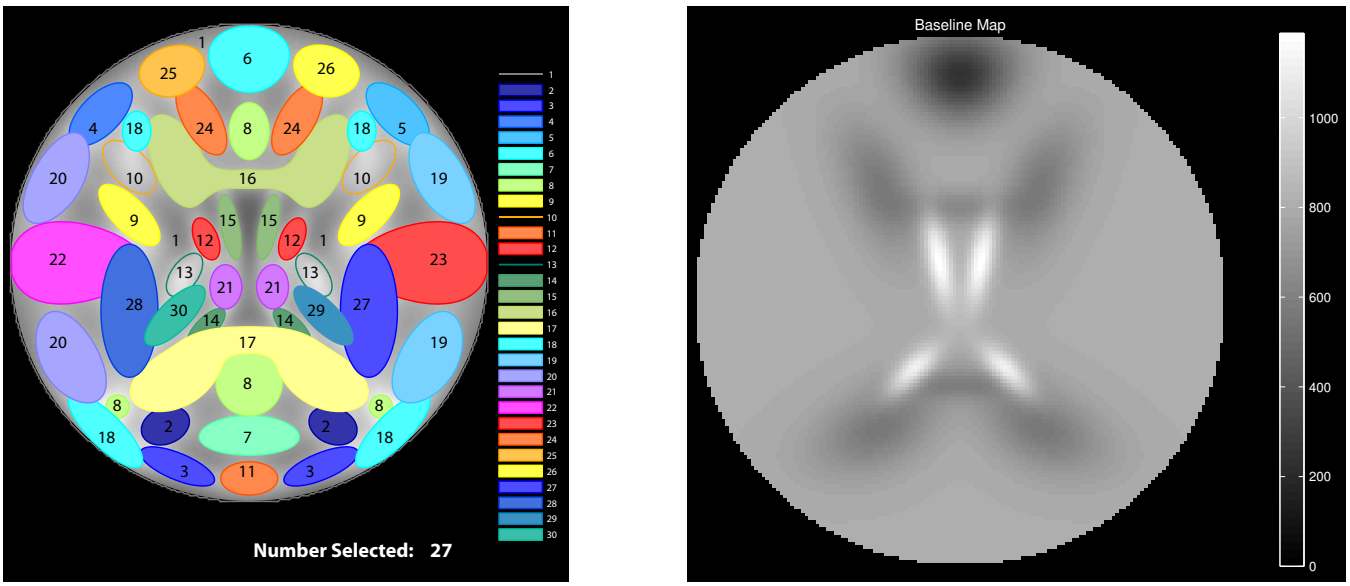


Figure 2: Configuration of default sources (left) and default tissue baseline (right). Sources are labeled and 27 of 30 have been selected for a simulation. Spatial maps are designed to represent components observed in axial slices of real fMRI data.

2.2 Time courses

Each component TC is T time points in length, where the user specifies the repetition time (TR) in seconds per sample. TCs are constructed under the assumption that component activations result from underlying neural events as well as noise. Neural events can follow block or event-related experimental designs, or can represent unexplained, random deviations from baseline. We refer to an underlying event time series as TS to distinguish it from the subsequent TC that is created with a hemodynamic model.

Experimental paradigms are designed with task blocks and task events which can be assigned to several components and can be identical across subjects. Unique events refer to unexplained deviations that are unique to each component and subject. These three types of TS inputs are controlled independently. Each task block is described by a block length and an inter-stimulus interval. If multiple task blocks are defined, their order is pseudo-randomized so that each task block occurs an equal number of times. Task events and unique events are defined by a probability of occurrence at each TR. For a given component, the TS is created by adding together amplitude-scaled task blocks, task events and unique events. Amplitudes for task inputs can be negative or positive (indicating suppression or activation with the task), or can be zero (indicating that component activation does not follow the task).

Generating the fMRI BOLD-like TCs from the event TS may be done in several ways, including linear convolution with a canonical hemodynamic response function (HRF) (difference of two gamma functions) (Friston et al., 1995) and the Windkessel ballon model (Friston et al., 2000). Users may vary hemodynamic parameters between components and subjects (see Section 6.3), and define their own TC source models (see Section 6.5). After creation of the TCs, each component TC is scaled to have a peak-to-peak range of one. As with the SMs, Gaussian noise distributed as $\mathcal{N}(0, 2.5 \times 10^{-5})$ is added to ensure non-zero TCs.

2.3 Baseline intensity

A baseline intensity, b_i, is specified for each subject. An optional tissue-type modifier $\mathbf{u}_i^\top$ scales the baseline for each voxel. Tissue types with corresponding intensity levels, ω are assigned to each component. Tissue types are assigned to each component and voxel intensity levels are then determined by

$$\mathbf{u}_i^\top = \mathbf{J}_T^V + \sum_{c=1}^{C} (\omega_c - 1)|\mathbf{S}_{ic}|.$$

For example, Figure 2B displays the baseline intensity map where four tissue types are defined: sinus signal dropout ($\omega_6 = 0.3$), cerebrospinal fluid (CSF) ($\omega_{14,15} = 1.5$), white matter ($\omega_{16,17} = 0.7$) and gray matter ($\omega_c = 1$ for all other sources). For the example subject, $b_i = 800$, thus the intensity map ranges from $0.3 \times 800 = 240$ in areas with signal dropout to $1.5 \times 800 = 1200$ in CSF.

2.4 Dataset, putting it all together

SMs and TCs are scaled according to the component amplitudes, $\mathbf{g}_i$, which are specified in terms of peak-to-peak percent signal change relative to the local baseline. Component features are linearly combined to form the no-noise dataset as in equation 1.

Motion may then be added by specifying the maximum x- and y-translation and rotation for each subject. Rotation and translation timeseries are generated via an autoregressive process of model order 1 [AR(1)] with parameter 0.95, which is a relatively smooth, but effectively bounded, random walk. This model assumes that the head moves randomly between time points but tends toward a central position more than extremes. This process is effectively bounded at 10 standard deviations of the normal deviate. Let the quantity of translational or rotational motion at time t, m_t, start at $m_1 = 0$ and randomly walk with $m_{t+1} = 0.95m_t + z_t(\text{MaxMot}/10)$, $t = 2, \ldots, T$, where MaxMot is the maximum translation or rotation and $z_t \sim \text{Normal}(0, 1)$. For each subject, x- and y-translation and rotation are simulated independently. The dataset is transformed accordingly by linear interpolation at each time point. Datasets are padded with an enlarged bounding box to accommodate translation.

To construct the noisy subject data matrix, $\mathbf{Y}_i$, we add Rician noise to the data relative to a specified contrast-to-noise ratio (CNR) (Gudbjartsson and Patz, 1995). Here, we define CNR as $\hat{\sigma}_\text{s}/\hat{\sigma}_\text{n}$, where $\hat{\sigma}_\text{s}$ is the temporal standard deviation of the true signal and $\hat{\sigma}_\text{n}$ is the temporal standard deviation of the noise. We calculate $\hat{\sigma}_\text{s}$ as the 30% trimmed mean of the standard deviations of the no-noise voxel timeseries (i.e., columns of $\mathbf{Y}_i^\text{nn}$). Unless otherwise stated, we use CNR $= 1$, which is within the range of typical CNR values for BOLD contrast at conventional voxel sizes and field

strengths. Each element of $\mathbf{Y}_i$ is then $Y_{itv} = \sqrt{(Y^{\text{nn}}_{itv} + \nu_{1tv})^2 + \nu^2_{2tv}}$, $t = 1, \ldots, T$ and $v = 1, \ldots, V$, where ν_{1tv} and ν_{2tv} are distributed as $\mathcal{N}(0, \hat{\sigma}^2_{\text{n}})$.

The final dataset, $\mathbf{Y}_i$, is saved to disk in .mat or .nii format along with the simulation parameters, the true SMs and TCs, and the motion parameters. When saved in a standard imaging format (.nii), simulated datasets can be used with pre-processing or analysis streams developed in any of the standard functional imaging toolboxes (e.g., SPM[1] AFNI[2], FSL[3]).

[1]Statistical Parameteric Mapping, http://www.fil.ion.ucl.ac.uk/spm/
[2]Analysis of Functional NeuroImages, http://afni.nimh.nih.gov/afni
[3]FMRIB Software Library, http://www.fmrib.ox.ac.uk/fsl/

3 Getting Started

3.1 Installation

The SimTB source code and examples can be downloaded from http://mialab.mrn.org/software. To begin using the toolbox, place the folder simtb_v18 in a desired installation directory (SIMTB_INSTALL_PATH). Add the directory and sub-directories to your MATLAB path using pathtool, or from the command line type

```
>> addpath(genpath('SIMTB_INSTALL_PATH\simtb_v18'))
```

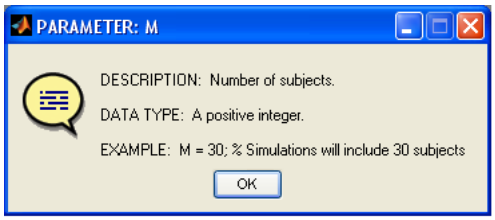
You are now ready to start using SimTB. To launch the GUI, enter simtb at the command line:

```
>> simtb
```

Alternatively, use the toolbox in batch mode by defining parameters in a script (.m file). A full walk-through for both these methods is provided in Section 5. Below we provide a few hints to help you get started.

3.2 Help!

Help is always right around the corner. When using the graphical user interface (GUI), explanations for every parameter can be accessed by clicking on the help boxes: ? . For example, clicking on the ? box for parameter M pops up the window below.



The same information is provided using the function simtb_params:

```
>> simtb_params('M')
```

```
-----------------------------------------------------------------------------------------
   NAME: M
   DESC: Number of subjects.
   TYPE: A positive integer.
DEFAULT: 10
EXAMPLE: M = 30; % Simulations will include 30 subjects
-----------------------------------------------------------------------------------------
```

A list of functions and their descriptions can be found in Section 7. Help for all functions is available by typing help 'function_name'. HTML versions of the documentation can be accessed within MATLAB by typing simtb_doc 'function_name' and online at http://mialab.mrn.org/software.

3.3 Parameter Structure

Simulation parameters can be specified using the `SimTB` GUI or via parameter files. Both these methods are described at length in the walk-through. To create and view the default parameter structure, call `simtb_create_sP` at the command line:

```
>> sP = simtb_create_sP
```

```
sP =
                  M: 10
                 nC: 30
                 nV: 100
                 nT: 150
                 TR: 2
       SM_source_ID: [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30]
     TC_source_type: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
   TC_source_params: {10x30 cell}
         TC_event_n: 0
 TC_event_same_FLAG: 0
       TC_event_amp: []
      TC_event_prob: []
         TC_block_n: 0
 TC_block_same_FLAG: 0
    TC_block_length: 0
       TC_block_ISI: 0
       TC_block_amp: []
     TC_unique_FLAG: 1
     TC_unique_prob: [1x30 double]
      TC_unique_amp: [10x30 double]
         SM_present: [10x30 double]
     SM_translate_x: [10x30 double]
     SM_translate_y: [10x30 double]
           SM_theta: [10x30 double]
          SM_spread: [10x30 double]
         D_baseline: [800 800 800 800 800 800 800 800 800 800]
          D_TT_FLAG: 0
         D_TT_level: [0.3000 0.7000 1 1.5000]
              D_pSC: [10x30 double]
       D_noise_FLAG: 1
              D_CNR: [1 1 1 1 1 1 1 1 1 1]
      D_motion_FLAG: 0
  D_motion_TRANSmax: 0
    D_motion_ROTmax: 0
  D_motion_deviates: [10x3 double]
    verbose_display: 1
               seed: 208248
       saveNII_FLAG: 0
           out_path: 'X:\MyData\Simulations\'
             prefix: 'sim'
              pfile: ''
```

To create a custom parameter structure you can pre-define parameters in a `.m` file and pass the filename to `simtb_create_sP`. Any defined parameters will over-ride the default values. For example, the file `experiment_params_block` defines a few parameters that implement a simple block design. We can create the parameter structure with:

```
>> sP = simtb_create_sP('experiment_params_block');
```

```
Loading Parameters from 'experiment_params_block'
```

This parameter structure will be similar to the default but will have a block paradigm with two conditions, e.g.,

```
>> sP.TC_block_n
```

```
ans =

     2
```

Because this particular parameter file does not include definitions for the number of subjects or number of components, we can also define these at the command line. Other simulation variables will be identical, allowing for simple and flexible generation of parameter structures.

```
>> sP = simtb_create_sP('experiment_params_block', 30, 10) % 30 subjects, 10 comps
```

```
Loading Parameters from 'experiment_params_block'
sP =
                  M: 30
                 nC: 10
                   .
                   .
                   .
```

Another, more complex parameter file, `experiment_params_aod`, defines the number of subjects and components within the script, thus entering values at the command line will have no effect. In other words,

```
>> sP = simtb_create_sP('experiment_params_aod');
```

and

```
>> sP = simtb_create_sP('experiment_params_aod', 30, 10);
```

will both produce a parameter structure with 5 subjects and 27 components since these values are 'hard-coded' in the parameter file.

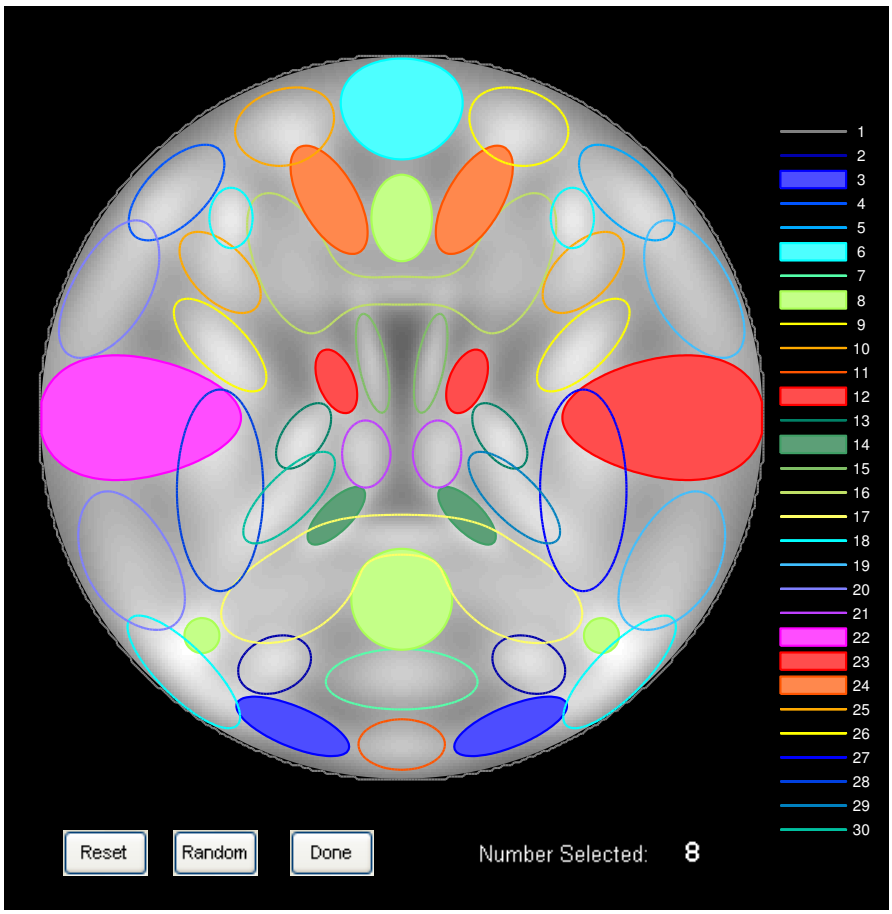
Users are encouraged to examine the example scripts, included in Section 8, and to use them as templates in their own simulations. The parameter files are also distributed with `SimTB` in `SIMTB_INSTALL_PATH\simtb_v18\examples`.

3.4 Select Sources

Viewing and selecting spatial sources may be done at any time by launching the stand-alone GUI `simtb_pickSM`. Note that `simtb_pickSM` is launched automatically in the main GUI ('Select Sources', Step 3).

```
>> simtb_pickSM
```

By clicking within the contours of each SM, you may select or de-select components.



The window is interactive until you hit 'Done', at which point the GUI becomes static and output regarding selected components will be displayed to the command window.

```
Number of selected components: 8
Component Source IDs:
    [3  6  8 12 14 22 23 24]
```

These Source IDs can then be copied and pasted directly into a parameter file, e.g.,

```
SM_source_ID = [3  6  8 12 14 22 23 24];
```

3.5 Run

To run a simulation, call `simtb_main` with the parameter structure (`sP`) as input:

```
>> simtb_main(sP)
```

Running the simulation will produce a number of figures (not shown) as well as display the following output to the screen.

```
---------------------------------------------------------------------
---------------------Initializing Simulation-----------------------
---------------------------------------------------------------------

Output directory: X:\MyData\Simulations\
     File prefix: sim
 Verbose display: ON

    Subject 1 of 10:
        Building Time Courses
        Building Spatial Maps
        Building Dataset
        Adding Noise
        Saving Simulated Components
        Saving Simulated Dataset
    Complete.
    Time to simulate subject 1: 6.6 s

    Subject 2 of 10:
        Building Time Courses
        Building Spatial Maps
        Building Dataset
        Adding Noise
        Saving Simulated Components
        Saving Simulated Dataset
    Complete.
    Time to simulate subject 2: 6.0 s
             .
             .
             .
    Subject 10 of 10:
        Building Time Courses
        Building Spatial Maps
        Building Dataset
        Adding Noise
        Saving Simulated Components
        Saving Simulated Dataset
    Complete.
    Time to simulate subject 10: 5.7 s

Saving Parameter Structure
---------------------------------------------------------------------
------- Simulation Complete. Total Time: 1.0 minutes--------------
---------------------------------------------------------------------
```

3.6 Output

Results from the simulation will be saved in the output directory which is designated by the parameter `sP.out_path`. All filenames are pre-pended with a designated prefix; for the default parameter structure this is `sP.prefix = 'sim'`. Output includes six file types with variables as described below.

1. Parameter file: `[prefix]_PARAMS.mat`
   Contains the parameter structure used in the simulation. Parameter structure is stored in the variable `sP`.

2. SIM file: `[prefix]_subject_[subject_number]_SIM.mat`
   Contains simulated TCs and SMs for an individual subject. A separate file is produced for each subject. Variable `TC` is the `[nT x nC]` matrix of component TCs . Variable `SM` is the `[nC x nV*nV]` matrix of component SMs .

3. DATA file: `[prefix]_subject_[subject_number]_DATA.mat` or
   `[prefix]_subject_[subject_number]_DATA.nii`
   Contains the simulated dataset for an individual. Incorporates the baseline as well as optional noise and motion into the `TC*SM` product. If using .mat format (default), variable `D` is the `[nT x nV*nV]` matrix of data. If .nii format is used, data as saved in `nT` image volumes with dimensions `[nV x nV x 1]`.

4. MOT file: `[prefix]_subject_[subject_number]_MOT.txt`
   Contains the head motion parameters for an individual subject. Note that MOT files are only produced if motion is simulated (i.e., `sP.D_motion_FLAG = 1`). Motion data is saved in tab-delimited text file with nT rows and 3 columns. Column 1 lists the x-position in voxels at each time point, Column 2 lists the y-position in voxels `y`, and Column 3 lists the rotation in degrees. Head position at the first time point is always `[0, 0, 0]`.

5. MASK file: `[prefix]_MASK.nii`
   Contains a binary image mask with 1's inside the head boundaries and 0's outside. Note that a MASK file is not produced as part of standard output, but is generated easily at the command line with `MASK = simtb_createmask(sP, 1)`. Also note that this MASK generation assumes stationary head boundaries and should not be used if motion is incorporated into the simulation. The .nii file includes a single image volume with dimensions `[nV x nV x 1]`.

6. Figure files: `[variable].fig` and `[variable].jpg`
   MATLAB figures in orginal .fig format and exported .jpg format. Figures display the simulation models, parameters, motion parameters (if used) and final TCs and SMs for each subject. Figure files are automatically generated only if `sP.verbose_display = 1`.

4 Simulation Parameters

Datasets are defined by a set of 40 user-specified parameters. A full listing of parameters, their definition and brief examples follow. Note that parameter descriptions can be obtained at the command line by calling `simtb_params`.

```
-------------------------------------------------------------------------------------
   NAME: M
   DESC: Number of subjects.
   TYPE: A positive integer.
DEFAULT: 10
EXAMPLE: M = 30; % Simulations will include 30 subjects
-------------------------------------------------------------------------------------
   NAME: nC
   DESC: Number of components.
   TYPE: A positive integer.
DEFAULT: Number of defined sources: 30
EXAMPLE: nC = 12; % Datasets will have 12 components
-------------------------------------------------------------------------------------
   NAME: nV
   DESC: Side length of 2-D square image; full image will have [nV x nV] voxels.
   TYPE: A positive integer.
DEFAULT: 100
EXAMPLE: nV = 100; % Datasets will have [100 x 100] voxels.
-------------------------------------------------------------------------------------
   NAME: nT
   DESC: Number of time points (TRs).
   TYPE: A positive integer.
DEFAULT: 150
EXAMPLE: nT = 150; % Datasets will have 150 time points.
-------------------------------------------------------------------------------------
   NAME: TR
   DESC: Repetition time, in seconds.
   TYPE: Positive real.
DEFAULT: 2
EXAMPLE: TR = 2; % Repetition time of 2 seconds.
-------------------------------------------------------------------------------------
   NAME: SM_source_ID
   DESC: [1 x nC] vector of IDs for spatial sources used to generate SMs. You can
         select defined sources using the GUI simtb_pickSM() or define your own in
         simtb_SMsource().
   TYPE: Positive integers.
DEFAULT: 1:nC
EXAMPLE: SM_source_ID = [2 4 5 10 13 15 19 20 21 22 23 27];
         % SMs will be generated from these 12 sources.
-------------------------------------------------------------------------------------
   NAME: TC_source_type
   DESC: [1 x nC] vector of model types used to generation of TCs. An example of a
         model is convolution of a time course with a haemodynamic response function
         (HRF). You can learn about the defined models with simtb_countTCmodels() or
         define your own model in simtb_TCsource().
   TYPE: Positive integers.
DEFAULT: ones(1,nC)
```

```
EXAMPLE: TC_source_type = ones(1,nC); TC_source_type(6) = 3;
         % TCs for most components will be generated by model 1;
         % TCs for component 6 will be generated with model 3.
-------------------------------------------------------------------------------------
   NAME: TC_source_params
   DESC: {M,nC} cell array of parameters for generating TCs with the selected models.
         An example is parameters defining a haemodynamic response function (HRF). Set
         TC_source_params = cell(M,nC) to use the default parameters. Use
         simtb_countTCmodels() to learn about the necessary parameters, which may be a
         different length for each model.
   TYPE: Real.
DEFAULT: cell(M,nC)
EXAMPLE: for sub = 1:M
            for c = 1:nC
                TC_source_params{sub,c} = [5, 13, 1, 1, 5, 0, 24];
            end
            TC_source_params{sub,6} = [1, 5, .8, 1, 3, 0, 20];
         end
         % TC generation models are the same for all subjects;
         % component 6 has a model distinct from all other components.
-------------------------------------------------------------------------------------
   NAME: TC_block_n
   DESC: Number of different block conditions in a block design.
         Set TC_block_n = 0 for no block design.
   TYPE: A positive integer.
DEFAULT: 0
EXAMPLE: TC_block_n = 2; % Experiment with 2 block conditions.
         % In an Visual Contrast paradigm, these might designate SAD and HAPPY faces.
-------------------------------------------------------------------------------------
   NAME: TC_block_same_FLAG
   DESC: FLAG to make block timing the same (1) or different (0) across subjects.
   TYPE: Binary (1 = yes, 0 = no).
DEFAULT: 0
EXAMPLE: TC_block_same_FLAG = 1; % Timing of blocks will be the same across subjects.
-------------------------------------------------------------------------------------
   NAME: TC_block_length
   DESC: Length of each block (in TRs).
   TYPE: A positive integer.
DEFAULT: 0
EXAMPLE: TC_block_length = 15; % Each block will be 15 TRs long.
-------------------------------------------------------------------------------------
   NAME: TC_block_ISI
   DESC: Interstimulus interval is the length of the OFF block, in TRs.
   TYPE: A positive integer.
DEFAULT: 0
EXAMPLE: TC_block_ISI = 10; % Blocks will be separated by 10 TRs.
-------------------------------------------------------------------------------------
   NAME: TC_block_amp
   DESC: [nC x TC_block_n] matrix of task-modulation amplitudes.
         Units are arbitrary but should be relative to experiment and/or unique event
         amplitudes.
   TYPE: Real.
DEFAULT: []
EXAMPLE: TC_block_amp = ones(nC,TC_block_n);
```

```
         TC_block_amp(:,2) = -0.5*TC_block_amp(:,2);
         TC_block_amp([1 5 8],:) = 0;
         % Most components have an amplitude of 1 for block type 1,
         % and an amplitude of -0.5 for block type 2.
         % Components 1, 5, and 8 have no block-related activity.
-------------------------------------------------------------------------------------
   NAME: TC_event_n
   DESC: Number of different types of events for an event-related design experiment.
         Set TC_event_n = 0 for no event-related design.
   TYPE: A positive integer.
DEFAULT: 0
EXAMPLE: TC_event_n = 3; % Experiment with 3 different types of events.
         % In an Oddball paradigm, these might be STANDARD, NOVEL and TARGET stimuli.
-------------------------------------------------------------------------------------
   NAME: TC_event_same_FLAG
   DESC: FLAG to make event structure timing the same (1) or different (0) across
         subjects.
   TYPE: Binary (1 = yes, 0 = no).
DEFAULT: 0
EXAMPLE: TC_event_same_FLAG = 1; % Timing events will be the same across subjects.
-------------------------------------------------------------------------------------
   NAME: TC_event_amp
   DESC: [nC x TC_event_n] matrix of task-modulation amplitudes.
         Units are arbitrary but should be relative to block and/or unique event
         amplitudes.
   TYPE: Real.
DEFAULT: []
EXAMPLE: TC_event_amp = 2*ones(nC,TC_event_n);
         TC_event_amp(:,1) = 1;
         TC_event_amp(2,:) = -TC_event_amp(2,:);
         TC_event_amp(5,:) = 0;
         % Most events have amplitude 2 and event type 1 has amplitude 1.
         % Component 2 activity decreases with events.
         % Component 5 has no event-related activity.
-------------------------------------------------------------------------------------
   NAME: TC_event_prob
   DESC: [1 x TC_event_n] vector of probabilities that an event occurs at each TR.
         The sum of TC_event_prob (i.e., the probability of any event) must be ≤ 1.
   TYPE: Probabilities ([0,1]).
DEFAULT: []
EXAMPLE: TC_event_prob = 0.2*ones(1,TC_event_n); TC_event_prob(2) = 0.05;
         % Most events will occur (on average) every 5 TRs.
         % Event type 2 will occur more rarely (on average once every 20 TRs).
-------------------------------------------------------------------------------------
   NAME: TC_unique_FLAG
   DESC: FLAG to include (1) or exclude (0) random events that are unique to each TC.
         If there is no experimental design (TC_event_n = 0 and TC_block_n = 0),
         TC_unique_FLAG should be 1 so that component TCs are not completely flat.
   TYPE: Binary (1 = yes, 0 = no).
DEFAULT: 1
EXAMPLE: TC_unique_FLAG = 1; % TCs will have random and unique events.
-------------------------------------------------------------------------------------
   NAME: TC_unique_prob
   DESC: [M x nC] vector of probabilities that an event occurs at each TR.
```

```
   TYPE: Probabilities ([0,1]).
DEFAULT: 0.5*ones(M,nC)
EXAMPLE: TC_unique_prob = 0.2*ones(M,nC); % Unique events occur on average every 5 TRs.
-------------------------------------------------------------------------------------
   NAME: TC_unique_amp
   DESC: [M x nC] matrix of amplitude of unique events. Units are arbitrary but should
         be relative to designed (block or event) amplitudes. If there is no
         experimental design (TC_event_n = 0 and TC_block_n = 0), TC_unique_amp is
         irrelevant.
   TYPE: Real.
DEFAULT: ones(M,nC)
EXAMPLE: TC_unique_amp = ones(M,nC);
         % Unique events for all subjects and components will have amplitude 1.
-------------------------------------------------------------------------------------
   NAME: SM_present
   DESC: [M x nC] matrix indicating whether component is present (1) or absent (0) from
         the subject's dataset. Note that if a component is absent, any deviations in
         in baseline intensity associated with that component will also be absent. To
         retain alterations in the baseline intensity map without component activation,
         set D_pSC(subject, component) = 0;.
   TYPE: Binary (1 = yes, 0 = no).
DEFAULT: ones(M,nC)
EXAMPLE: SM_present = ones(M, nC); SM_present([1,3,6], 2) = 0;
         % Most components are present for all subjects.
         % Component 2 is absent for subjects 1, 3 and 6.
-------------------------------------------------------------------------------------
   NAME: SM_translate_x
   DESC: [M x nC] matrix of translations in the x-direction for component SMs, in units
         of voxels.
   TYPE: Real.
DEFAULT: zeros(M,nC)
EXAMPLE: SM_translate_x = zeros(M, nC);
         SM_translate_x(1:5, 5) = -0.05*nV;
         SM_translate_x(6:10, 5) = 5;
         % Most components are not horizontally translated.
         % For subjects 1-5, component 5 is translated left by 5% of the image size.
         % For subjects 6-10, component 5 is translated right by 5 voxels.
-------------------------------------------------------------------------------------
   NAME: SM_translate_y
   DESC: [M x nC] matrix of translations in the y-direction for component SMs, in units
         of voxels.
   TYPE: Real.
DEFAULT: zeros(M,nC)
EXAMPLE: SM_translate_y = zeros(M, nC);
         SM_translate_y(1:5, 7) = -0.05*nV;
         SM_translate_y(6:10, 7) = 5;
         % Most components are not vertically translated.
         % For subjects 1-5, component 7 is translated down by 5% of the image size.
         % For subjects 6-10, component 7 is translated up by 5 voxels.
-------------------------------------------------------------------------------------
   NAME: SM_theta
   DESC: [M x nC] matrix of rotation angles for component SMs, in degrees. Positive
         degrees denote clockwise rotation.
   TYPE: Real.
```

```
DEFAULT: zeros(M,nC)
EXAMPLE: SM_theta = zeros(M, nC); SM_theta(1:5, 9) = -45; SM_theta(6:10, 9) = 45;
         % Most components are not rotated.
         % For subjects 1-5, component 9 is rotated counter-clockwise 45 degrees.
         % For subjects 6-10, component 9 is rotated clockwise 45 degrees.
-------------------------------------------------------------------------------------
   NAME: SM_spread
   DESC: [M x nC] matrix of spatial magnification factors. Values greater than 1
         increase the spatial spread of the SM, values less than 1 contract the SM.
   TYPE: Positive real.
DEFAULT: ones(M,nC)
EXAMPLE: SM_spread = ones(M, nC); SM_spread(4, :) = 0.9; SM_spread(5, :) = 1.1;
         % Most component SMs are not enlarged or contracted.
         % For subject 4, all components are slightly contracted.
         % For subject 5, all componetns are slightly enlarged.
-------------------------------------------------------------------------------------
   NAME: D_baseline
   DESC: [1 x M] vector of baseline signal intensities for subjects.
   TYPE: Positive real.
DEFAULT: 800*ones(1,M)
EXAMPLE: D_baseline = 1000+100*randn(1,M);
         % Baselines are normally distributed across subjects
         % with a mean of 1000 and standard deviation of 100.
-------------------------------------------------------------------------------------
   NAME: D_TT_FLAG
   DESC: FLAG to include (1) or exclude (0) distinct baselines (i.e., tissue types).
         You can designate the tissue types of different components in simtb_SMsource().
   TYPE: Binary (1 = yes, 0 = no).
DEFAULT: 0
EXAMPLE: D_baseline = 1; % Model will include regions with distinct baselines
-------------------------------------------------------------------------------------
   NAME: D_TT_level
   DESC: [1 x TTn] matrix of fractional intensities for each tissue type (TT). The
         primary TT should have a D_TT_level of 1 (which will correspond to the
         intensity value in D_baseline); D_TT_levels of the other TTs indicate a
         fraction (or proportion) of this value. TTn is the number of defined tissue
         types which you can determine from simtb_SMsource(), or using the helper
         function simtb_countTT().
   TYPE: Positive real.
DEFAULT: Default TT levels: [0.3, 0.7, 1.0, 1.5]
EXAMPLE: D_TT_level = [0.3, 0.7, 1, 1.5];
         % TT = 1 has a very low relative baseline            (e.g., signal dropout)
         % TT = 2 has a baseline less than the primary TT     (e.g., White Matter)
         % TT = 3 is the primary TT                           (e.g., Gray Matter)
         % TT = 4 has a baseline greater than the primary TT  (e.g., CSF)
-------------------------------------------------------------------------------------
   NAME: D_pSC
   DESC: [M x nC] matrix of percent signal changes (pSC) for component activations, in
         units of percentage points.
   TYPE: Percentages ([0,100]).
DEFAULT: ones(M,nC)
EXAMPLE: D_pSC = 2*ones(M, nC); D_pSC(:,8) = 6; D_pSC(:,6) = 0.1;
         % Most components have a 2% signal change.
         % Component 8 has a 6% signal change; component 6 has a 0.1% signal change.
```

```
-------------------------------------------------------------------------------------
   NAME: D_noise_FLAG
   DESC: FLAG to include (1) or exclude (0) the addition of Rician noise to the datasets.
   TYPE: Binary (1 = yes, 0 = no).
DEFAULT: 1
EXAMPLE: D_noise_FLAG = 1; % Rician noise will be added to the datasets.
-------------------------------------------------------------------------------------
   NAME: D_CNR
   DESC: [1 x M] vector of contrast-to-noise (CNR) ratios. CNR is defined as the
         peak-to-peak range of the component activation, divided by the peak-to-peak
         range of the added Rician noise.
   TYPE: Positive real.
DEFAULT: ones(1,M)
EXAMPLE: D_CNR = linspace(0.5, 2, M);
         % CNR ranges linearly from 0.5 for subject 1 (most noise)
         % to 2 for subject M (least noise).
-------------------------------------------------------------------------------------
   NAME: D_motion_FLAG
   DESC: FLAG to include (1) or exclude (0) simulated motion of datasets.
   TYPE: Binary (1 = yes, 0 = no).
DEFAULT: 0
EXAMPLE: D_motion_FLAG = 1; % Data will be rotated and/or translated over time.
-------------------------------------------------------------------------------------
   NAME: D_motion_TRANSmax
   DESC: Maximum possible image translation, as a proportion of the image length.
   TYPE: A proportion ([0,1]).
DEFAULT: 0
EXAMPLE: D_motion_TRANSmax = 0.05;
         % Data will be maximally translated by 5% of the image size.
-------------------------------------------------------------------------------------
   NAME: D_motion_ROTmax
   DESC: Maximum possible image rotation, in degrees.
   TYPE: Real.
DEFAULT: 0
EXAMPLE: D_motion_ROTmax = 5; % Data will be maximally rotated by 5 degrees.
-------------------------------------------------------------------------------------
   NAME: D_motion_deviates
   DESC: [M x 3] matrix of motion deviates, as proportions of the maximum motion.
         Column 1 refers to x-translation for each subject,
         Column 2 refers to y-translation for each subject,
         and Column 3 refers to rotation for each subject.
         See simtb_makeMotParams() for info on motion parameter generation.
   TYPE: A proportion ([0,1]).
DEFAULT: zeros(M,3)
EXAMPLE: D_motion_deviates(:,1) = linspace(0, 1, M)';
         D_motion_deviates(:,2) = linspace(0, 1, M)';
         D_motion_deviates(:,1) = 0.8*ones(1, M)';
         % Translation in x and y range linearly from 0 for subject 1
         % to maximal for subject M (as defined by D_motion_TRANSmax).
         % The degree of rotational motion is the same for all subjects.
-------------------------------------------------------------------------------------
   NAME: saveNII_FLAG
   DESC: FLAG to save datasets in nifti format (.nii) rather than .mat format.
         If saveNII_FLAG = 1, SPM functions must be on the matlab path.
```

```
         To download SPM: http://www.fil.ion.ucl.ac.uk/spm/.
   TYPE: Binary (1 = yes, 0 = no).
DEFAULT: 0
EXAMPLE: saveNII_FLAG = 1;
         % Datasets will be saved in nifti format, rather than .mat format.
-------------------------------------------------------------------------------------
   NAME: verbose_display
   DESC: FLAG to display simulation parameters and simulation output.
         Figures will be saved as .fig and .jpg files.
   TYPE: Binary (1 = yes, 0 = no).
DEFAULT: 1
EXAMPLE: verbose_display = 1; % Figures will be displayed throughout the simulation.
-------------------------------------------------------------------------------------
   NAME: seed
   DESC: Seed used to set the state of the random number generator.  Can be used to
         exactly reproduce simulations.
   TYPE: A positive integer.
DEFAULT: round(sum(100*clock))
EXAMPLE: seed = round(sum(100*clock));
         % Seed is randomized each time the parameter structure is created.
-------------------------------------------------------------------------------------
   NAME: out_path
   DESC: Full path to output directory.
   TYPE: A string.
DEFAULT: 'SIMTB_INSTALL_PATH\simulations'
EXAMPLE: out_path = 'X:\MyData\Simulations\'; % Path to output
-------------------------------------------------------------------------------------
   NAME: prefix
   DESC: String used as a prefix for all output files.
   TYPE: A string.
DEFAULT: 'sim'
EXAMPLE: prefix = 'sim8'; % All filenames will have 'sim8' as a prefix.
-------------------------------------------------------------------------------------
   NAME: pfile
   DESC: File used to created parameter structure (if any).
   TYPE: A string.
DEFAULT: Full filename of parameter file (if used).
EXAMPLE: % NOT a user-specified parameter.  Auto-generated in simtb_create_sP
-------------------------------------------------------------------------------------
```

5 Walk-through

5.1 Simulation story: auditory oddball fMRI paradigm

To illustrate use of the GUI and parameter file we utilize the example simulation from Erhardt et al. (2011). We simulate an auditory oddball task (AOD), which consists of detecting an infrequent sound within a series of regular and different sounds (Kiehl et al., 2001). This event-related paradigm task consists of a single run of three stimuli presented to each participant in random order. The standard stimulus is a baseline tone, the target stimulus is a distinct tone that subjects should press a button upon hearing, and the novel stimulus is a random digital noise. These each occur at each TR with probability 0.6, 0.075, and 0.075, respectively (8:1:1 ratio), thus no auditory stimulus occurs on a quarter of the TRs. We model distinct effects of standard, target, and novel tones on the BOLD signals of different sources. In addition, we adjust the baseline intensity and temporal properties of sources to match the statistical moments found in read data (see Figure 3). Details of the simulation follow.

We simulate $M = 5$ subjects, each with up to $C = 27$ components in a dataset with $V = 148 \times 148$ voxels and $T = 150$ time points collected at TR $= 2$ seconds. Selected sources are those filled in Figure 2. Some of the selected sources are not "of interest" and are present with probability 0.9, that is, some sources may be absent for each subject. To mimic between-subject spatial variability, the sources for each subject will be given a small amount of translation, rotation, and spread via normal deviates. Translation in the horizontal and vertical directions of each source have a standard deviation of 0.1 voxels, rotation has a standard deviation of 1 degree, and spread has a center of 1 and standard deviation of 0.03.

To define the TCs, we use task events and unique events. We define four task event types which occur in a random order for each subject. At each TR, in addition to the three task event types mentioned earlier (1=standards, 2=targets, and 3=novels), a spike event (4) occurs with probability 0.05. Components are separately modulated by each event type. Standard events are mapped to auditory sources (27 and 28) with amplitude 1 and other sources (24, 4, 5, and 18) with amplitude 0.7. Target events are mapped to auditory sources (27 and 28) with amplitude 1.2, motor sources (22 and 23) with amplitude 1, and other sources (24, 4, and 5 have amplitude 1; 18 has amplitude 0.8; 7 has amplitude 0.5). Novel events are mapped to auditory sources (27 and 28) with amplitude 1.5, motor sources (22 and 23) with amplitude 0.5, and other sources (24, 4, and 5 have amplitude 1; 18 has amplitude 1.2; 29 and 30 have amplitude 0.8). The DMN source (8) is negatively activated with all three sources having amplitude -0.3. Spike events are mapped only to the CSF sources (14 and 15) with amplitude 1.

All sources have unique events that occur with a probability of 0.2 at each TR. For sources not of interest (no task modulation), the unique event amplitude is 1. For task-modulated sources, unique events are added with small amplitudes (0.2 to 0.5) so that components responding to the same events have similar but not identical activation. CSF sources have smaller unique events (amplitude of 0.05).

From the event time series, TCs are generating using convolution with a canonical HRF, defined as a difference between two gamma functions. In general, each component may be generated with a different set of hemodynamic parameters. Here, we use the same set of parameters for all but four sources. The two frontal components (4 and 5) have a 1 second onset delay relative to other sources. The two CSF components (14 and 15) use a spike model which has much faster dynamics

that the canonical HRF (e.g., peak at 2 seconds rather than ∼6 seconds).

The baseline intensity for all subjects is $b_i = 800$, though in general this value can be different for each subject. The percent signal change is centered at $g_{ic} = 3$ with a standard deviation of 0.25 over for all sources, except CSF (Sources 14 and 15 are 1.2 times larger) and white matter (Sources 16 and 17 are 0.5 times larger) to approximate statistical moments of real data (Fig. 3). The tissue-type modifier $\mathbf{u}$ designates WM at 0.8, CSF at 1.2, and frontal WM at 1.15, relative to GM intensity of 1. Motion is added with a maximum translation of 0.02 of the image length, and a maximum rotation of 5 degrees. Subject 1 moves up to 0.5 of the maximum. Rician noise is added to the data of each subject to reach the appropriate CNR level, which is uniformly distributed over subjects from 0.65 to 2.

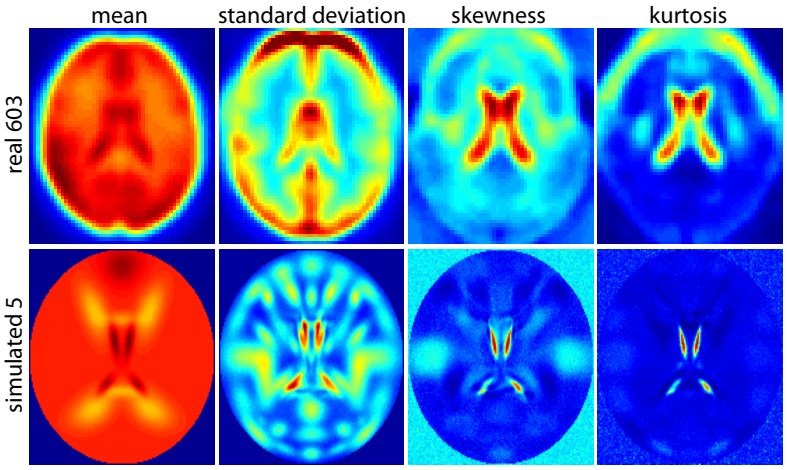


Figure 3: This dataset of 5 simulated subjects has the first four central statistical moments (mean, variance, skewness, and kurtosis) approximately matching those of a real dataset with 603 subjects collected on a Siemens 3T scanner (Allen et al., 2011b).

5.2 GUI and parameter file

The following screenshots and descriptions demonstrate the implementation of the AOD simulation in `SimTB`. Below each GUI panel are the corresponding lines of code from the parameter file, `'experiment_params_aod'`. The full parameter file is also provided in Section 8.2. We highlight operations that cannot be performed within the GUI, such as defining tissue type levels and TC model parameters, and refer GUI users to the appropriate How To examples.

Step 0: `simtb`

Calling `simtb` starts the GUI.

```
>> simtb
```

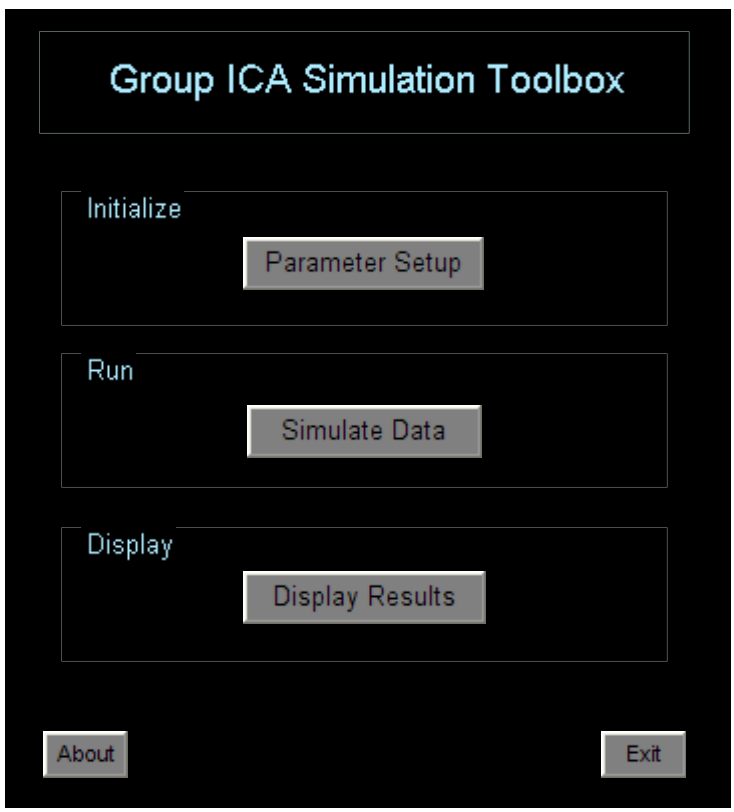
Initialize Guides the user through setting the parameters listed in Section 4. The core of the GUI.

Run Loads a parameter structure from file and runs the simulation.

Display Currently not functional.

About Provides details about the toolbox.

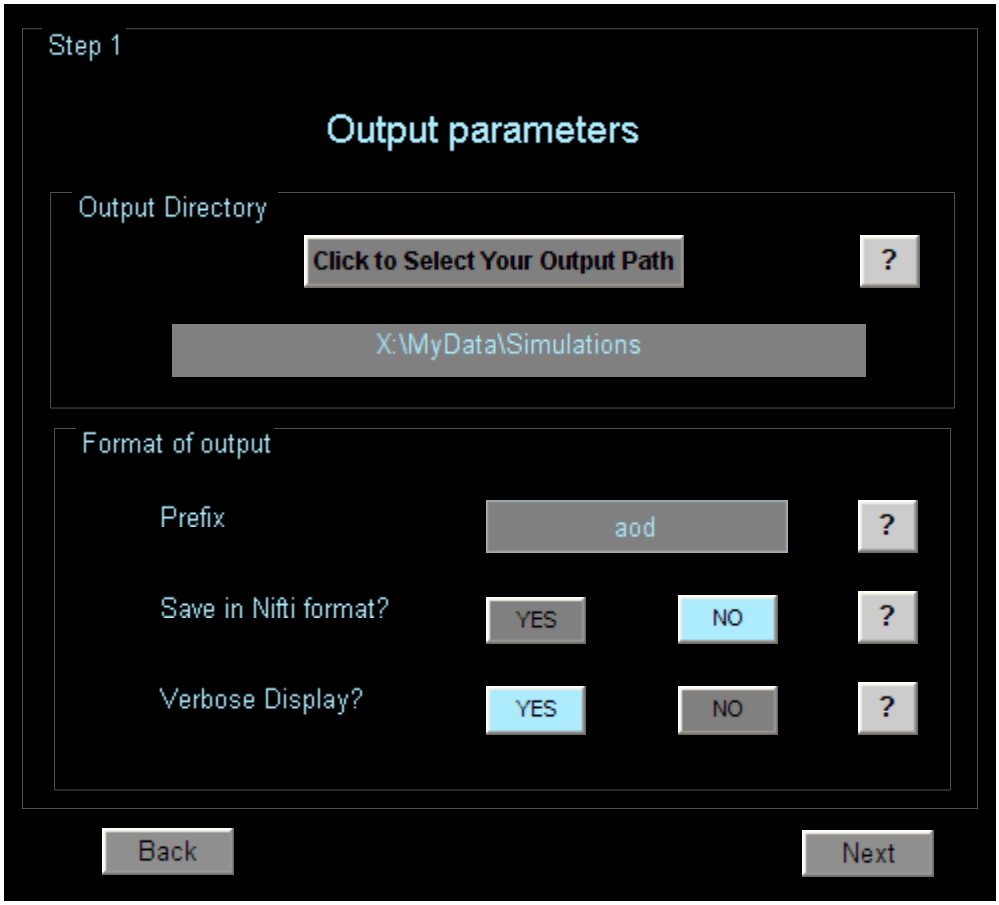
Exit Quits the GUI.



5.3 Initialize: Parameter Setup

Step 1: Output parameters

Set up the output directory, output file prefixes (multiple simulations can be saved in same directory and kept distinct with the file prefix), file format (.mat or .nii), and whether figures should be created and saved as part of the simulation.



The above GUI window can be equivalently coded as below from `experiment_params_aod.m`.

```
%% OUTPUT PARAMETERS
%-------------------------------------------------------------------------------
% Directory to save simulation parameters and output
out_path = 'X:\MyData\Simulations';
% Prefix for saving output
prefix = 'aod';
% FLAG to write data in NIFTI format rather than matlab
saveNII_FLAG = 0;
% Option to display output and create figures throughout the simulations
verbose_display = 1;
%-------------------------------------------------------------------------------
```

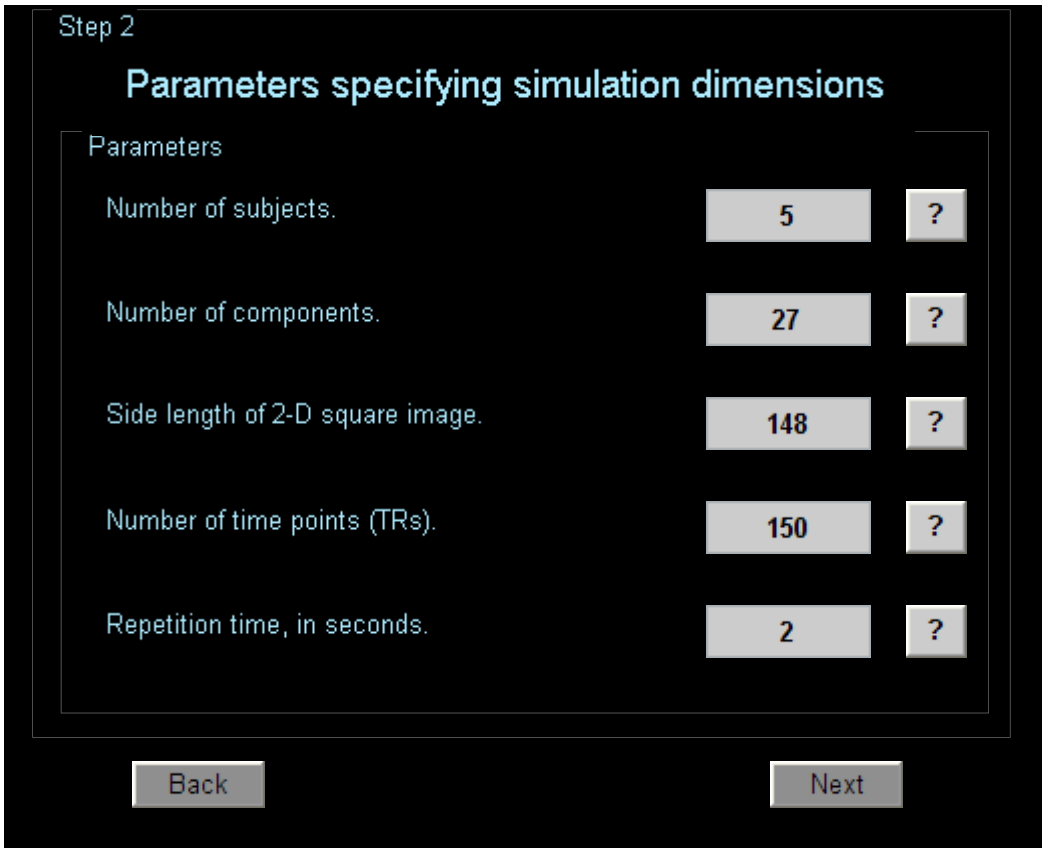
Parameter file only: Setting a random number seed

A seed can be specified in the parameter file to set the state of MATLAB random number generators. Setting the seed allows one to exactly reproduce parameter structures and simulations. This feature is not available in the GUI since random values can be chosen multiple times or in different orders. However, once the parameter structure is created and saved, simulations will be identical whenever they are run.

```
%% RANDOMIZATION
%-------------------------------------------------------------------------
%seed = round(sum(100*clock));  % randomizes parameter values
seed = 3571;                    % choose seed for repeatable simulation
simtb_rand_seed(seed);          % set the seed
%-------------------------------------------------------------------------
```

Step 2: Parameters specifying simulation dimensions

Enter the simulation dimensions. Note that the number of components entered below will be updated later if the user selects a different number of sources.

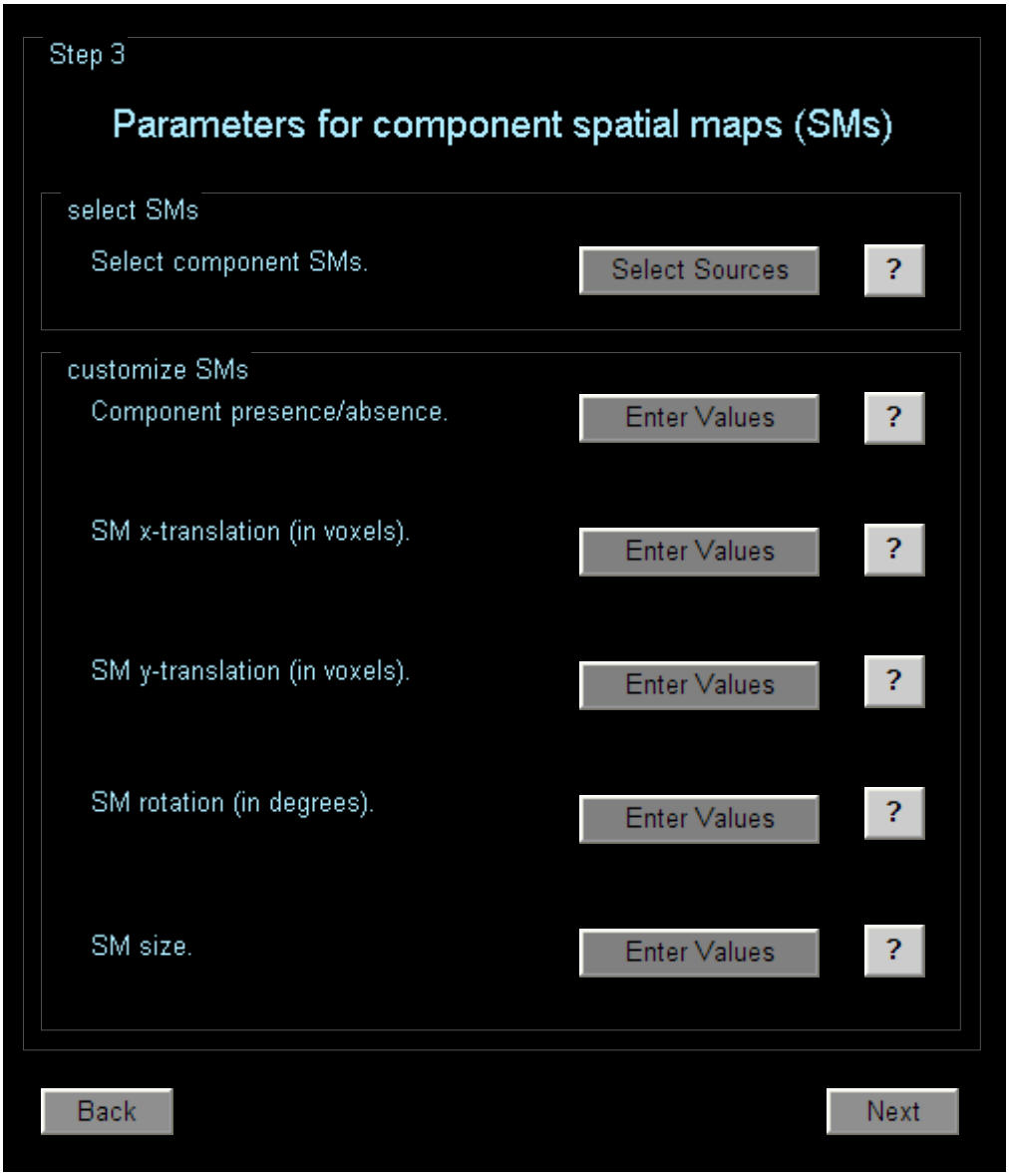


The above GUI window can be equivalently coded as below from `experiment_params_aod.m`.

```
%% SIMULATION DIMENSIONS
%-------------------------------------------------------------------------
M  = 5;   % number of subjects
% nC is the number of components defined below, nC = length(SM_source_ID);
nV = 148; % number of voxels; dataset will have [nV x nV] voxels.
nT = 150; % number of time points
TR = 2;   % repetition time
%-------------------------------------------------------------------------
```

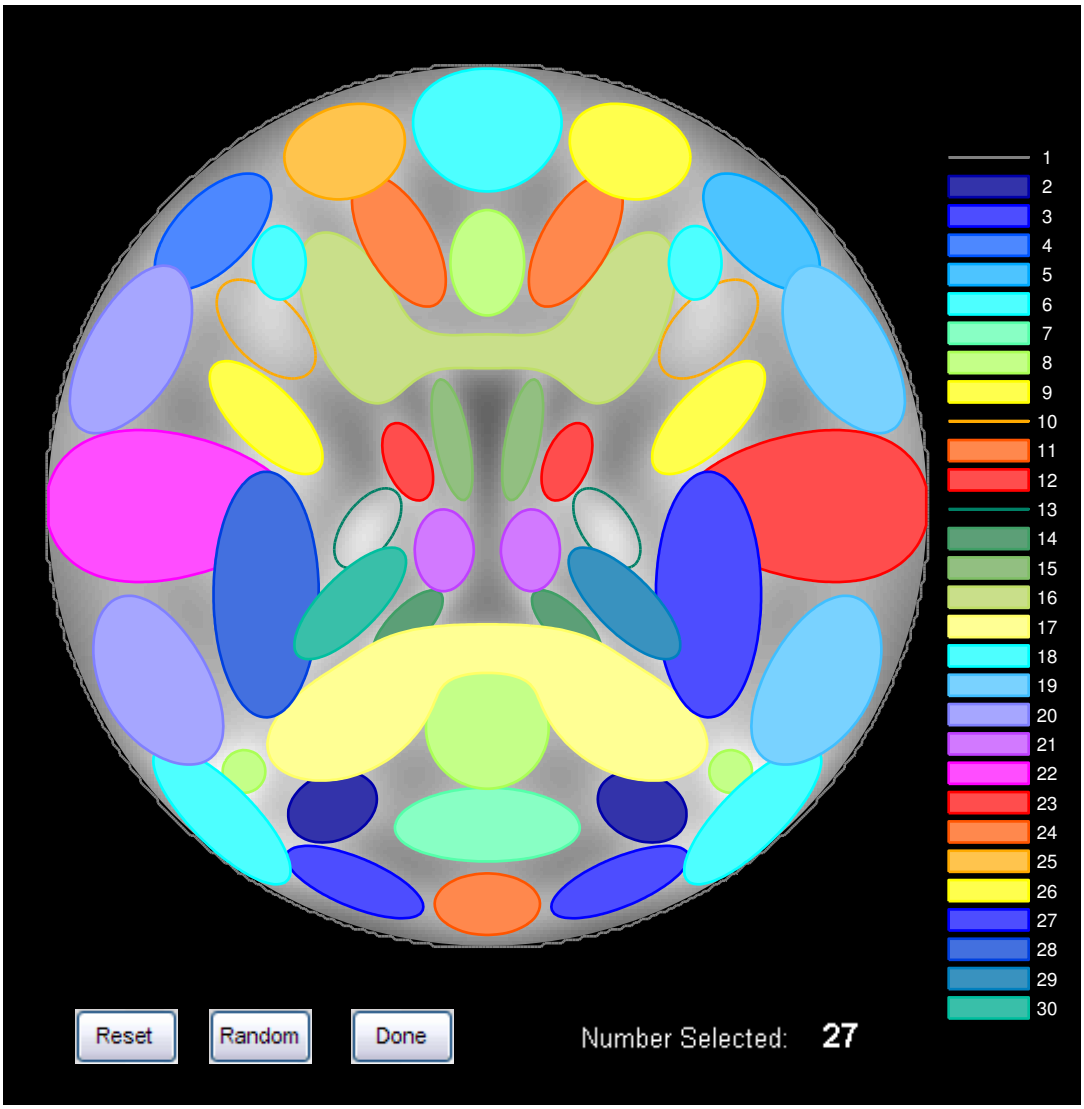
Step 3: Parameters for component spatial maps (SMs)

Main panel for SM parameters which opens multiple subsequent panels.



Step 3a: Select component SMs

Twenty-seven components have been selected by clicking within the component boundaries. Filled components have been selected, while outlined components (1, 10, and 13) have been excluded.



The above GUI window can be equivalently coded as below from `experiment_params_aod.m`.

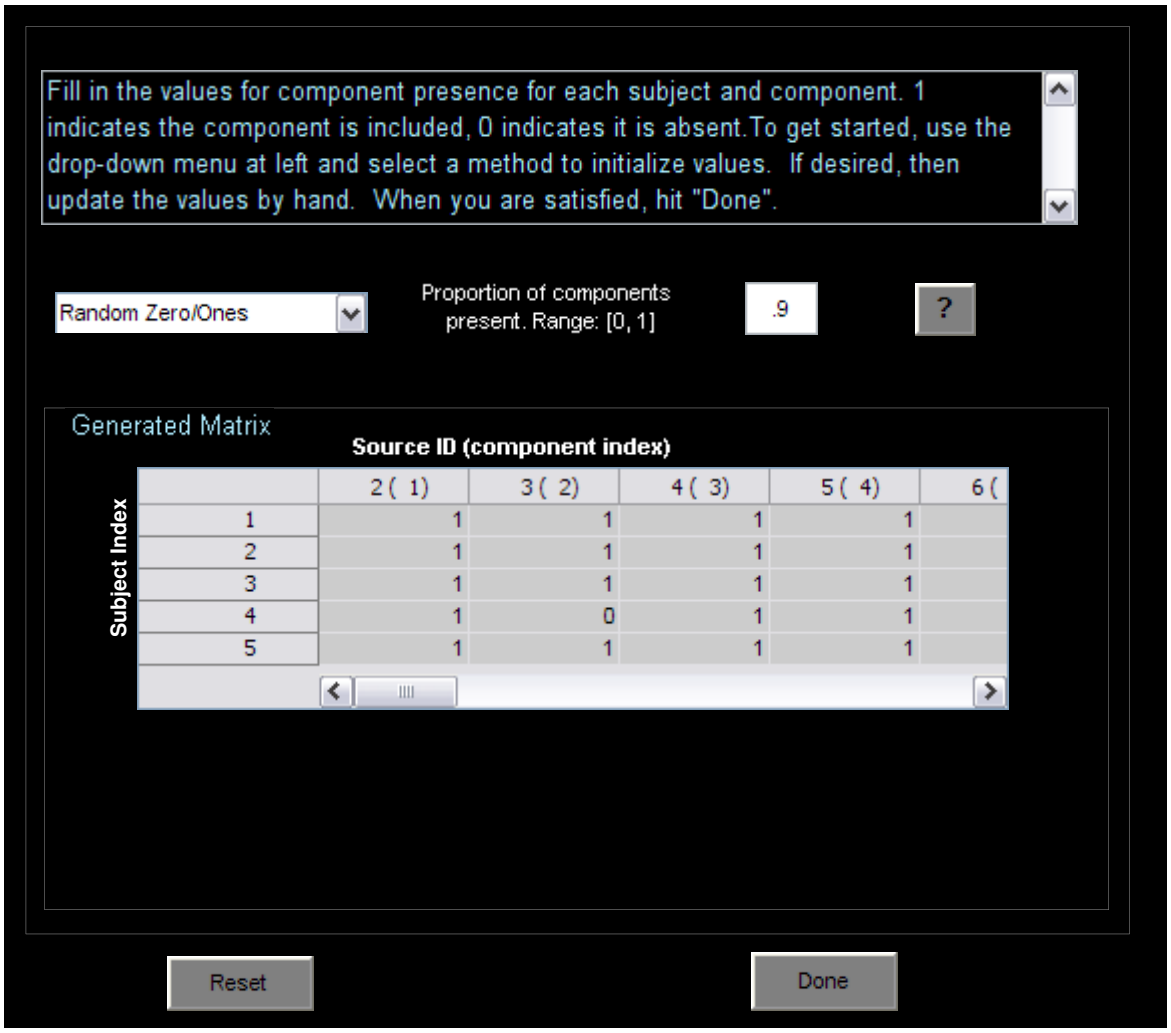
```
%% SPATIAL SOURCES
%-----------------------------------------------------------------------------
% Choose the sources. To launch a stand-alone GUI:
% >> simtb_pickSM
SM_source_ID = [    2  3  4  5  6  7  8  9     ...
                11 12    14 15 16 17 18 19 20  ...
                21 22 23 24 25 26 27 28 29 30]; % all but (1, 10, 13)

nC = length(SM_source_ID);  % number of components
```

For our simulation it is convenient to label selected components in the parameter file that will be assigned different conditions. This can be done as in the code below.

```
% LABEL COMPONENTS
% Here, we label components or component groups that may be used later
% Auditory: strong positive activation for all task events
comp_AUD1  = find(SM_source_ID == 27);
comp_AUD2  = find(SM_source_ID == 28);
% DMN: negative activation to task events
comp_DMN   = find(SM_source_ID ==  8);
% Bilateral frontal: positive activation to for targets and novels
comp_BF    = find(SM_source_ID == 24);
% Frontal: 1 second temporal delay from bilateral frontal
comp_F1    = find(SM_source_ID ==  4);
comp_F2    = find(SM_source_ID ==  5);
% Precuneus: activation only to targets
comp_P     = find(SM_source_ID ==  7);
% Dorsal Attention Network: activation to novels more than targets
comp_DAN   = find(SM_source_ID == 18);
% Hippocampus: activation only to novels
comp_H1    = find(SM_source_ID == 29);
comp_H2    = find(SM_source_ID == 30);
% (Sensori)Motor: activation to targets and novels (weakly)
comp_M1    = find(SM_source_ID == 22);
comp_M2    = find(SM_source_ID == 23);
% CSF and white matter: unaffected by task, but has signal amplitude differences
comp_CSF1  = find(SM_source_ID == 14);
comp_CSF2  = find(SM_source_ID == 15);
comp_WM1   = find(SM_source_ID == 16);
comp_WM2   = find(SM_source_ID == 17);
% Medial Frontal: has lower baseline intensity (signal dropout)
comp_MF    = find(SM_source_ID ==  6);

% compile list of all defined components of interest
complist = [comp_AUD1 comp_AUD2 comp_DMN comp_BF  comp_F1 comp_F2 ...
            comp_P    comp_DAN  comp_H1  comp_H2  comp_M1 comp_M2 ...
            comp_CSF1 comp_CSF2 comp_WM1 comp_WM2 comp_MF];
%-------------------------------------------------------------------------------
```

Step 3a: Component presence/absence

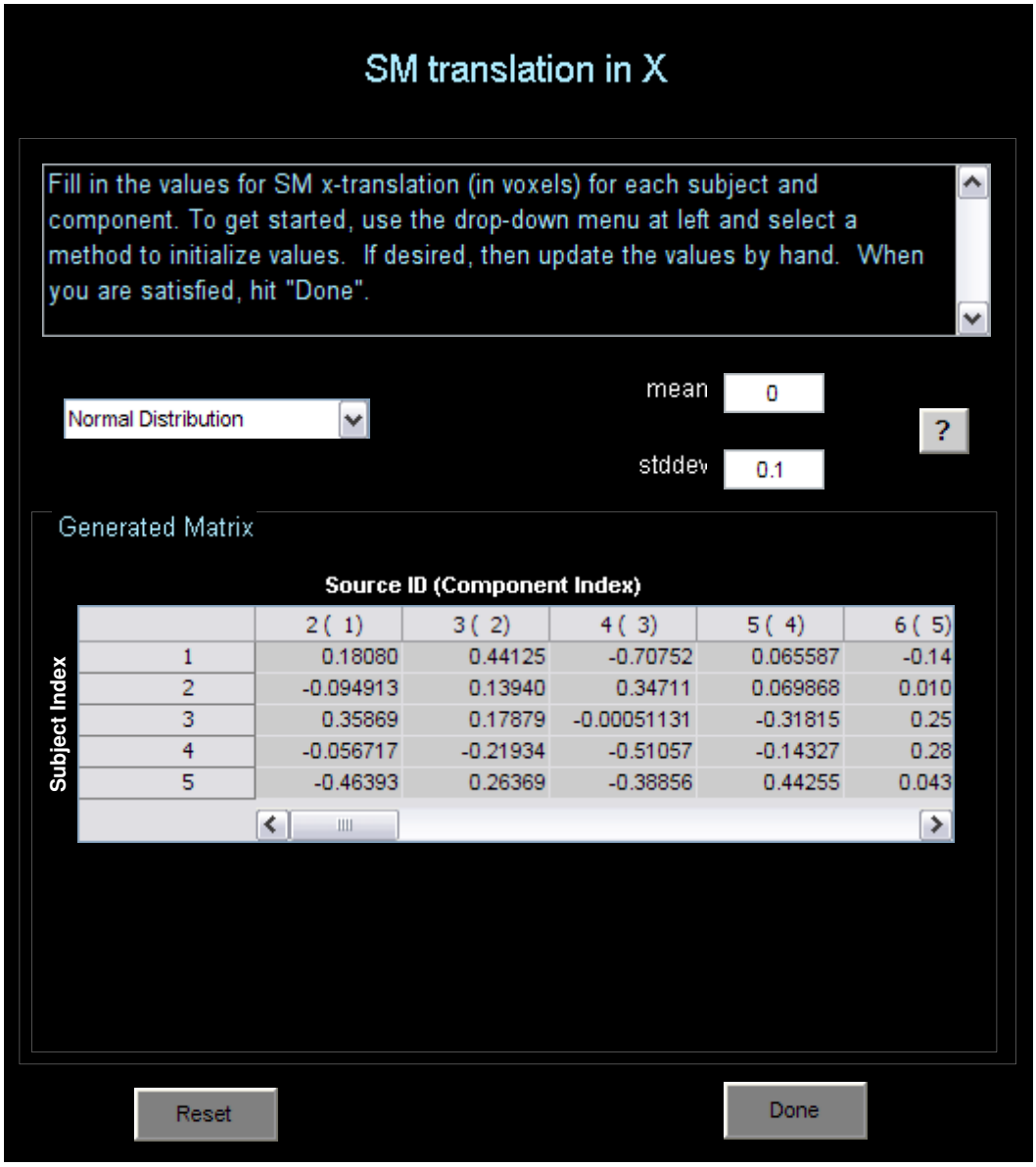
We would like components not of interest to appear with a probability of 0.9, thus we set the proportion of components present to 0.9 in the GUI. We then manually update the values for components of interest to make sure they all appear (have values of 1).



The above GUI window can be equivalently coded as below from `experiment_params_aod.m`.

```
%% COMPONENT PRESENCE
%-------------------------------------------------------------------------
% [M x nC] matrix for component presence: 1 if included, 0 otherwise
% For components not of interest there is a 90% chance of component inclusion.
SM_present = (rand(M,nC) < 0.9);
% Components of interest (complist) are included for all subjects.
SM_present(:,complist) = ones(M,length(complist));
%-------------------------------------------------------------------------
```

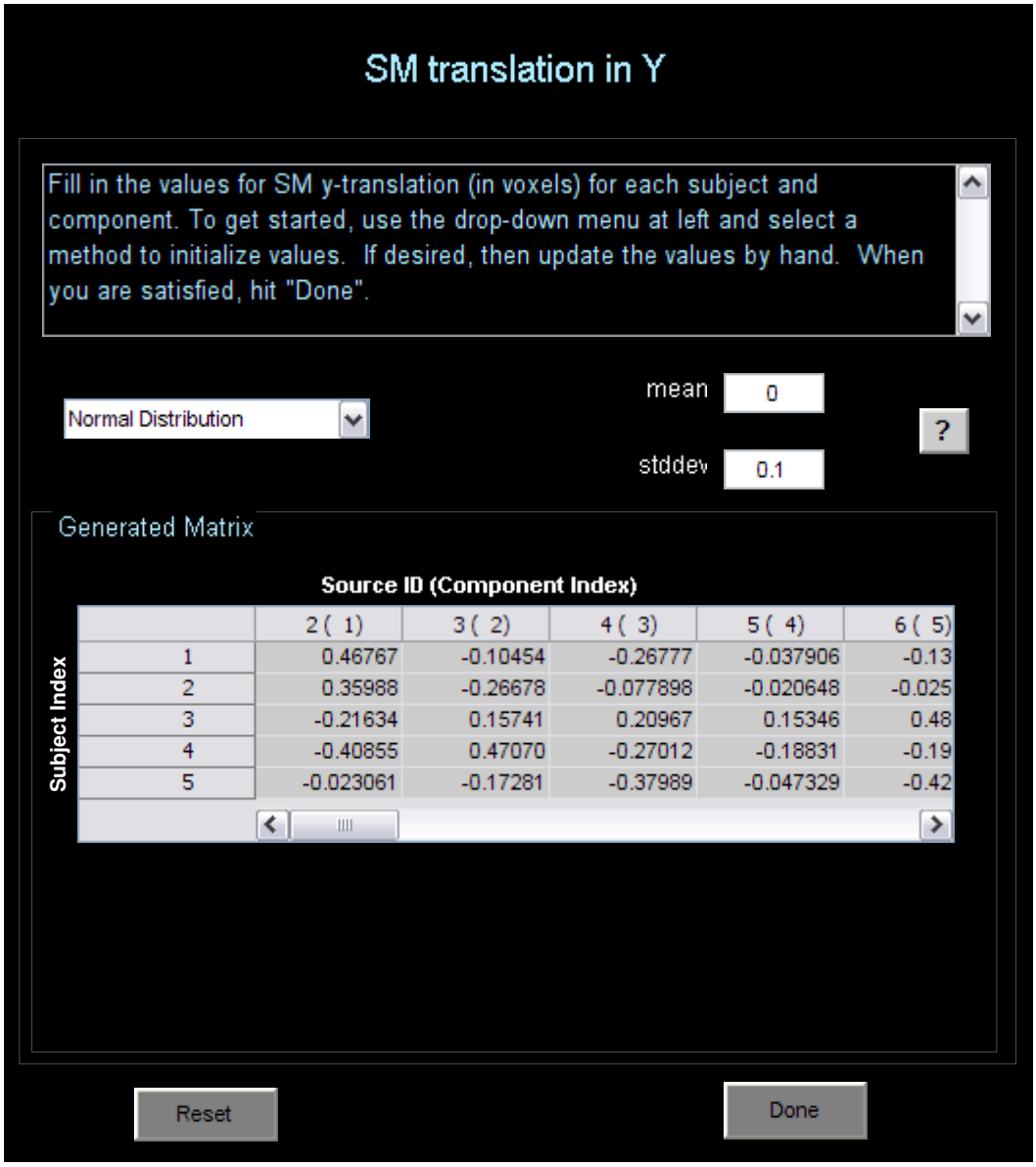
Step 3b: SM x-translation



The above GUI window can be equivalently coded as below from `experiment_params_aod.m`.

```
%% SPATIAL VARIABILITY
%-----------------------------------------------------------------------------
% Variability related to differences in spatial location and shape.
SM_translate_x = 0.1*randn(M,nC);  % Translation in x, mean 0, SD 0.1 voxels.
```

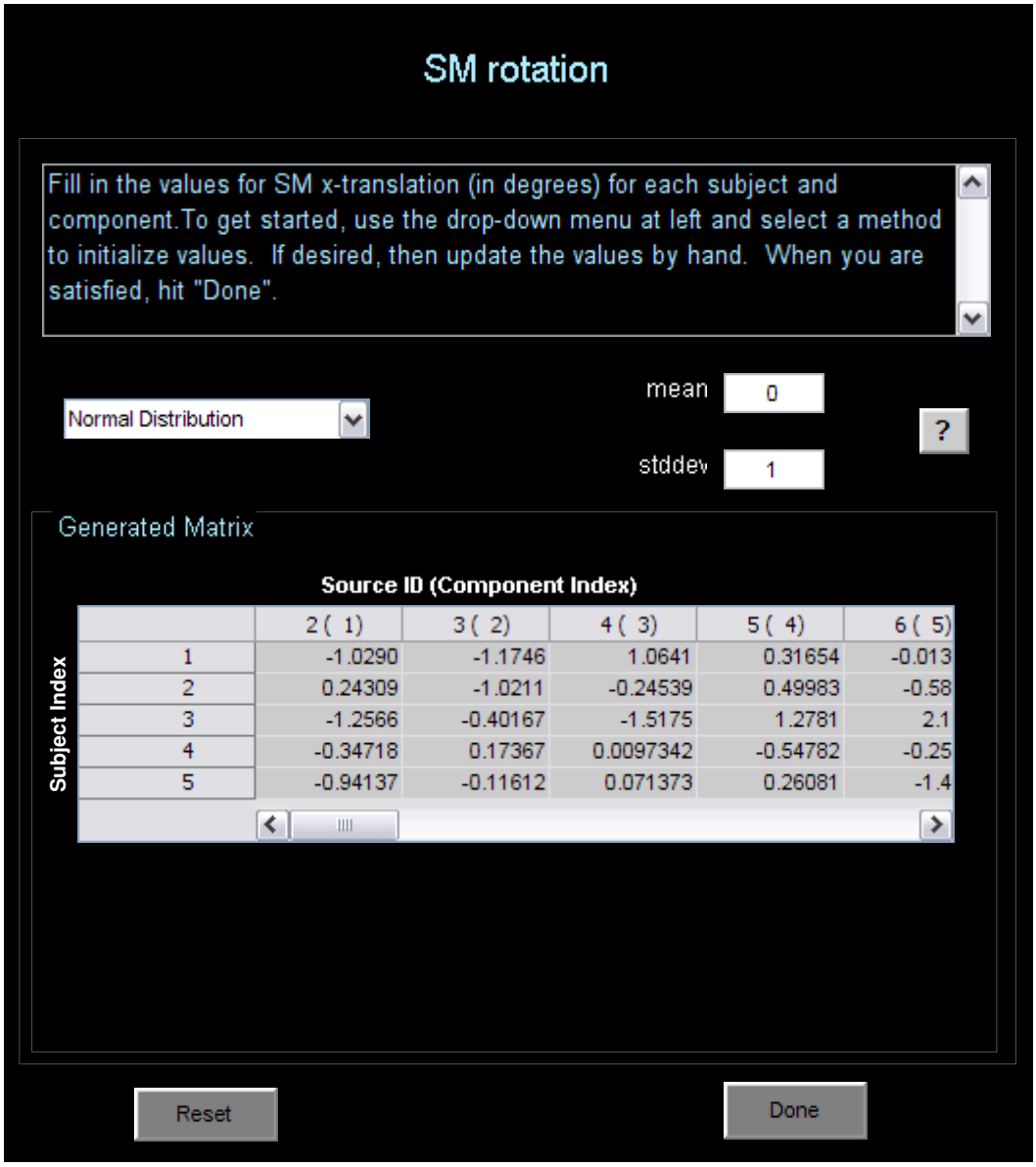
Step 3c: SM y-translation



The above GUI window can be equivalently coded as below from `experiment_params_aod.m`.

```
SM_translate_y = 0.1*randn(M,nC);  % Translation in y, mean 0, SD 0.1 voxels.
```

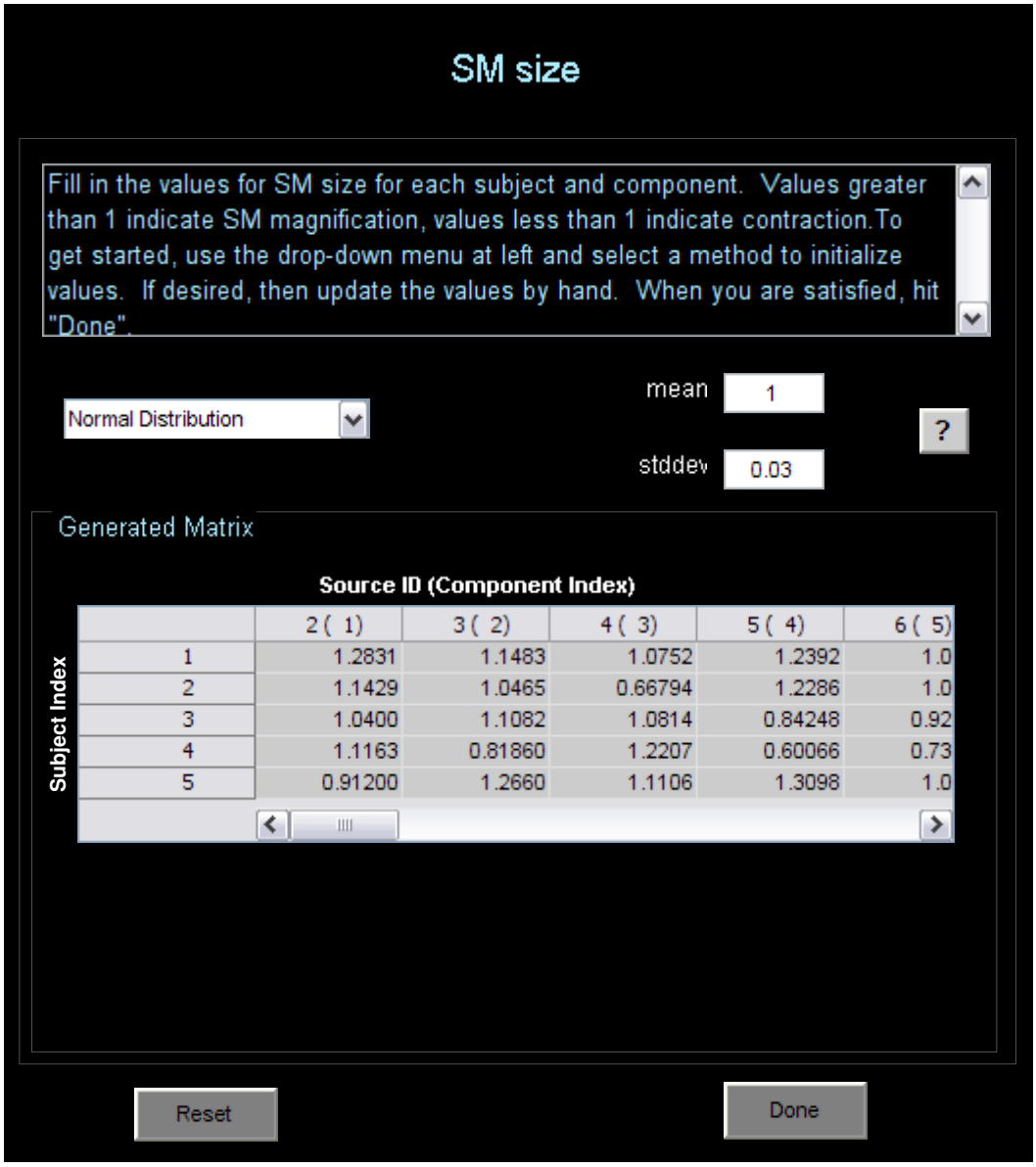
Step 3d: SM rotation



The above GUI window can be equivalently coded as below from `experiment_params_aod.m`.

```
SM_theta        = 1.0*randn(M,nC);  % Rotation, mean 0, SD 1 degree.
%                 Note that each 'activation blob' is rotated independently.
```

Step 3e: SM size/spread

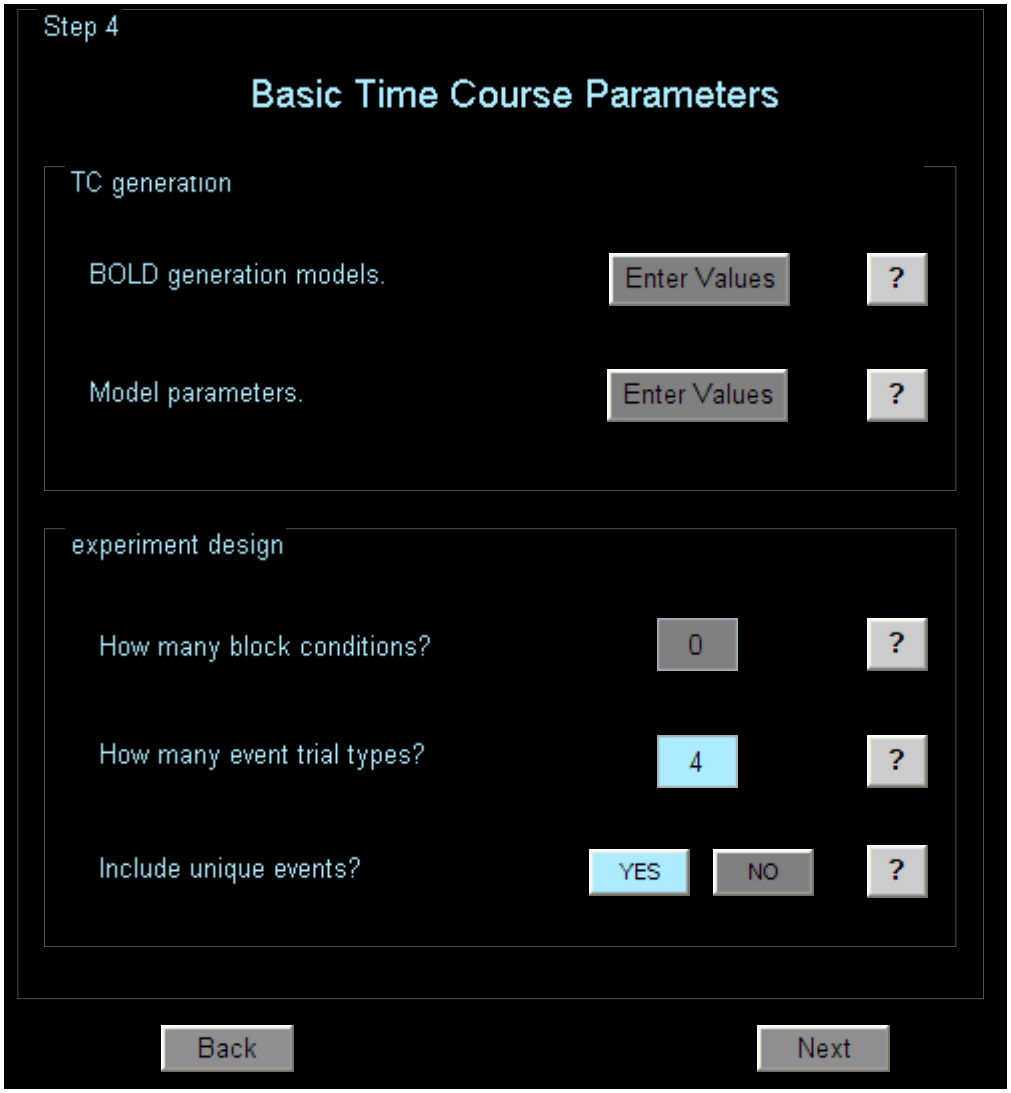


The above GUI window can be equivalently coded as below from `experiment_params_aod.m`.

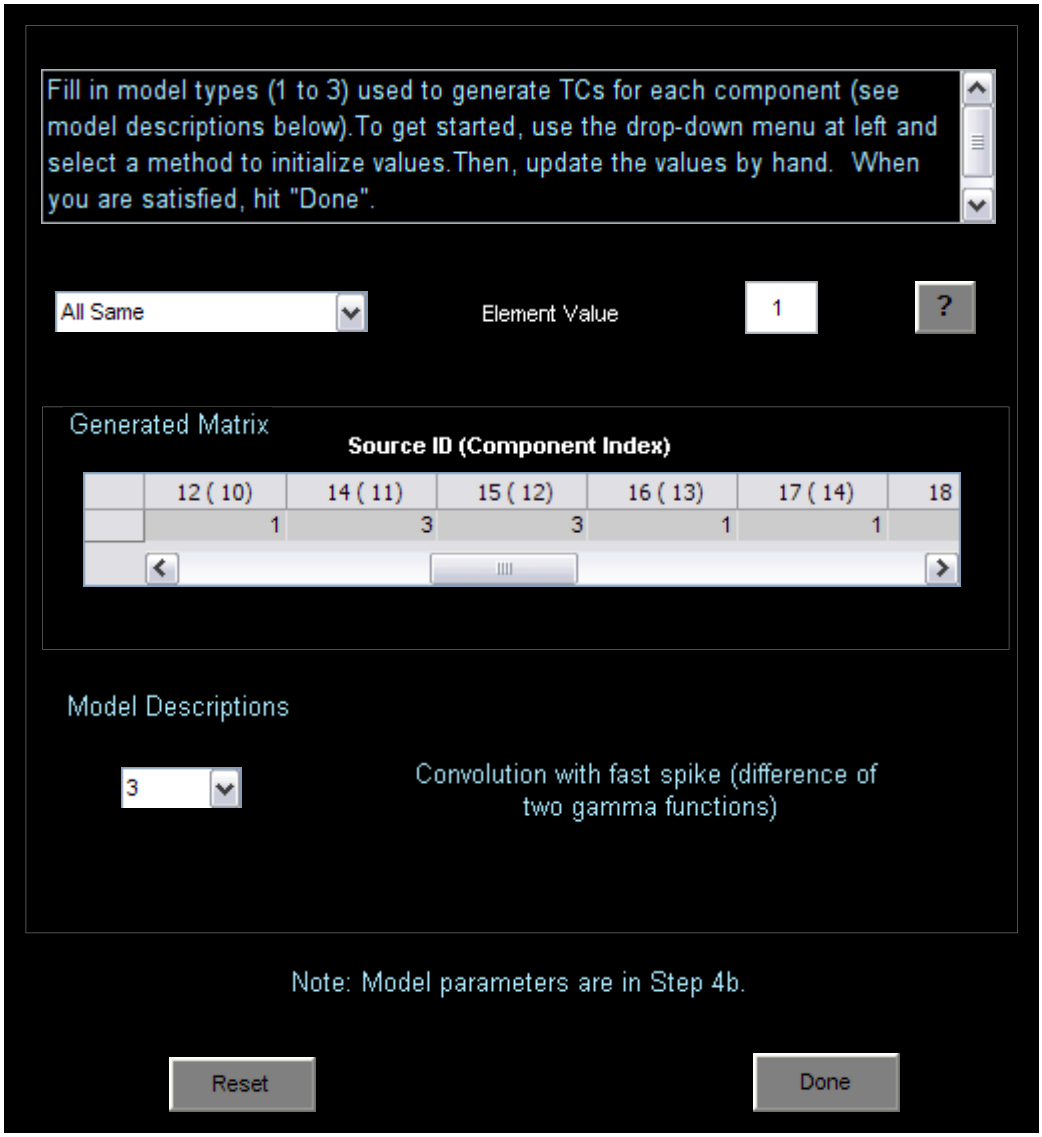
```
SM_spread = 1+0.03*randn(M,nC); % Spread < 1 is contraction, spread > 1 is expansion.
%-----------------------------------------------------------------------------
```

Step 4: Basic time course parameters

The BOLD generation model and experimental design are specified in this panel. Increasing the block conditions or event trial types to 1 or more will automatically open a window to enter the associated parameters.



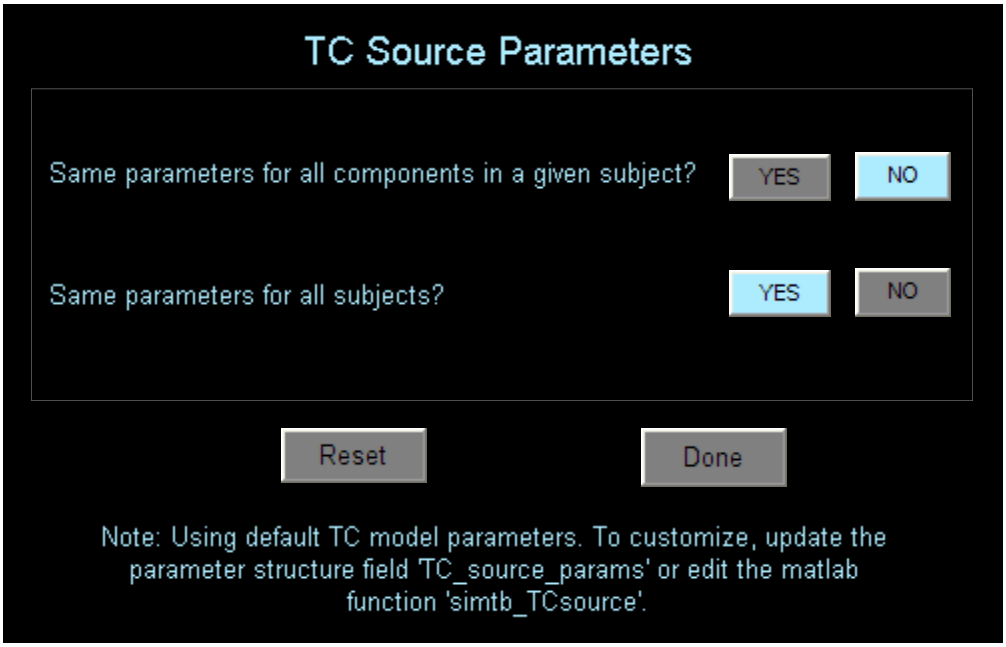
Step 4a: BOLD generation model



The above GUI window can be equivalently coded as below from `experiment_params_aod.m`.

```
%% TC GENERATION
%-------------------------------------------------------------------------------
% Choose the model for TC generation.  To see defined models:
% >> simtb_countTCmodels

TC_source_type = ones(1,nC);     % convolution with HRF for most components
% to make statistical moments of data look more like real data
TC_source_type([comp_CSF1 comp_CSF2]) = 3; % spike model for CSF
```

Step 4b: Model parameters (for TC generation)

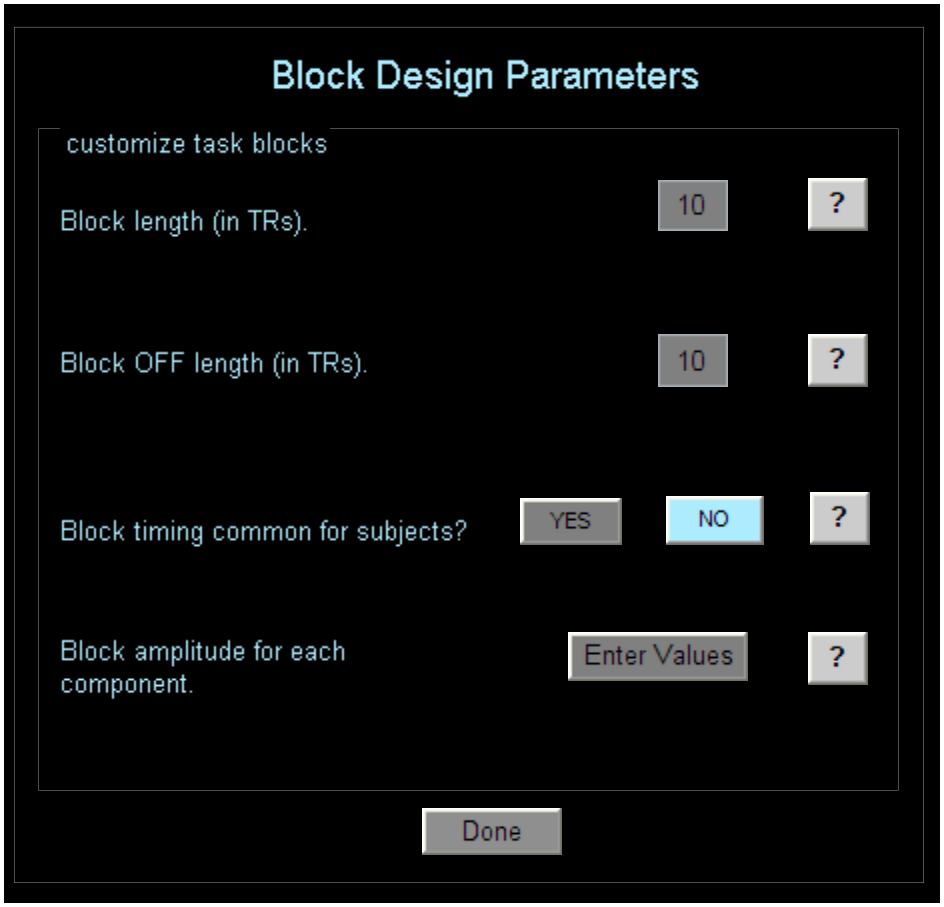


The default settings include variability in TC source parameters for each subject/source. In the GUI, we elect to have different parameters across sources but identical across subjects. In the parameter file, we illustrate how to specify particular parameters for every subject/source. See Section 6.3 for examples on how to further customize parameters.

```
TC_source_params = cell(M,nC);  % initialize the cell structure
% Use the same HRF for all subjects and relevant components
P(1) = 6;     % delay of response (relative to onset)
P(2) = 16;    % delay of undershoot (relative to onset)
P(3) = 1;     % dispersion of response
P(4) = 1;     % dispersion of undershoot
P(5) = 6;     % ratio of response to undershoot
P(6) = 0;     % onset (seconds)
P(7) = 32;    % length of kernel (seconds)
[TC_source_params{:}] = deal(P);

% Implement 1 second onset delay for components comp_F1 and comp_F2
P(6) = P(6) + 1;  % delay by 1s
[TC_source_params{:,[comp_F1 comp_F2]}] = deal(P);

sourceType = 3; % CSF components use spike model
% Generate a random set of parameters for TC model 3
[tc_dummy, MDESC, P3, PDESC] = simtb_TCsource(1, 1, sourceType);
% Assign identical parameters for model 3 to all subjects
[TC_source_params{:,[comp_CSF1 comp_CSF2]}] = deal(P3);
%-------------------------------------------------------------------------------
```

Step 4c: Block design parameters

Note: Blocks are not used in this experimental design. This panel shown for completeness but is not functional in this simulation. In the parameter file, `TC_block_n = 0` specifies that blocks are not used.

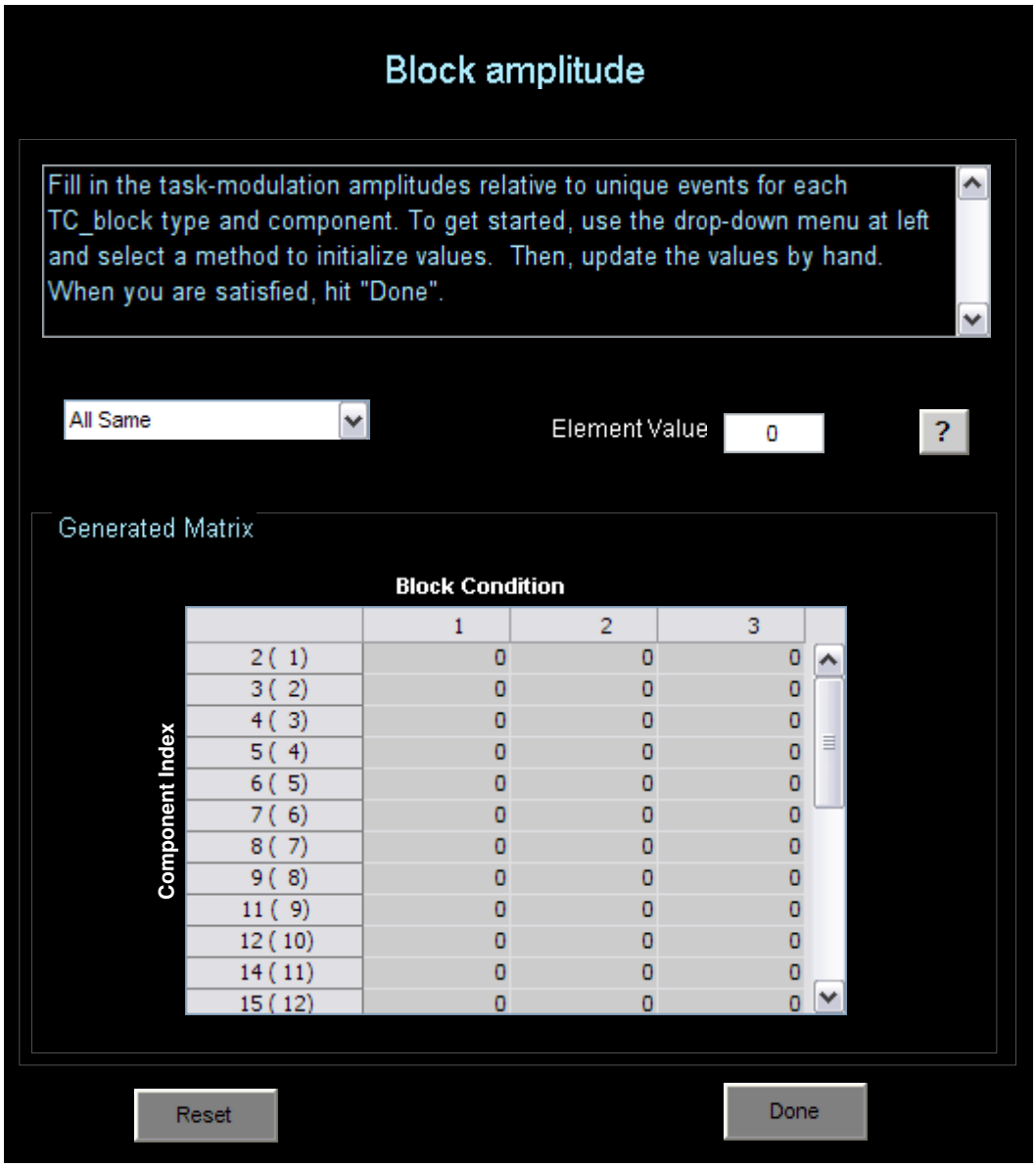


The above GUI window can be equivalently coded as below from `experiment_params_aod.m`.

```
%% EXPERIMENT DESIGN
%-----------------------------------------------------------------------------
% BLOCKS
% No blocks for this experiment
TC_block_n = 0;           % Number of blocks [set = 0 for no block design]
% Note that if TC_block_n = 0 the rest of these parameters are irrelevant
TC_block_same_FLAG = 0;   % 1 = block structure same for all subjects
                          % 0 = otherwise order will be randomized
TC_block_length = 10;     % length of each block (in samples)
TC_block_ISI    = 10;     % length of OFF inter-stimulus-intervals (in samples)
```

Step 4c1: Block amplitude

Note: Blocks are not used in this experimental design. This panel shown for completeness but is not functional in this simulation. For an example of task-modulation amplitudes, refer to the event related design in the next few pages.



The above GUI window can be equivalently coded as below from `experiment_params_aod.m`.

```
TC_block_amp     = [];     % [nC x TC_block_n] matrix of task-modulation amplitudes
```

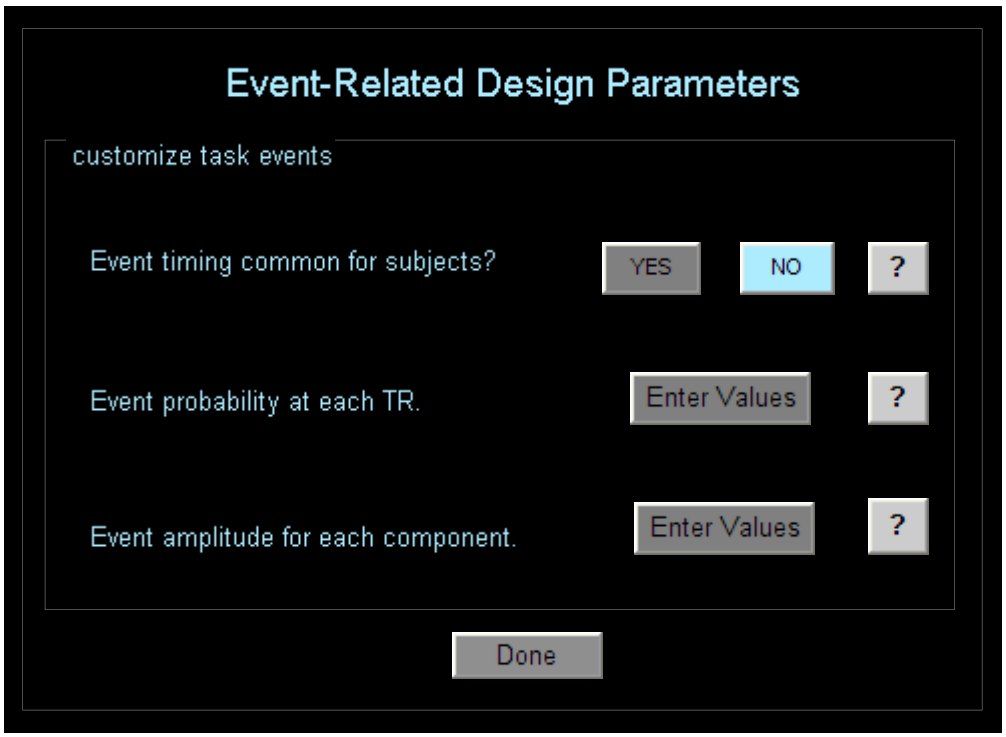
Step 4d: Event-related design parameters

Timing Yes = Event timing is the same for each subject. No = Event timing is random across subjects subjects.

Probability For each TR, an event either occurs or not based on the event probability defined. The sum of the probabilities should be less than or equal to 1.

Amplitude Amplitude of each event type for each source.

The number of events (4) was specified in a previous panel (Step 4).



The above GUI window can be equivalently coded as below from `experiment_params_aod.m`.

```
% EVENTS
TC_event_n = 4;            % Number of event types (0 for no event-related design)
                           % 1: standard tone
                           % 2: target tone
                           % 3: novel tone
                           % 4: 'spike' events in CSF (not related to task)
TC_event_same_FLAG = 0;    % 1=event timing will be the same for all subjects
```

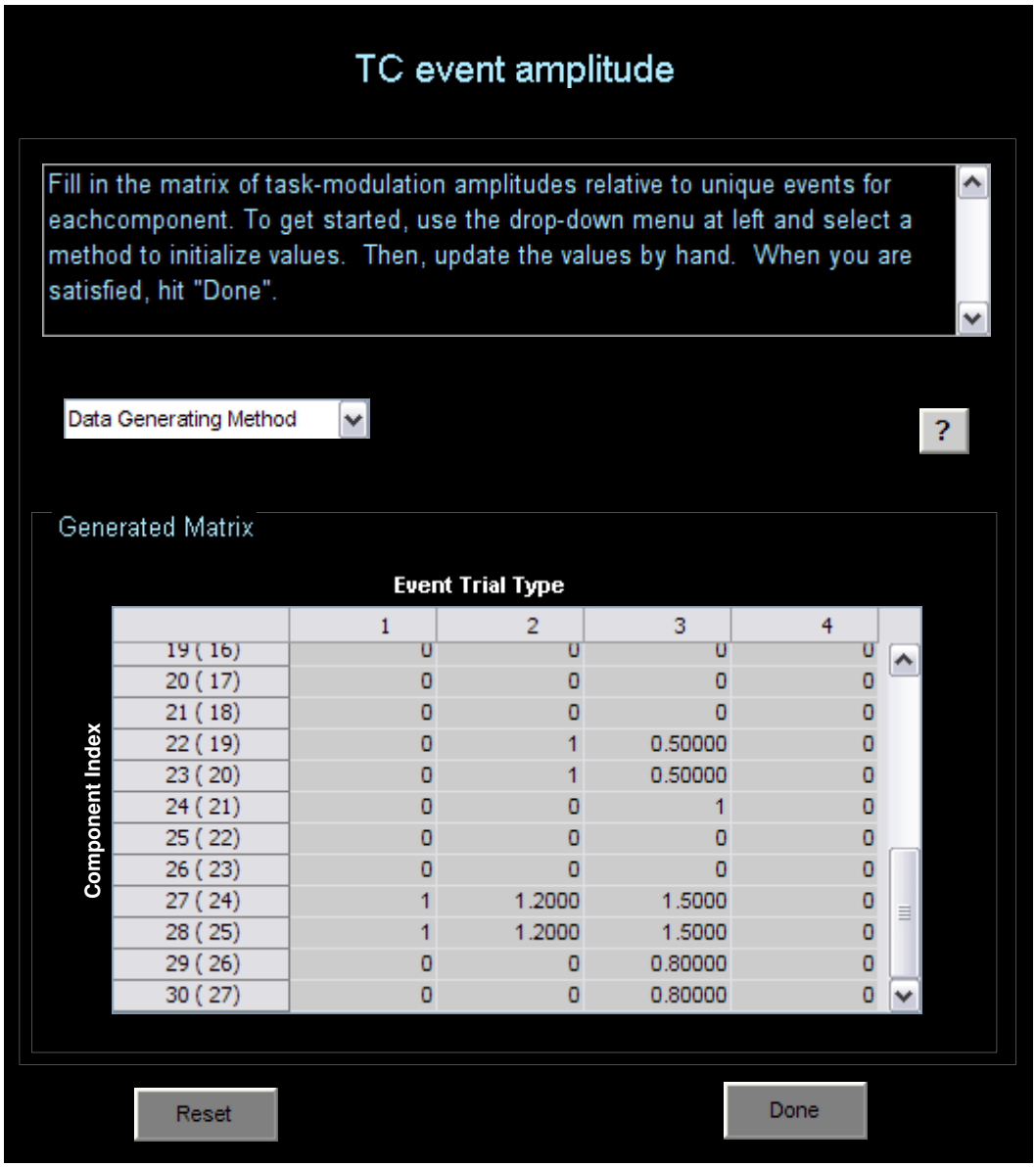
Step 4d1: Task event probability

The above GUI window can be equivalently coded as below from `experiment_params_aod.m`.

```
% event probabilities (0.6 standards, 0.075 targets and novels, 0.05 CSF spikes)
TC_event_prob = [0.6, 0.075, 0.075, 0.05];  % an 8:1:1 ratio
```

Step 4d2: Task event amplitude

Values for TC event amplitude can be randomized using the drop-down menu, but for this simulation must be entered in manually. Note that component index 27 (Source ID 24) refers to `comp_AUD1`.



Component Index	1	2	3	4
19 (16)	0	0	0	0
20 (17)	0	0	0	0
21 (18)	0	0	0	0
22 (19)	0	1	0.50000	0
23 (20)	0	1	0.50000	0
24 (21)	0	0	1	0
25 (22)	0	0	0	0
26 (23)	0	0	0	0
27 (24)	1	1.2000	1.5000	0
28 (25)	1	1.2000	1.5000	0
29 (26)	0	0	0.80000	0
30 (27)	0	0	0.80000	0

The above GUI window can be equivalently coded as below from `experiment_params_aod.m`.

```
% initialize [nC x TC_event_n] matrix of task-modulation amplitudes
TC_event_amp = zeros(nC,TC_event_n);
% event type 1: standard tone
TC_event_amp([comp_AUD1 comp_AUD2],                1) = 1.0; % moderate task-modulation
TC_event_amp([comp_BF comp_F1 comp_F2 comp_DAN], 1) = 0.7; % mild
TC_event_amp([comp_DMN],                           1) =-0.3; % negative weak
% event type 2: target tone
TC_event_amp([comp_AUD1 comp_AUD2],                2) = 1.2; % strong
```

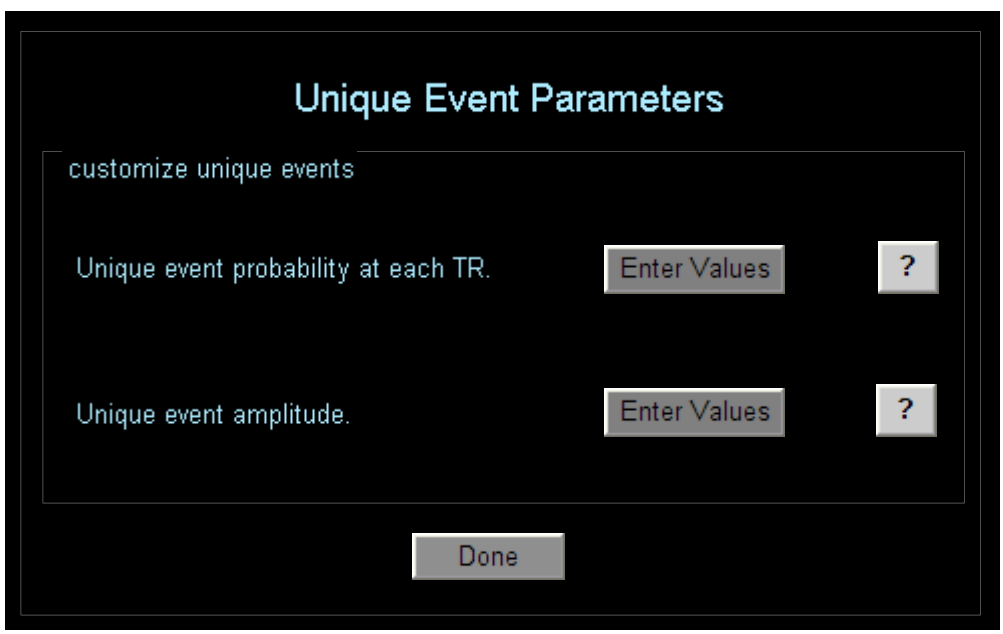
```
TC_event_amp([comp_BF comp_F1 comp_F2],          2) = 1.0; % moderate
TC_event_amp([comp_DAN],                         2) = 0.8; % mild
TC_event_amp([comp_P],                           2) = 0.5; % weak
TC_event_amp([comp_M1 comp_M2],                  2) = 1.0; % moderate
TC_event_amp([comp_DMN],                         2) =-0.3; % negative weak
% event type 3: novel tone
TC_event_amp([comp_AUD1 comp_AUD2],              3) = 1.5; % very strong
TC_event_amp([comp_BF comp_F1 comp_F2],          3) = 1.0; % moderate
TC_event_amp([comp_DAN],                         3) = 1.2; % strong
TC_event_amp([comp_H1 comp_H2],                  3) = 0.8; % mild
TC_event_amp([comp_M1 comp_M2],                  3) = 0.5; % weak
TC_event_amp([comp_DMN],                         3) =-0.3; % negative weak
% event type 4: 'spikes' in CSF (not related to task)
TC_event_amp([comp_CSF1 comp_CSF2],              4) = 1.0; % moderate
%-----------------------------------------------------------------------------
```

Step 4e: Unique event parameters

Probability For each TR, a unique event either occurs or not based on the event probability defined.

Amplitude Amplitude of each event type for each source.

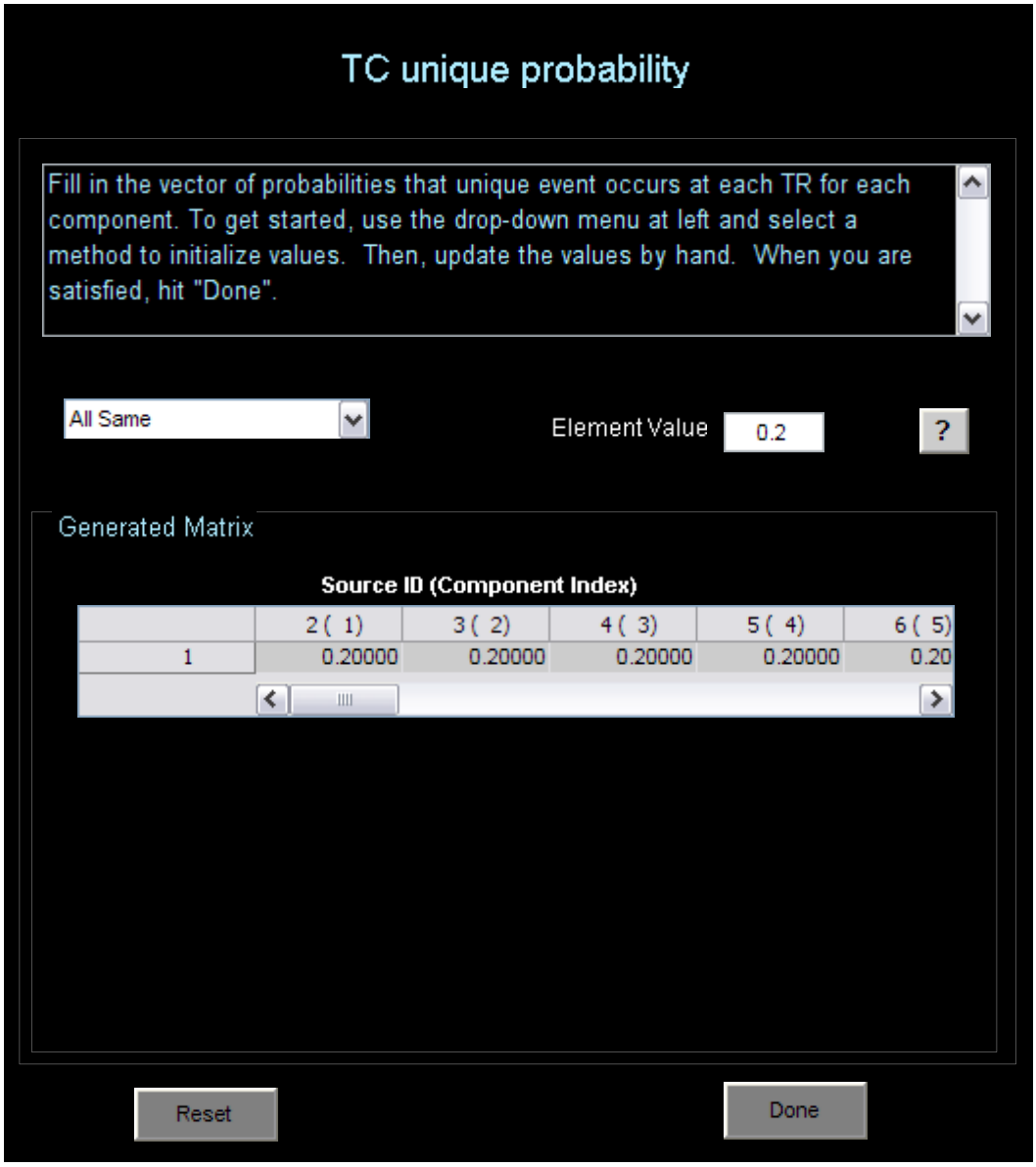
Unique events flag was set to 'YES' in the main TC panel (Step 4).



The above GUI window can be equivalently coded as below from `experiment_params_aod.m`.

```
%% UNIQUE EVENTS
%-----------------------------------------------------------------------------
TC_unique_FLAG = 1; % 1 = include unique events
```

Step 4e1: Unique event probability

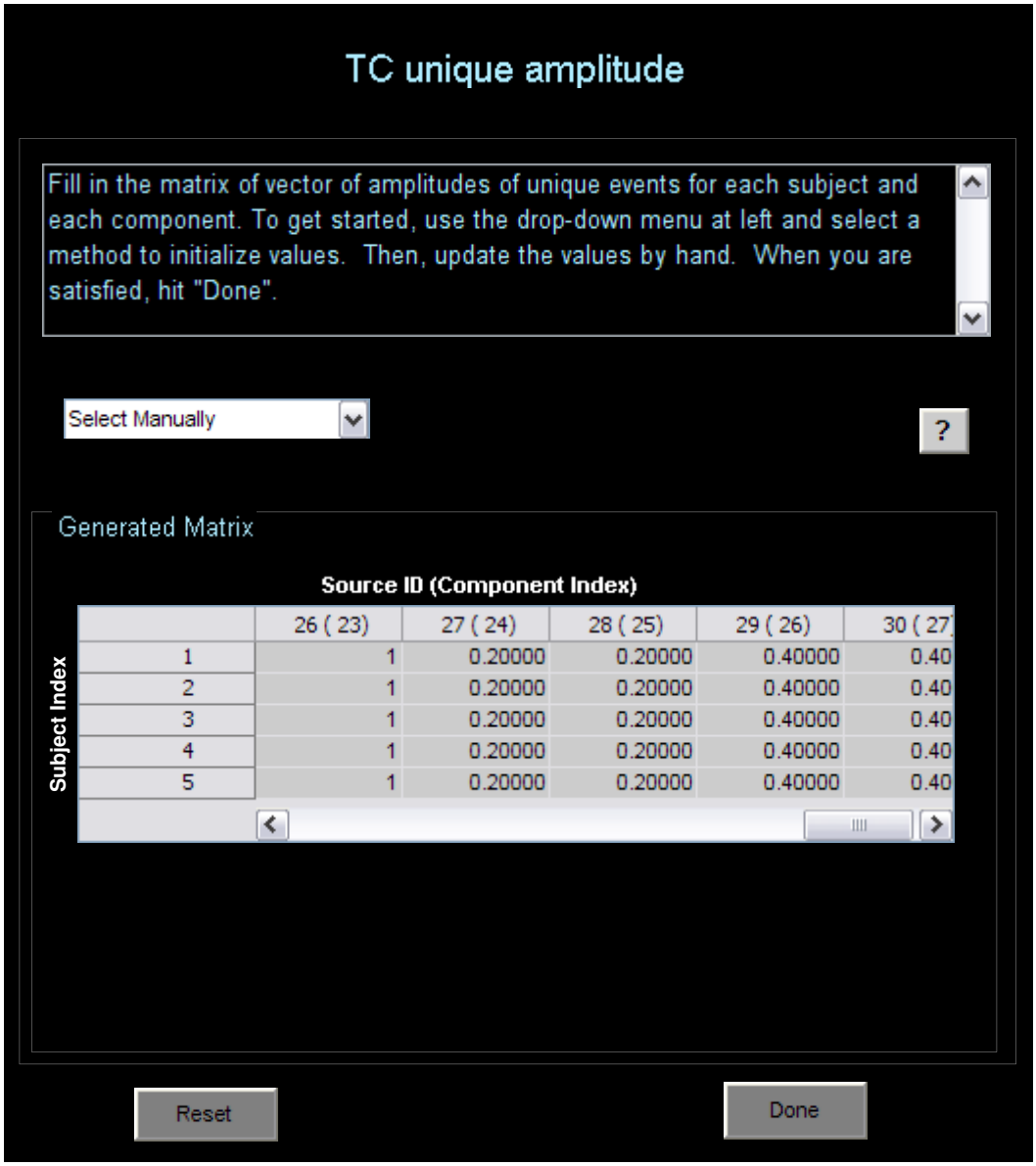


The above GUI window can be equivalently coded as below from `experiment_params_aod.m`.

```
TC_unique_prob = 0.2*ones(1,nC); % [1 x nC] prob of unique event at each TR
```

Step 4e2: Unique event amplitude

Unique event amplitudes are reduced for components of interest.



The above GUI window can be equivalently coded as below from `experiment_params_aod.m`.

```
TC_unique_amp  = ones(M,nC);      % [M x nC] matrix of amplitude of unique events
% smaller unique activations for task-modulated and CSF components
TC_unique_amp(:,[comp_AUD1 comp_AUD2])              = 0.2;
TC_unique_amp(:,[comp_BF comp_F1 comp_F2])          = 0.3;
TC_unique_amp(:,[comp_DAN])                         = 0.5;
TC_unique_amp(:,[comp_P])                           = 0.5;
TC_unique_amp(:,[comp_M1 comp_M2])                  = 0.2;
TC_unique_amp(:,[comp_H1 comp_H2])                  = 0.4;
TC_unique_amp(:,[comp_DMN])                         = 0.3;
TC_unique_amp(:,[comp_CSF1 comp_CSF2])              = 0.05; %very small
%-----------------------------------------------------------------------------
```

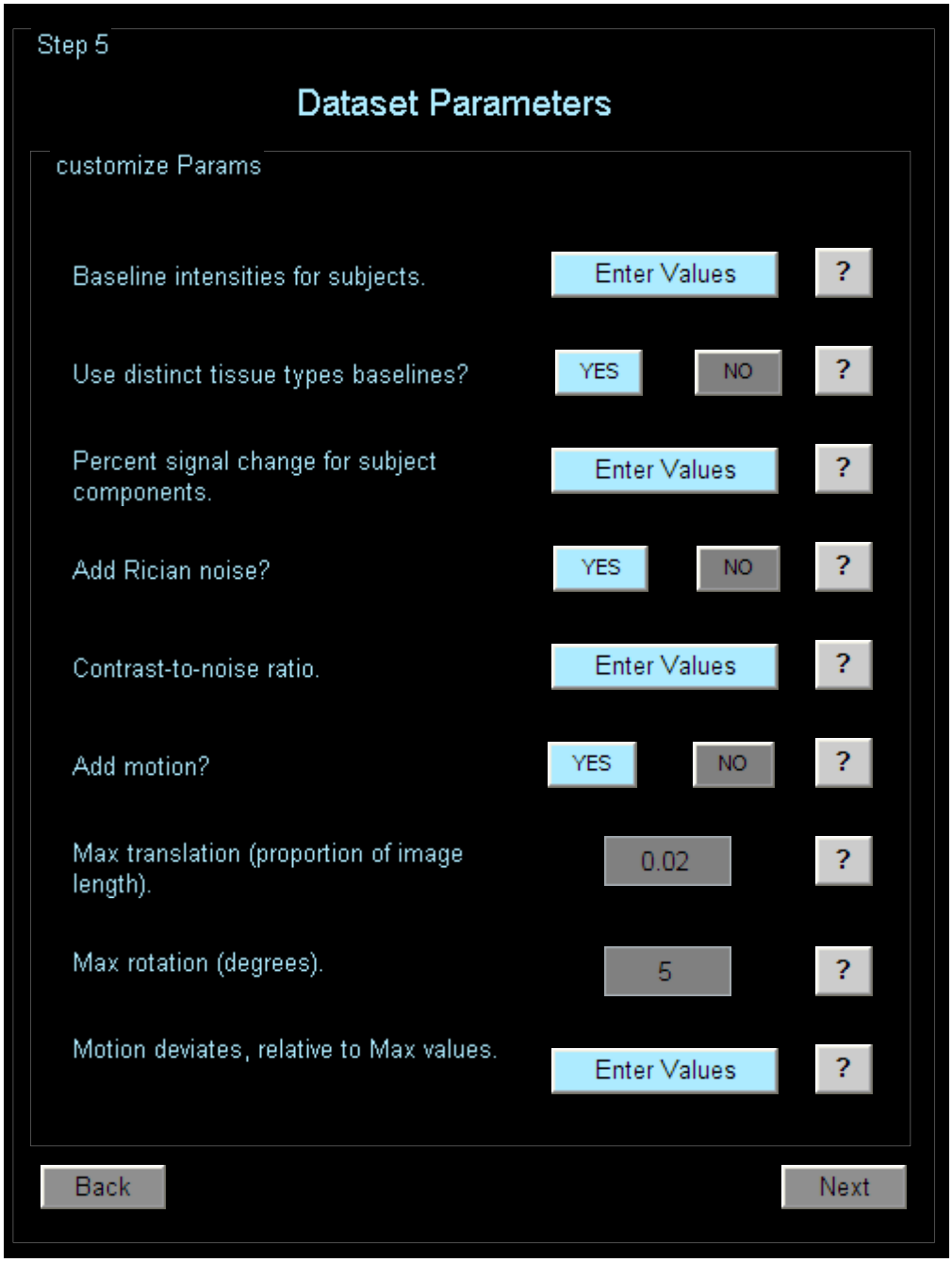
Step 5: Dataset parameters

Baseline An arbitrary signal baseline (such as 800).

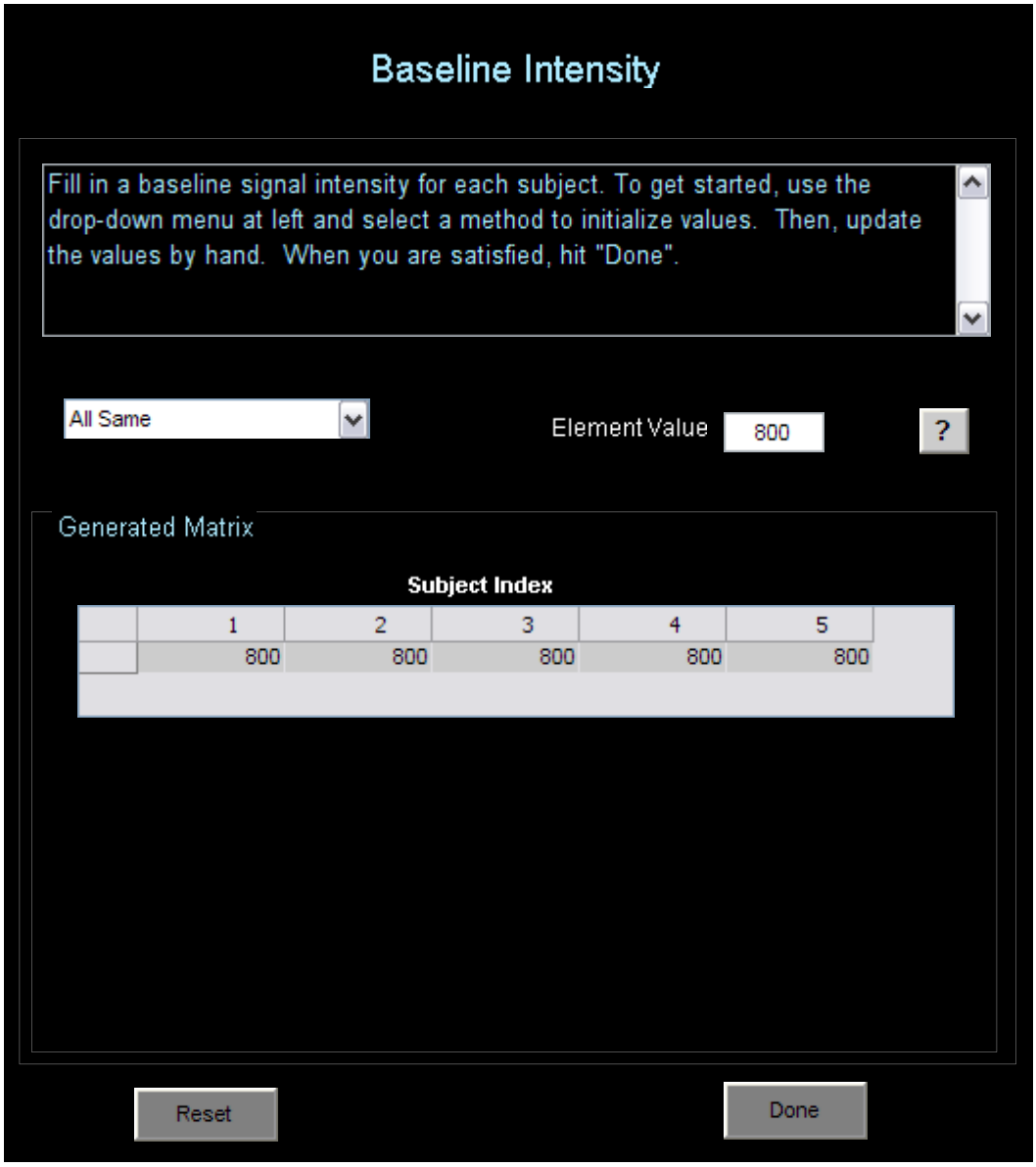
Tissue types Should the dataset baseline vary due to differences in tissue type (white/gray matter, CSF, etc.)?

Noise If adding Rician noise to dataset, set the CNR for each subject.

Motion Will subjects move? If so, how much is the maximum movement over the run, and how much is each subject allowed to move?



Step 5a: Baseline parameters

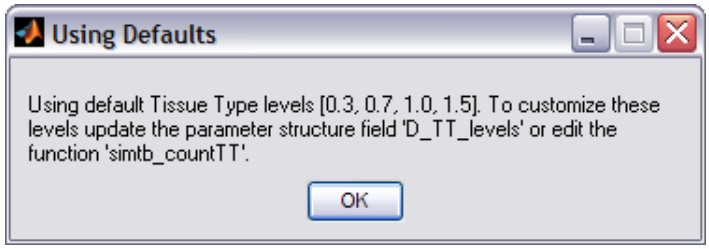


The above GUI window can be equivalently coded as below from `experiment_params_aod.m`.

```
%% DATASET BASELINE
%-------------------------------------------------------------------------
% [1 x M] vector of baseline signal intensity for each subject
D_baseline = 800*ones(1,M); % [1 x M] vector of baseline signal intensity
%-------------------------------------------------------------------------
```

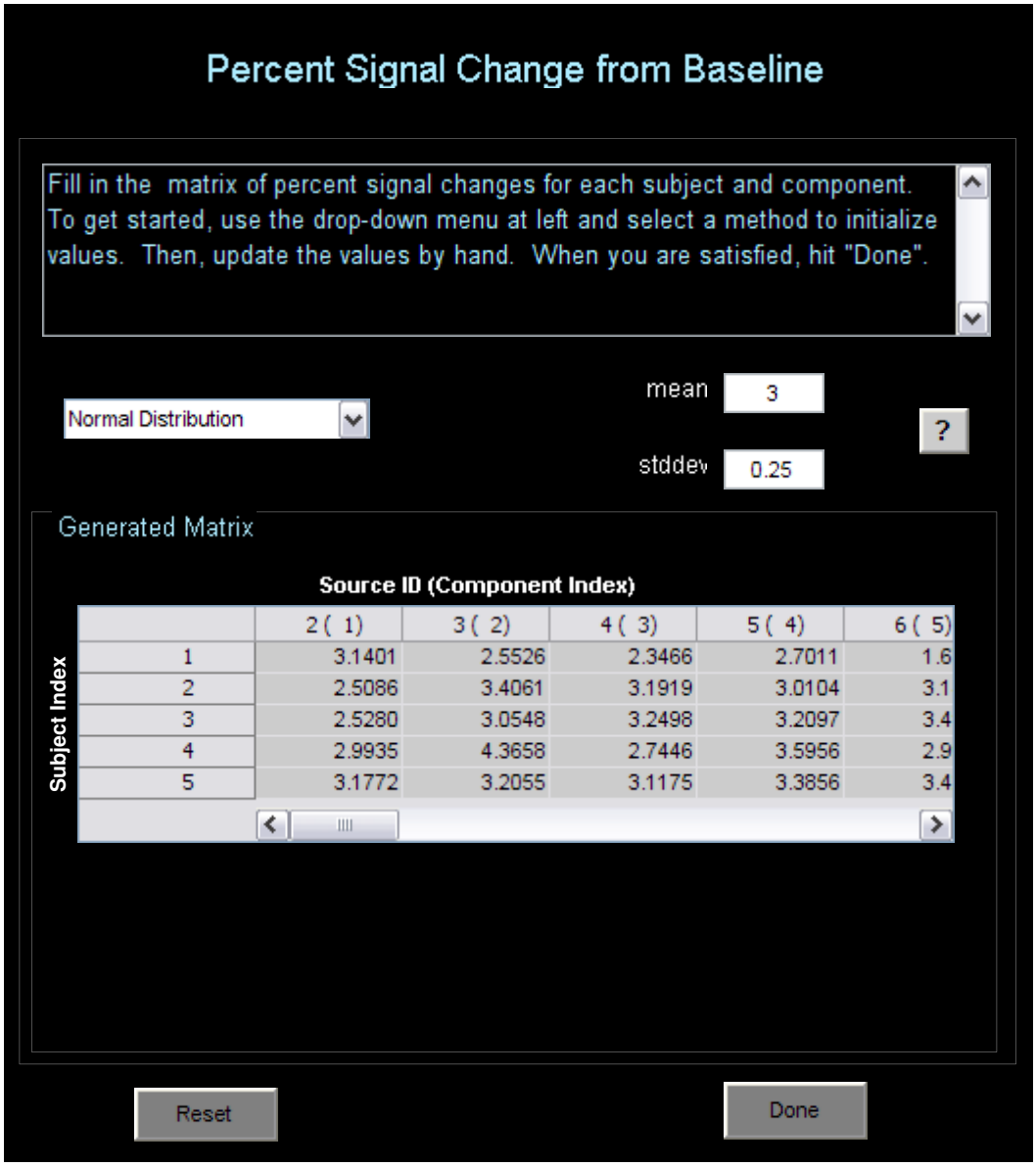
Step 5-: Setting tissue type parameters

If the tissue type model is being used, tissue type levels will be set to their defaults and the window below will be displayed. For manual setting of tissue types levels, see Section 6.6.




```
%% TISSUE TYPES
%-------------------------------------------------------------------------------
% FLAG to include different tissue types (distinct baselines in the data)
D_TT_FLAG = 1;                      % if 0, baseline intensity is constant
D_TT_level = [1.15, 0.8, 1, 1.2]; % TT fractional intensities
% To see/modify definitions for tissue profiles:
% >> edit simtb_SMsource.m
%-------------------------------------------------------------------------------
```

Step 5b: Percent signal change from baseline



The above GUI window can be equivalently coded as below from `experiment_params_aod.m`.

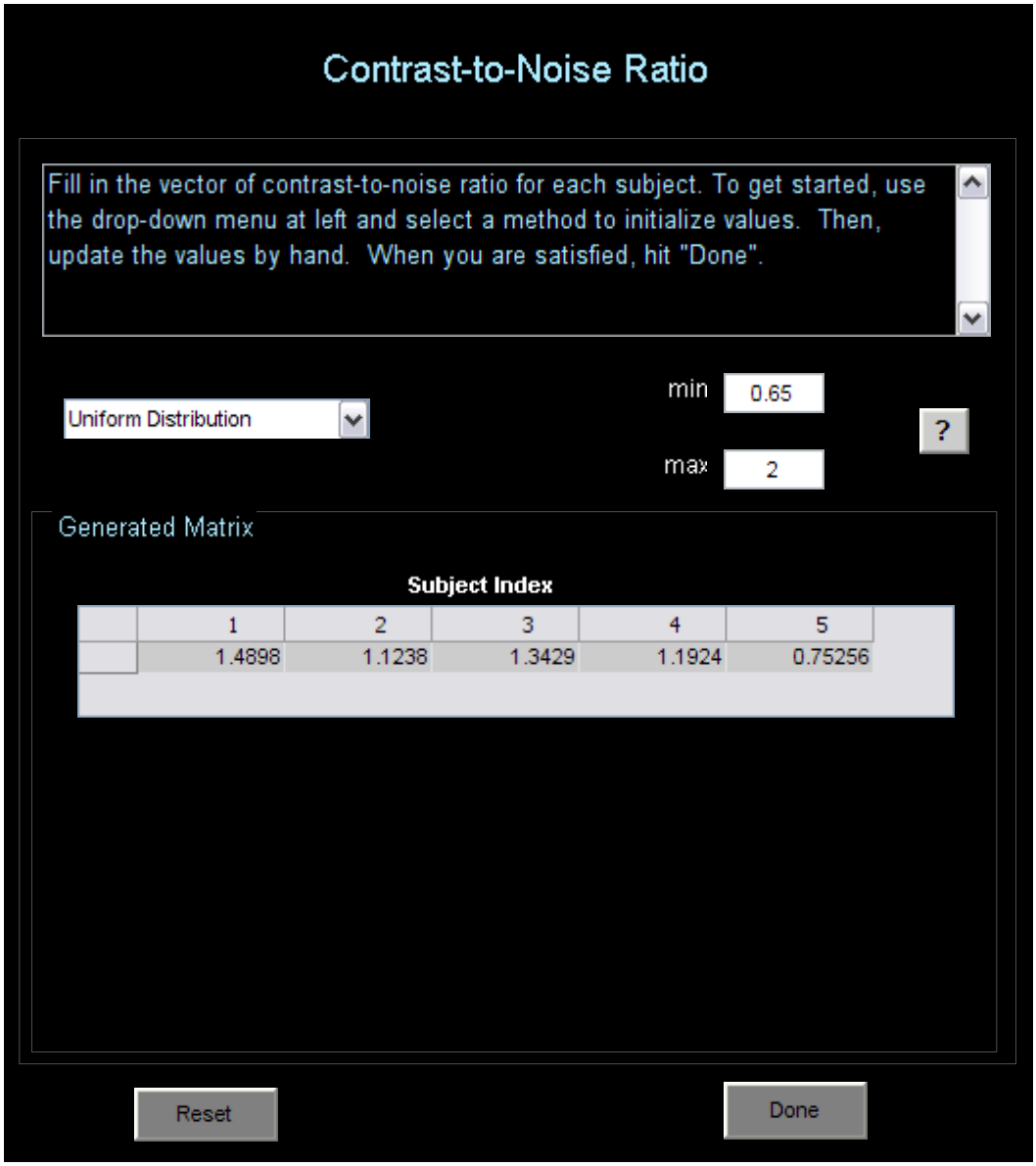
```
%% PEAK-TO-PEAK PERCENT SIGNAL CHANGE
%-------------------------------------------------------------------------
D_pSC = 3 + 0.25*randn(M,nC);    % [M x nC] matrix of percent signal changes
```

Additionally, in the parameter file we increase the CSF and decrease the WM component signal change. This will need to be done by hand in the GUI.

```
% To make statistical moments of data look more like real data
D_pSC(:,comp_CSF1) = 1.2*D_pSC(:,comp_CSF1);
D_pSC(:,comp_CSF2) = 1.2*D_pSC(:,comp_CSF2);
D_pSC(:,comp_WM1)  = 0.5*D_pSC(:,comp_WM1);
D_pSC(:,comp_WM2)  = 0.5*D_pSC(:,comp_WM2);
```

```
%-------------------------------------------------------------------------------
```

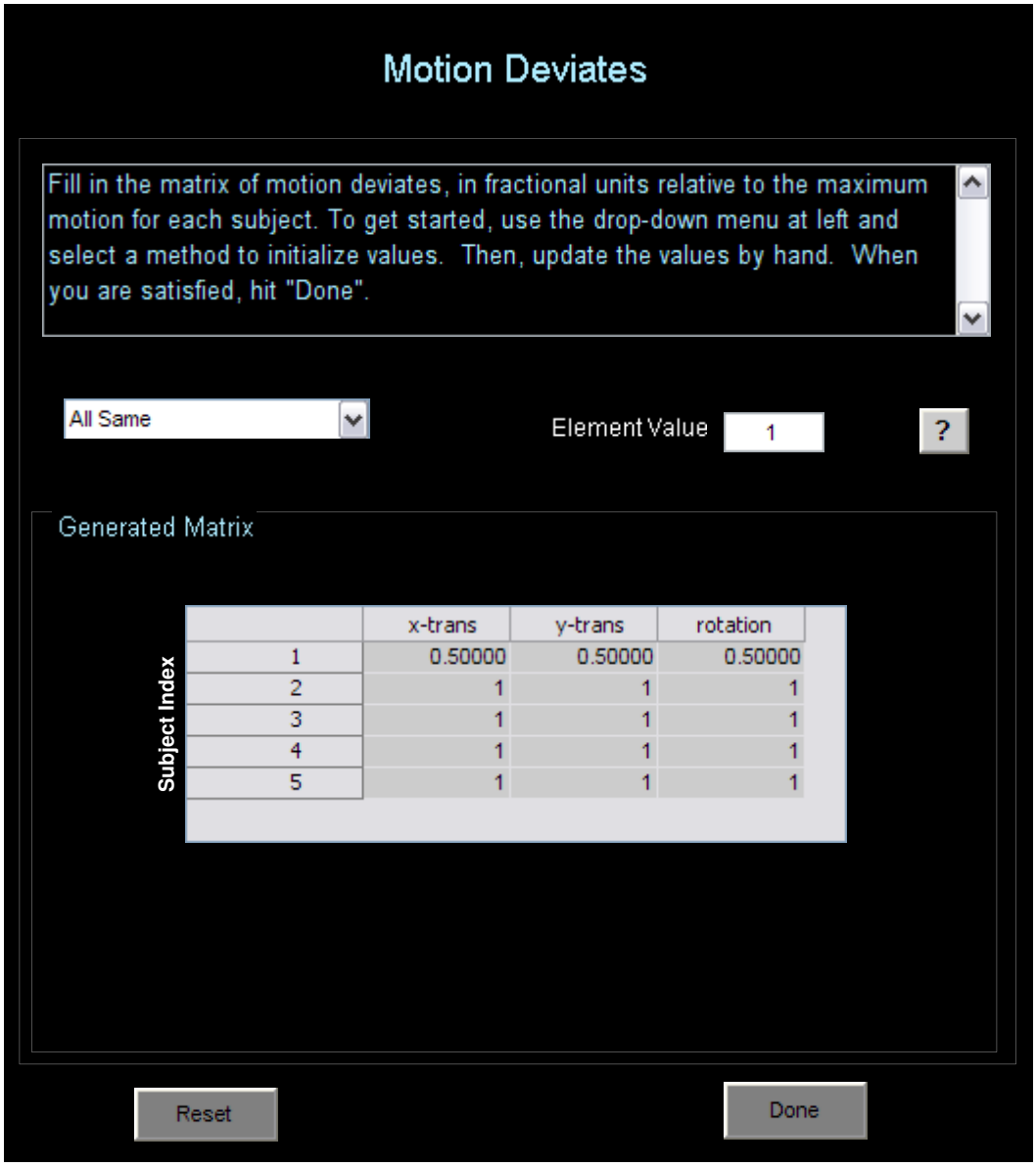
Step 5c: Contrast-to-noise ratio



The above GUI window can be equivalently coded as below from `experiment_params_aod.m`.

```
%% NOISE
%-------------------------------------------------------------------------------
D_noise_FLAG = 1;                % FLAG to add rician noise to the data
% [1 x M] vector of contrast-to-noise ratio for each subject
% CNR is distributed as uniform between 0.65 and 2.0 across subjects.
minCNR = 0.65;  maxCNR = 2;
D_CNR = rand(1,M)*(maxCNR-minCNR) + minCNR;
%-------------------------------------------------------------------------------
```

Step 5d: Motion deviates



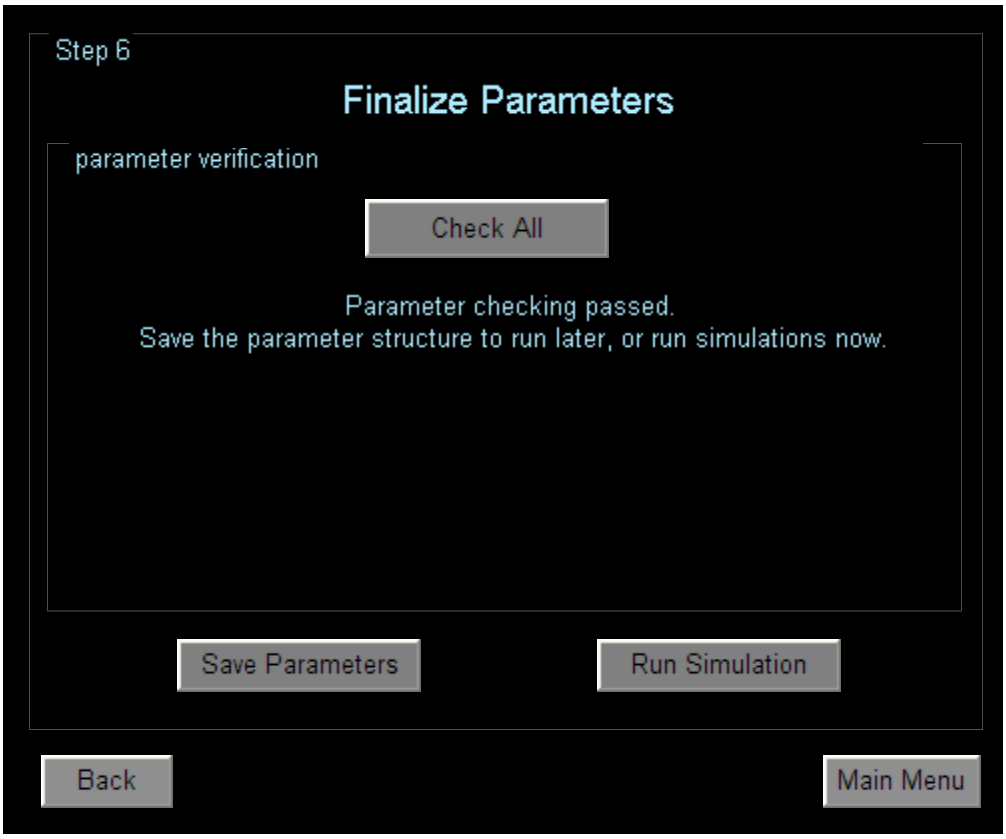
The above GUI window can be equivalently coded as below from `experiment_params_aod.m`.

Note: Max translation and rotation are in a previous panel.

```
%% MOTION
%-----------------------------------------------------------------------------
D_motion_FLAG = 1;                % 1=motion, 0=no motion
D_motion_TRANSmax = 0.02;         % max translation, proportion of entire image
D_motion_ROTmax = 5;              % max rotation, in degrees
D_motion_deviates = ones(M,3);    % proportion of max each subject moves
D_motion_deviates(1,:) = 0.5;     % Subject 1 moves half as much
%-----------------------------------------------------------------------------
% END of parameter definitions
```

Step 6: Finalize parameters

Check all verifies all parameters are set correctly. This check should always pass because selected parameters are checked upon exiting each panel.

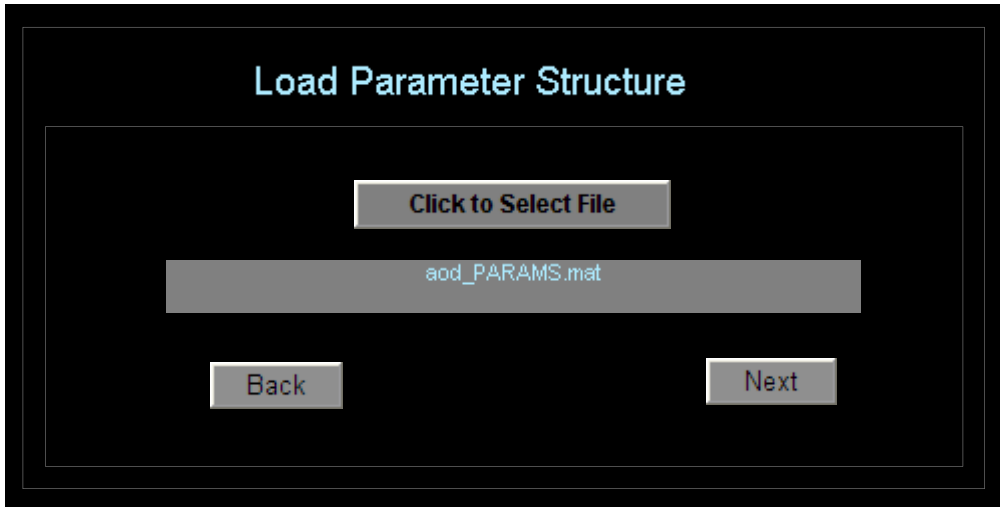


The above GUI window can be equivalently coded as below from `experiment_params_aod.m`.

```
% load parameter file
sP = simtb_create_sP('experiment_params_aod');
% check parameters for permissible values
[errorflag, Message] = simtb_checkparams(sP);
% display error messages, if any
disp(Message)
```

5.4 Run: Simulate Data

Run reads a parameter structure .mat file and runs the simulation. If `verbose_display=1`, then the following figures are produced in addition to the text output to the command window. The code to manually produce the figures (post hoc) is also provided.



The above GUI window can be equivalently coded as below from `experiment_params_aod.m`.

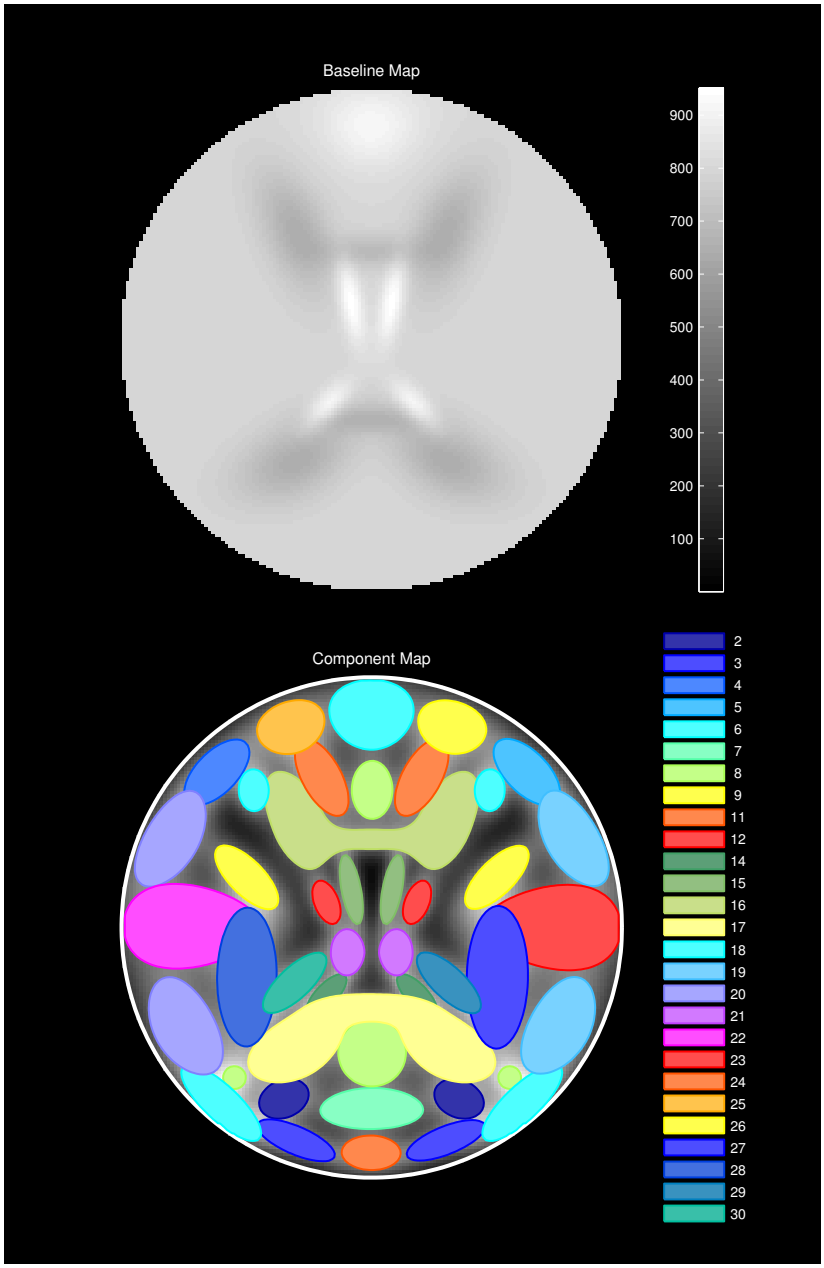
```
% Run the simulation
simtb_main(sP);
% save a MASK to make subsequent analyses easier
MASK = simtb_createmask(sP, 1);
```

```
Loading Parameters from 'experiment_params_aod'
Simulations will use approximately 80 MB of memory (per subject).

----------------------------------------------------------------------
---------------------Initializing Simulation-----------------------
----------------------------------------------------------------------

Output directory: X:\MyData\Simulations
     File prefix: aod
 Verbose display: ON

    Subject 1 of 5:
        Building Time Courses
        Building Spatial Maps
        Building Dataset
        Adding Motion
        Saving Motion Parameters
        Adding Noise
        Saving Simulated Components
        Saving Simulated Dataset
    Complete.
    Time to simulate subject 1: 18.0 s
```

```
    Subject 2 of 5:
        Building Time Courses
        Building Spatial Maps
        Building Dataset
        Adding Motion
        Saving Motion Parameters
        Adding Noise
        Saving Simulated Components
        Saving Simulated Dataset
    Complete.
    Time to simulate subject 2: 16.1 s

    Subject 3 of 5:
        Building Time Courses
        Building Spatial Maps
        Building Dataset
        Adding Motion
        Saving Motion Parameters
        Adding Noise
        Saving Simulated Components
        Saving Simulated Dataset
    Complete.
    Time to simulate subject 3: 16.1 s

    Subject 4 of 5:
        Building Time Courses
        Building Spatial Maps
        Building Dataset
        Adding Motion
        Saving Motion Parameters
        Adding Noise
        Saving Simulated Components
        Saving Simulated Dataset
    Complete.
    Time to simulate subject 4: 16.0 s

    Subject 5 of 5:
        Building Time Courses
        Building Spatial Maps
        Building Dataset
        Adding Motion
        Saving Motion Parameters
        Adding Noise
        Saving Simulated Components
        Saving Simulated Dataset
    Complete.
    Time to simulate subject 5: 16.6 s

Saving Parameter Structure
-------------------------------------------------------------------
------- Simulation Complete. Total Time: 1.4 minutes ---------------
-------------------------------------------------------------------
```

Simulation model for SMs

The tissue type baseline map is at top and the selected component map is at bottom.

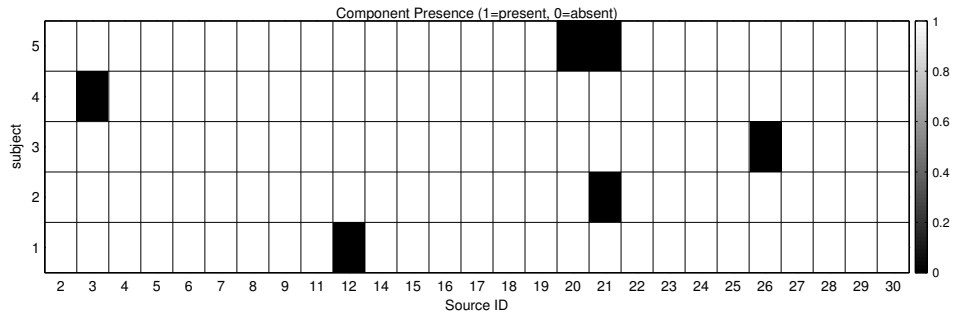



```
simtb_figure_model(sP); % both plots together as above
% simtb_figure_model(sP, 1, 0.5, 1); % Component Map
% simtb_figure_model(sP, 1, 0.5, 2); % Baseline Map
```

Parameters of the simulation

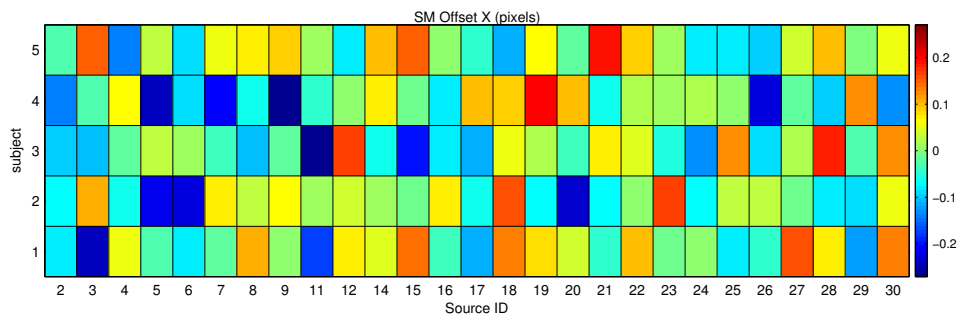
The following 12 figures can be produced with the single statement `simtb_figure_params(sP)`, and the code for individual figures are provided below each one.

Spatial source presence



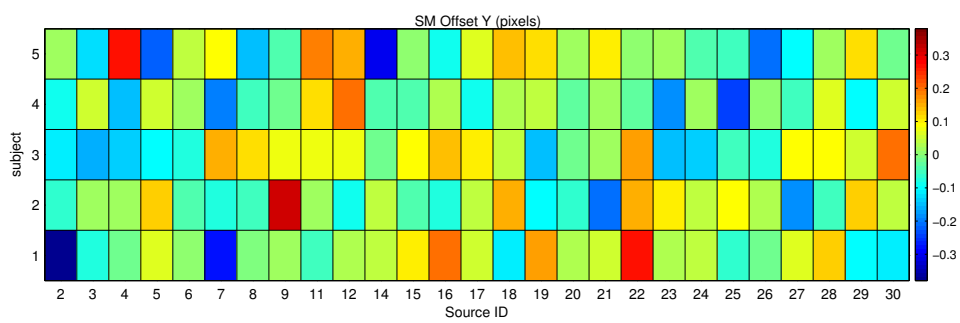

```
simtb_figure_params(sP, 'SM_present');
```

Spatial source horizontal translation



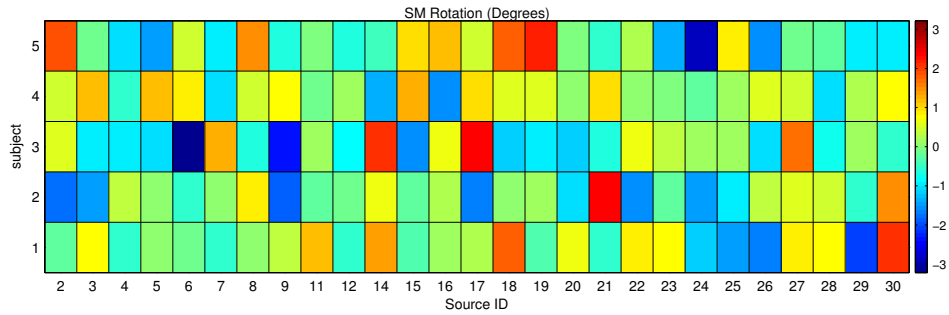

```
simtb_figure_params(sP, 'SM_translate_x');
```

Spatial source vertical translation



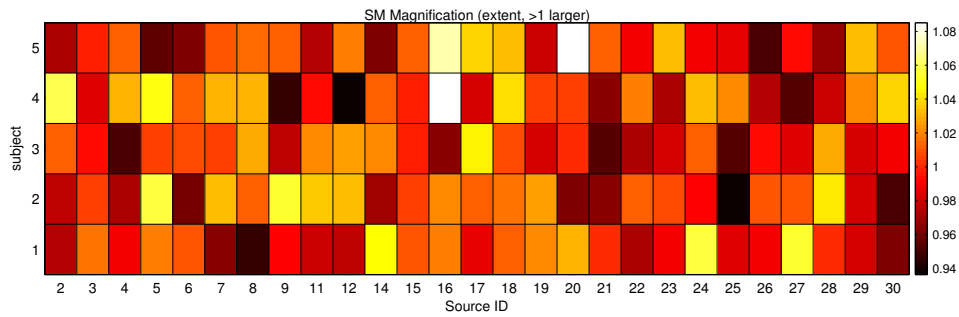

```
simtb_figure_params(sP, 'SM_translate_y');
```

Spatial source rotation



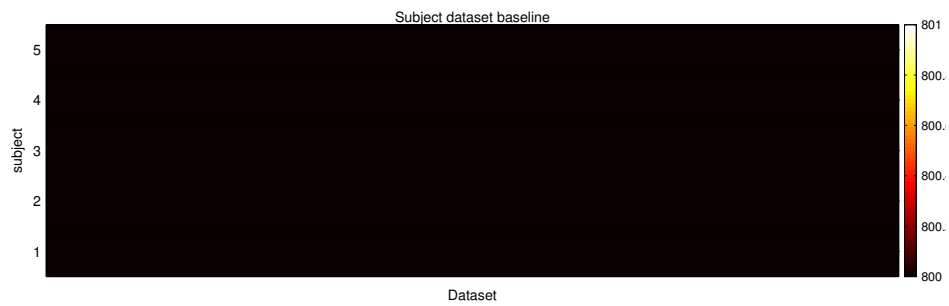

```
simtb_figure_params(sP, 'SM_theta');
```

Spatial source size/spread



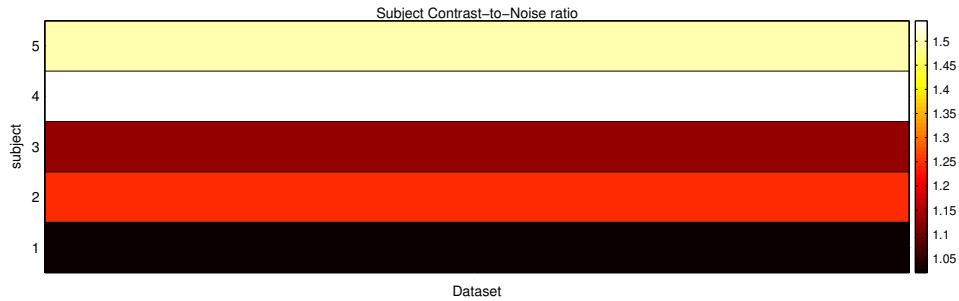

```
simtb_figure_params(sP, 'SM_spread');
```

Subject dataset baseline



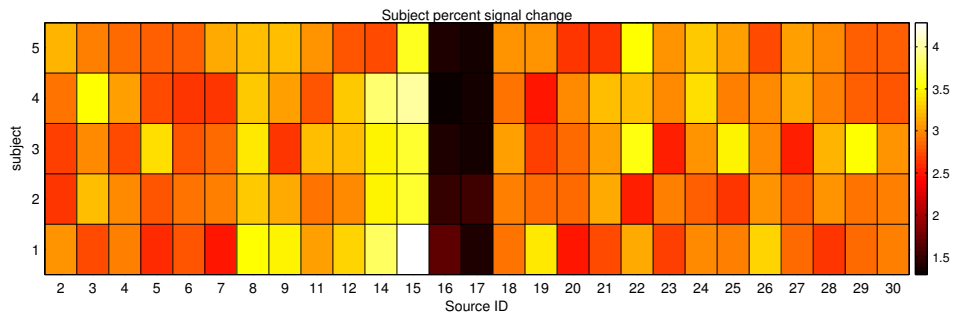

```
simtb_figure_params(sP, 'D_baseline');
```

Subject contrast-to-noise ratio (CNR)



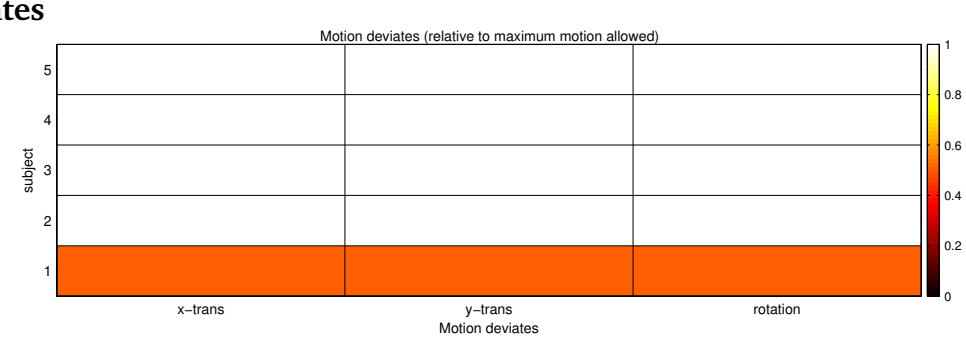

```
simtb_figure_params(sP, 'D_CNR');
```

Percent signal change




```
simtb_figure_params(sP, 'D_pSC');
```

Motion deviates



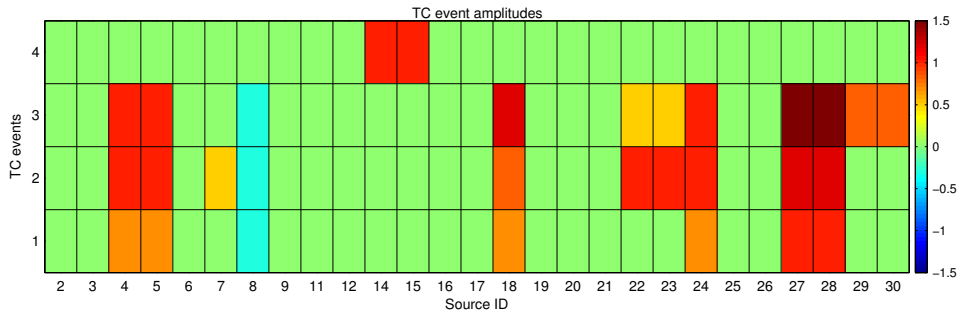

```
simtb_figure_params(sP, 'D_motion_deviates');
```

Task block amplitude

Note that there is no figure for block amplitude because blocks are not included in the simulation. When blocks are specified, this figure provides similar information as the plot for `TC_event_amp`.

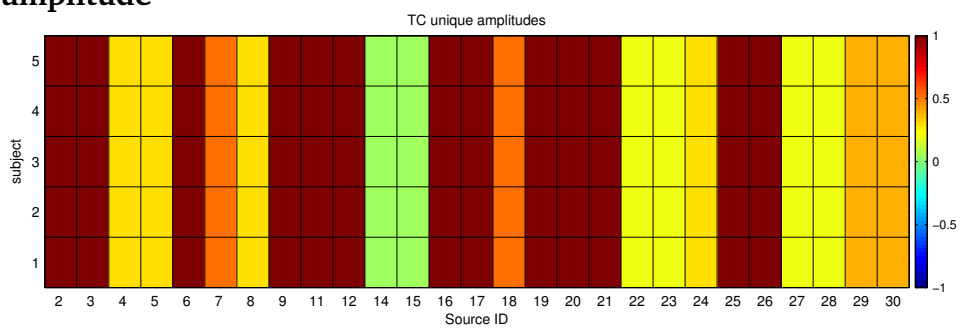
```
simtb_figure_params(sP, 'TC_block_amp');
```

Task event amplitude




```
simtb_figure_params(sP, 'TC_event_amp');
```

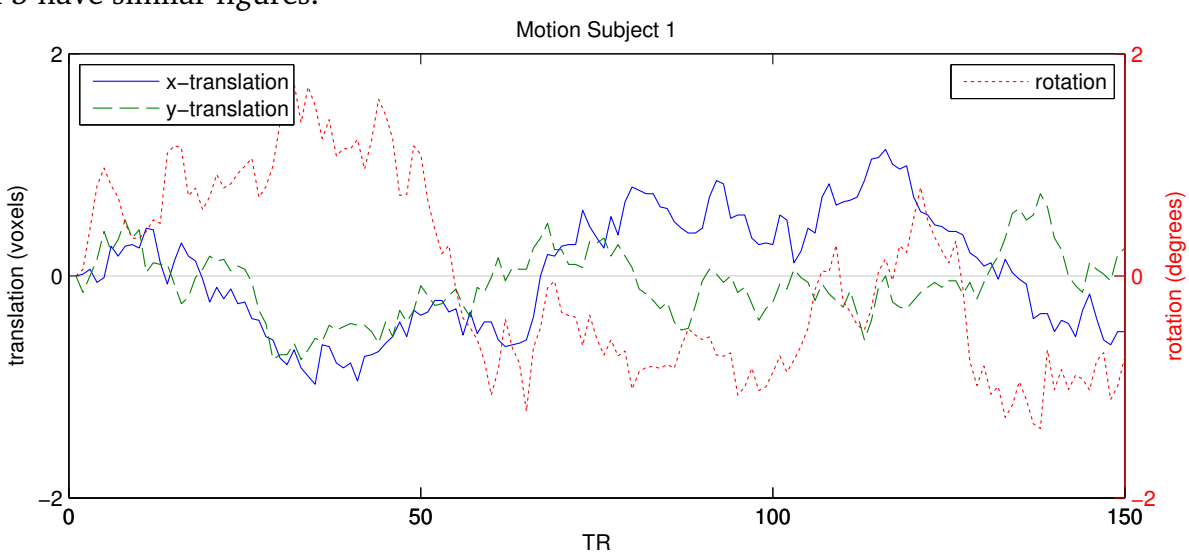
Unique event amplitude




```
simtb_figure_params(sP, 'TC_unique_amp');
```

Motion over time

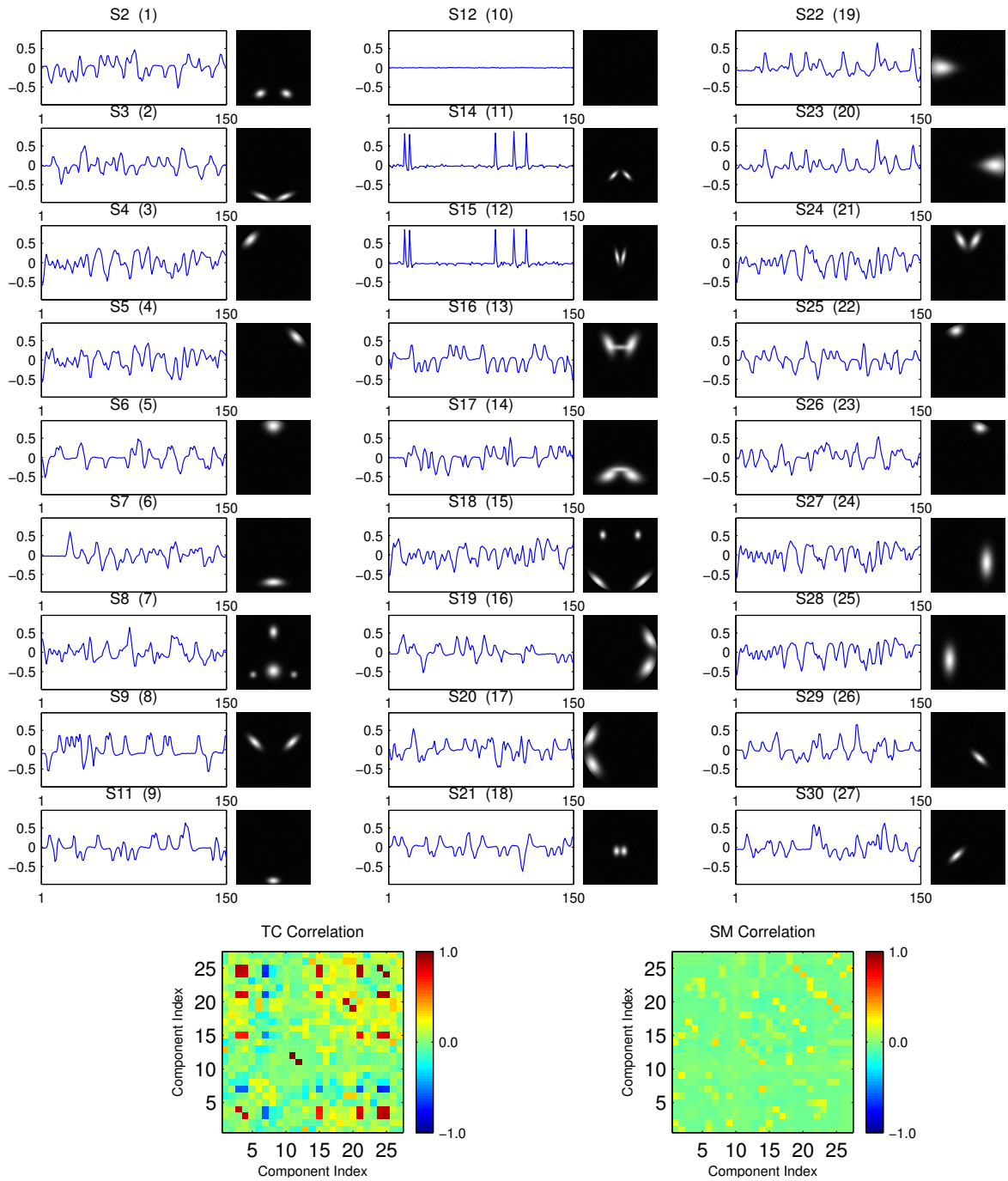
Motion parameters for subject 1 with translation on left axis, rotation on right axis. Subjects 2 through 5 have similar figures.




```
simtb_show_motion(sP, 1);
```

Subject Output

TCs and SMs for subject 1. Below are the TC and SM correlation matrices showing the similarity between components. Subjects 2 through 5 have similar figures.




```
simtb_figure_output(sP, 1);
```

6 How To...

In this section we provide a number of examples demonstrating how to customize simulations. Examples include relatively common scenarios, such as adding and testing spatial variability across subjects, as well as more complex modifications, such as developing new TC generation models. We have embedded copious snippets of MATLAB code to highlight the use of standard `SimTB` functions for generating, loading, and displaying simulation features.

6.1 Design an experiment

Component TCs can be modulated by experimental paradigms that include blocks and/or event-related designs. Here we demonstrate how to design experiments and visualize TC generation. For more information on simulation parameters related to experiments, users are referred to Section 4.

The following blocks of code are intended as command line operations, though could easily be included in a parameter file as in Section 5. In the first block, we will create a default parameter structure with specified number of subjects and components. Default parameters do not include an experimental paradigm (`TC_block_n = 0` and `TC_event_n = 0`). We will alter parameters to include a simple block experiment with a single condition and will assign task-modulation to particular components. The resulting TCs will then be displayed using the utility `simtb_showTC`.

```
% Create a parameter structure with default values
M = 1; nC = 4;                  % for simplicity, use 1 subject, 4 components
sP = simtb_create_sP([],M,nC);

% Set the parameters for a block experiment
sP.TC_block_n = 1;              % simple experiment with 1 condition
sP.TC_block_length = 20;        % On for 20 TRs
sP.TC_block_ISI = 15;           % Off for 15 TRs
sP.TC_block_amp(3,1) = 1;       % Component 3 is positively task modulated
sP.TC_block_amp(4,1) = -1;      % Component 4 is negatively task modulated

% Display the generation of TC for all components, single subject
compIND = 1:nC;
sub = 1;
TC = simtb_showTC(sP, compIND, sub);
```

Output from the above code is displayed in Figure 4. Note that all component TCs have unique events, while only sources 3 and 4 have additional activation (or de-activation) corresponding to the experiment blocks.

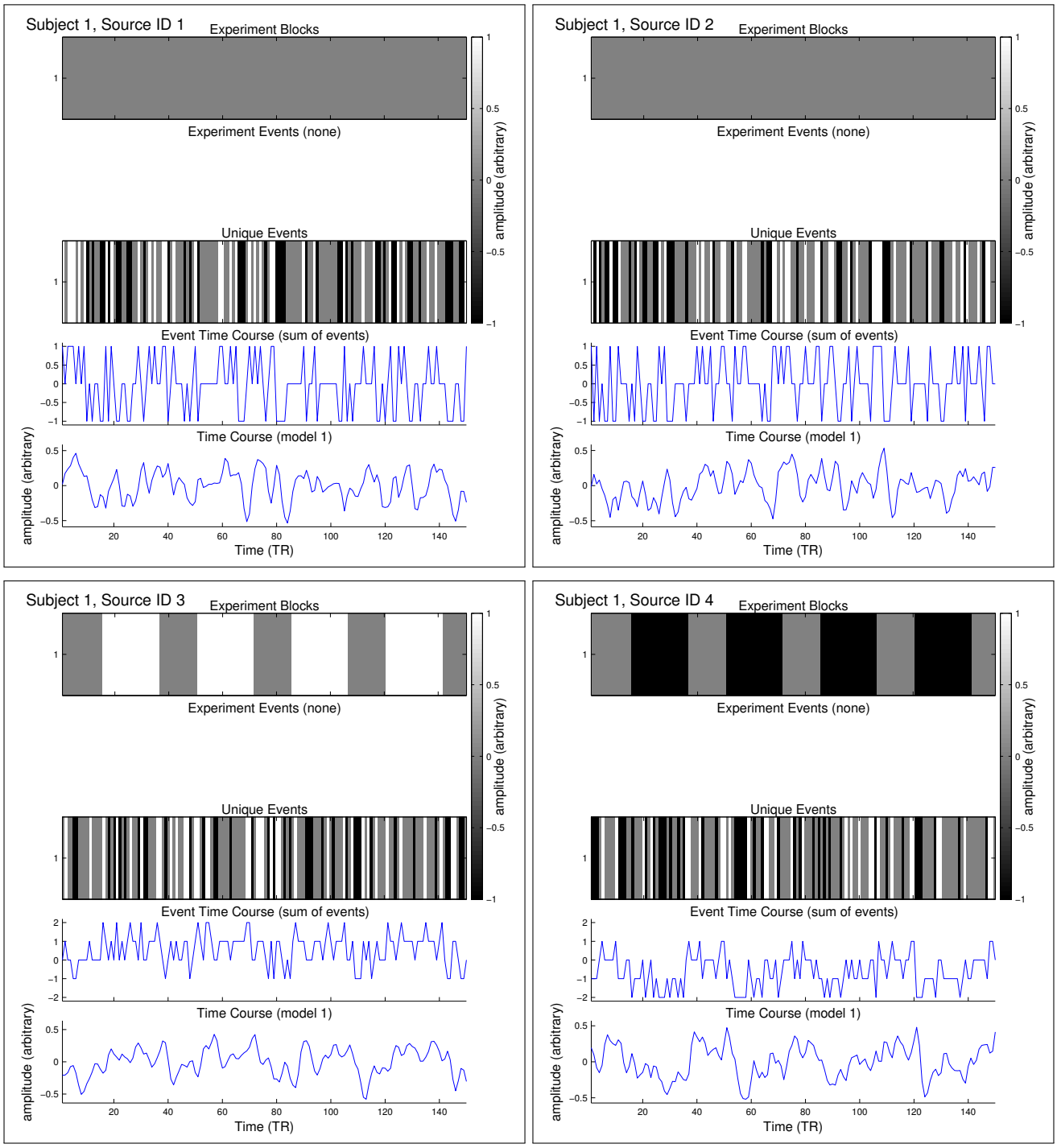


Figure 4: Standard output from `simtb_showTC` showing the generation of TCs for sources 1 through 4. The experimental paradigm is a simple block design with one condition. Sources 3 and 4 are task modulated while sources 1 and 2 have only unique events.

Though task modulation is visible in the component TCs in Figure 4, it is fairly weak. We may desire stronger task modulation which we can introduce by increasing the amplitudes of task blocks (`TC_block_amp`) relative to unique events (`TC_unique_amp`; by default, unique event amplitudes are 1 for all subjects and components). The following code updates the amplitudes of task blocks for components 3 and 4.

```
% Increase block amplitudes relative to unique event amplitude (1 by default)
sP.TC_block_amp(3,1) = 2;       % Component 3 is strongly positively task modulated
sP.TC_block_amp(4,1) = -2;      % Component 4 is strongly positively task modulated
% Alternatively, we could change the amplitude of unique events, i.e.,
% sP.TC_unique_amp(:,[3,4]) = 0.5;

% Display the generation of TC for just components 3 and 4
compIND = [3 4];
TC = simtb_showTC(sP, compIND, sub);
```

In Figure 5 we show the updated TCs. Task modulations are now more pronounced. Note that the timing for unique events and blocks is identical to those previously generated (Fig. 4) since we have not changed the simulation seed stored in `sP.seed`. To change the seed, one can call `simtb_rand_seed` which produces a new seed, e.g., `sP.seed = simtb_rand_seed;`. Calling `simtb_showTC` would now produce a different result (not shown).

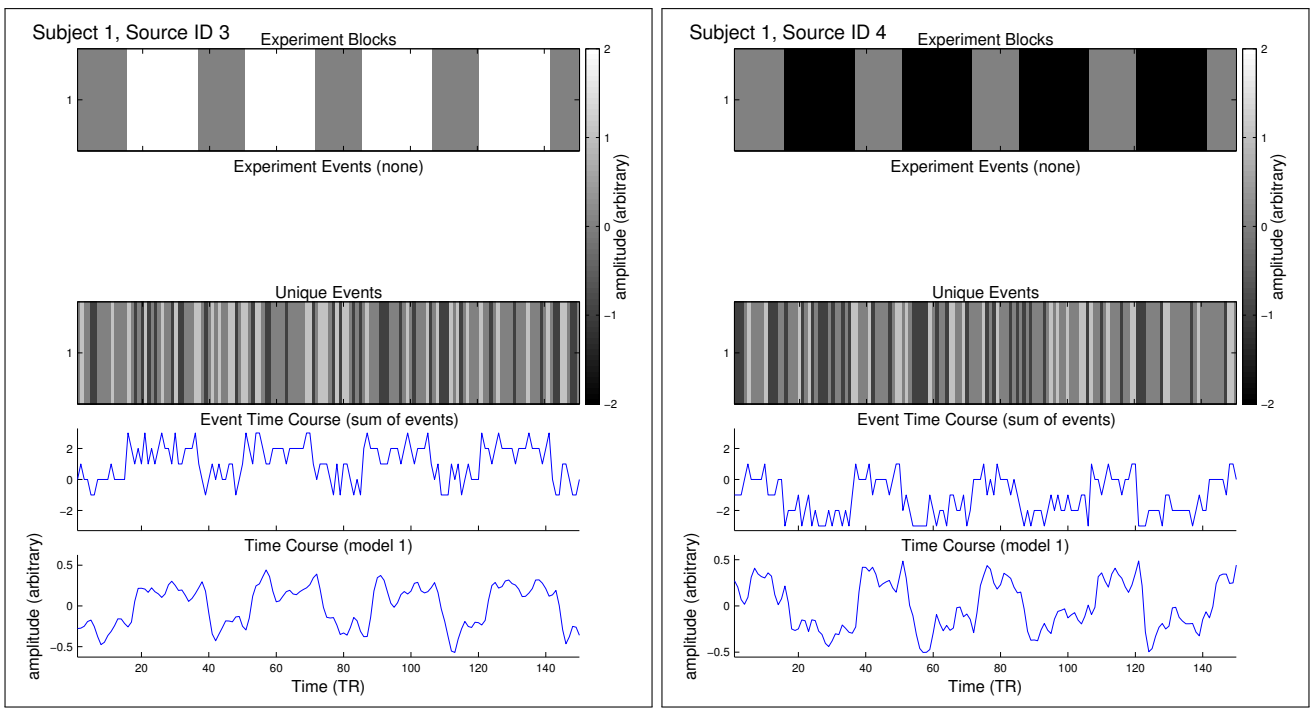


Figure 5: Output from `simtb_showTC` showing the generation of TCs for sources 3 and 4, with greater task modulation as compared to Figure 4.

We can also design a more complex experiment with multiple block conditions. Below, we set the number of conditions to 2 and increase the length of the experiment to accommodate more blocks. Block timing will be pseudo-random so that each condition is presented the same number of times (if possible given the number of time points). Output from the code below is provided in Figure 6.

```
sP.TC_block_n = 2;           % We now have 2 conditions
sP.nT = 260;                 % increase the length of the experiment
sP.TC_block_amp(3,1) = 2;    % Comp 3 is strongly modulated by condition 1
sP.TC_block_amp(3,2) = 0.5;  % Comp 3 is weakly modulated by condition 2
sP.TC_block_amp(4,1) = -1;   % Comp 4 is negatively modulated by condition 1
sP.TC_block_amp(4,2) = 1.5;  % Comp 4 is strongly modulated by condition 2

% Display the generation of TC for just components 3 and 4
compIND = [3 4];
TC = simtb_showTC(sP, compIND, sub);
```

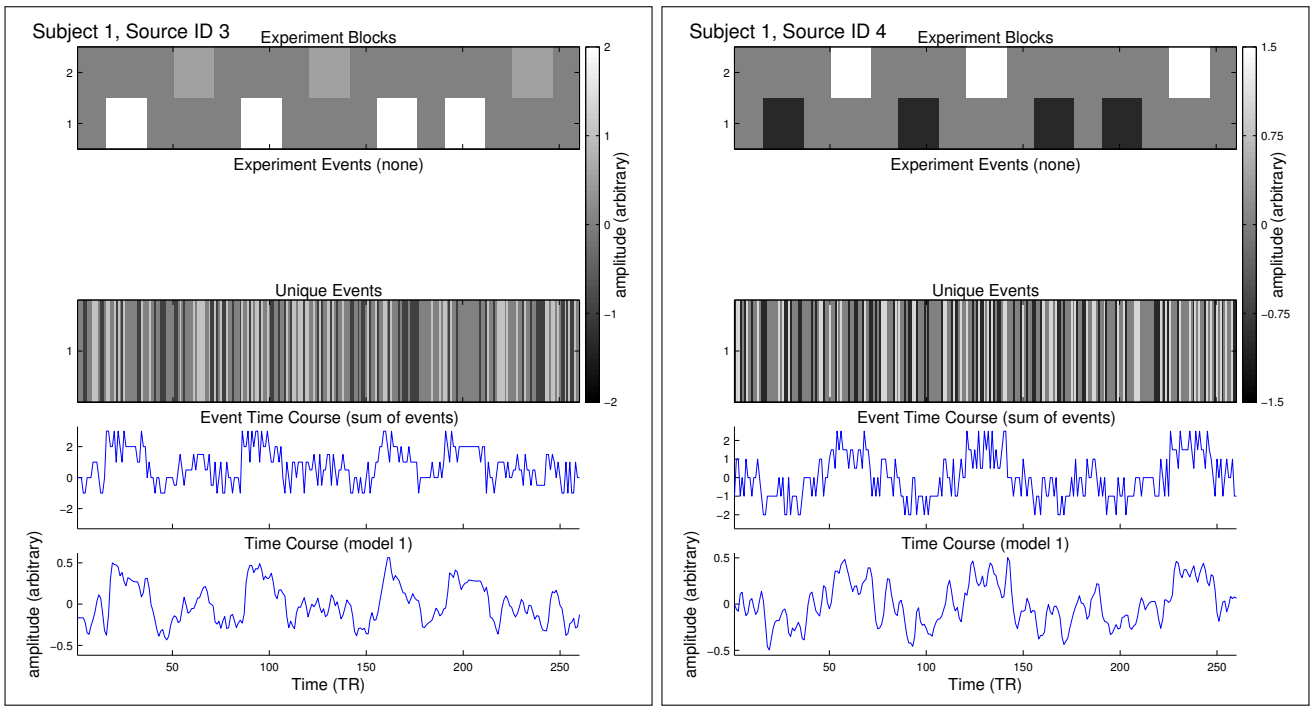


Figure 6: Output from `simtb_showTC` showing the generation of TCs for sources 3 and 4. The experimental paradigm is a block design with two conditions.

Instead of a block design, we may want an event-related design (though it is possible to include both blocks and experimental events). In the code below, we begin from the default parameter structure and add an even-related design with 3 trial types. Events are parameterized by a probability of occurrence at each TR (`TC_event_prob`). Note that the probability of any experimental event may not exceed 1, that is, the sum of `TC_event_prob` must be less or equal than 1. In this example the sum of experimental event probability is 0.5, thus events will occur at roughly half of the time points.

```
% Create a structure parameter with default values
% For simplicity, use 2 subjects, 4 components
M = 2; nC = 4;
sP = simtb_create_sP([],M,nC);

% Set the parameters for an event-related experiment
sP.TC_event_n = 3;              % experiment with 3 trial types
sP.TC_event_prob = [.3 .1 .1]; % event type 1 occurs more often than 2 and 3
sP.TC_event_same_FLAG = 1;      % experiment timing the same across subjects
sP.TC_event_amp(4,1) = 0.5;     % Comp 4 is weakly modulated by trial type 1
sP.TC_event_amp(4,2) = 2;       % Comp 4 is strongly modulated by trial type 2
sP.TC_event_amp(4,3) = -2;      % Comp 4 is strongly modulated by trial type 3

% Display the generation of TC for component 4, subjects 1 and 2
compIND = [4];
sub = [1 2];
TC = simtb_showTC(sP, compIND, sub);
```

Output from this code block is displayed in Figure 7. Note that experiment events are identical between subjects 1 and 2, while unique events are different. To introduce different event timing across subjects, we simply add the following line of code: `sP.TC_event_same_FLAG = 0;`. By changing this parameter and rerunning the `simtb_showTC` display function, we produce the output in Figure 8.

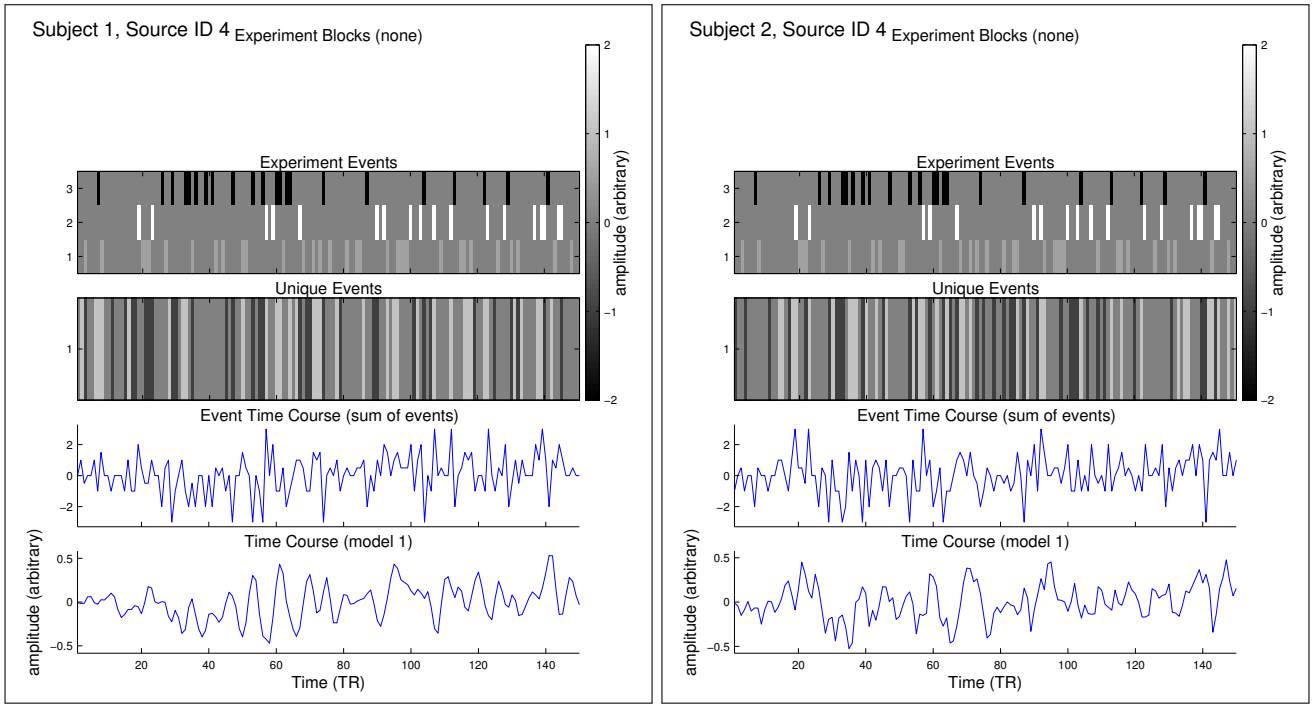


Figure 7: Output from `simtb_showTC` shows the generation of TCs for source 4 in an event-related paradigm with three trial types. Note that experiment events are identical between subjects 1 and 2, while unique events are different.

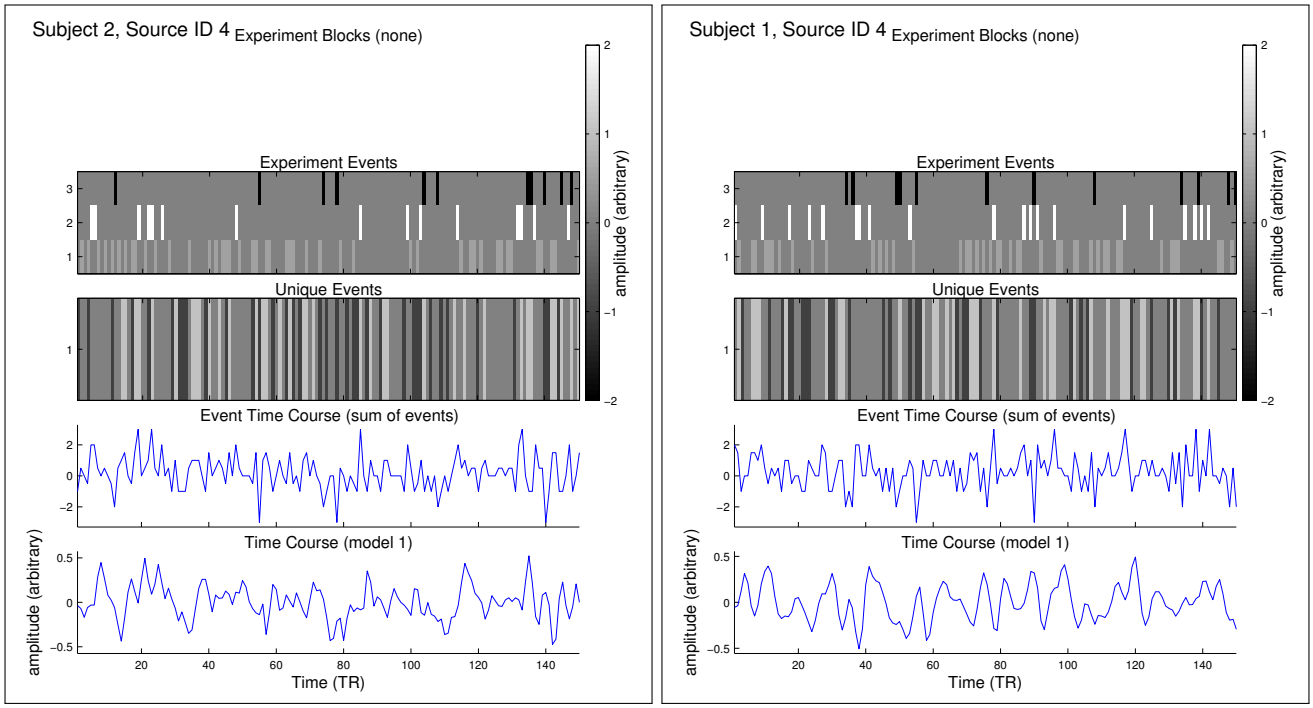


Figure 8: Output from `simtb_showTC` shows the generation of TCs for subjects 1 and 2 when `sP.TC_event_same_FLAG = 0;`. Compare TCs to those in Figure 7.

6.2 Introduce spatial variability

In general, spatial variability between subjects is implemented by varying simulation parameters across subjects. There are four parameters that affect spatial properties of components and can be modified for individual subjects. These are `SM_translate_x`, `SM_translate_y`, `SM_theta`, and `SM_spread`. The following example demonstrates how to alter these parameters, with emphasis on testing and visualizing modifications. Below, we create a default parameter structure (in which component parameters are identical across subjects) and plot the SMs for subject 1.

```
sP = simtb_create_sP;        % Create default parameter structure
                             % Will have 10 subjects, 30 components
sub = 1;
SM = simtb_makeSM(sP, sub); % Make the SMs for subject 1.
simtb_showSM(SM);            % Plot the SMs
```

The output is displayed in Figure 9. Note that the default model includes Source 1, which represents a global mean component that spans the head uniformly.

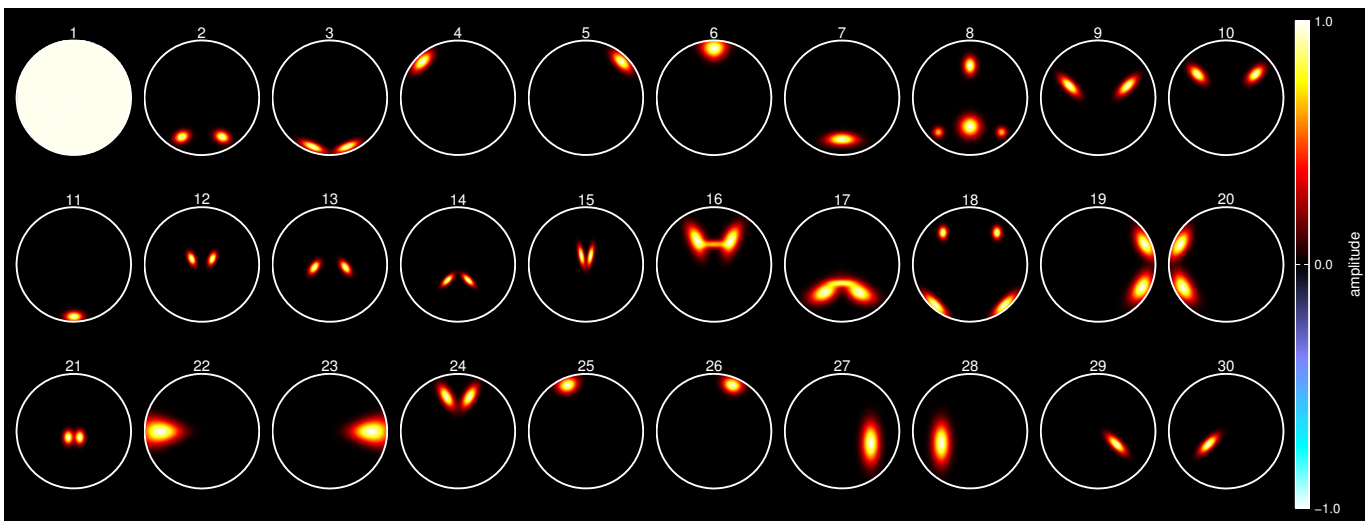


Figure 9: The default 30 SMs for a single subject, plotted using `simtb_showSM`.

Now we introduce spatial variability across subjects. For clarity, each parameter is altered for a single component and parameter deviations are spaced linearly and systematically across subjects. Note that normally distributed deviates (as implemented in Section 5) would be more similar to what is found in real data.

```
% Source 21: offset in y position, linearly spaced between -5 to +5 voxels
compIND = find(sP.SM_source_ID == 21); % get the component index for the source
sP.SM_translate_y(:, compIND) = linspace(-5, 5, sP.M)';

% Source 27: offset in x position, linearly spaced between -6 to +6 voxels
compIND = find(sP.SM_source_ID == 27);
sP.SM_translate_x(:, compIND) = linspace(-6, 6, sP.M)';

% Source 28: rotation, linearly spaced between -30 to +30 degrees
compIND = find(sP.SM_source_ID == 28);
sP.SM_theta(:, compIND) = linspace(-30, 30, sP.M)';

% Source 7: variability in spread (size), linearly spaced between 0.5 and 3
compIND = find(sP.SM_source_ID == 7);
sP.SM_spread(:, compIND) = linspace(0.5, 3, sP.M)';
```

When updating parameters, it's always a good idea to use `simtb_figure_params` to visually confirm that modifications are as we intend. The code below calls `simtb_figure_params` for the parameters of interest. Corresponding output is displayed in Figure 10.

```
% Check to make sure that the parameters have been updated correctly
simtb_figure_params(sP,{'SM_translate_y','SM_translate_x','SM_theta','SM_spread'});
```

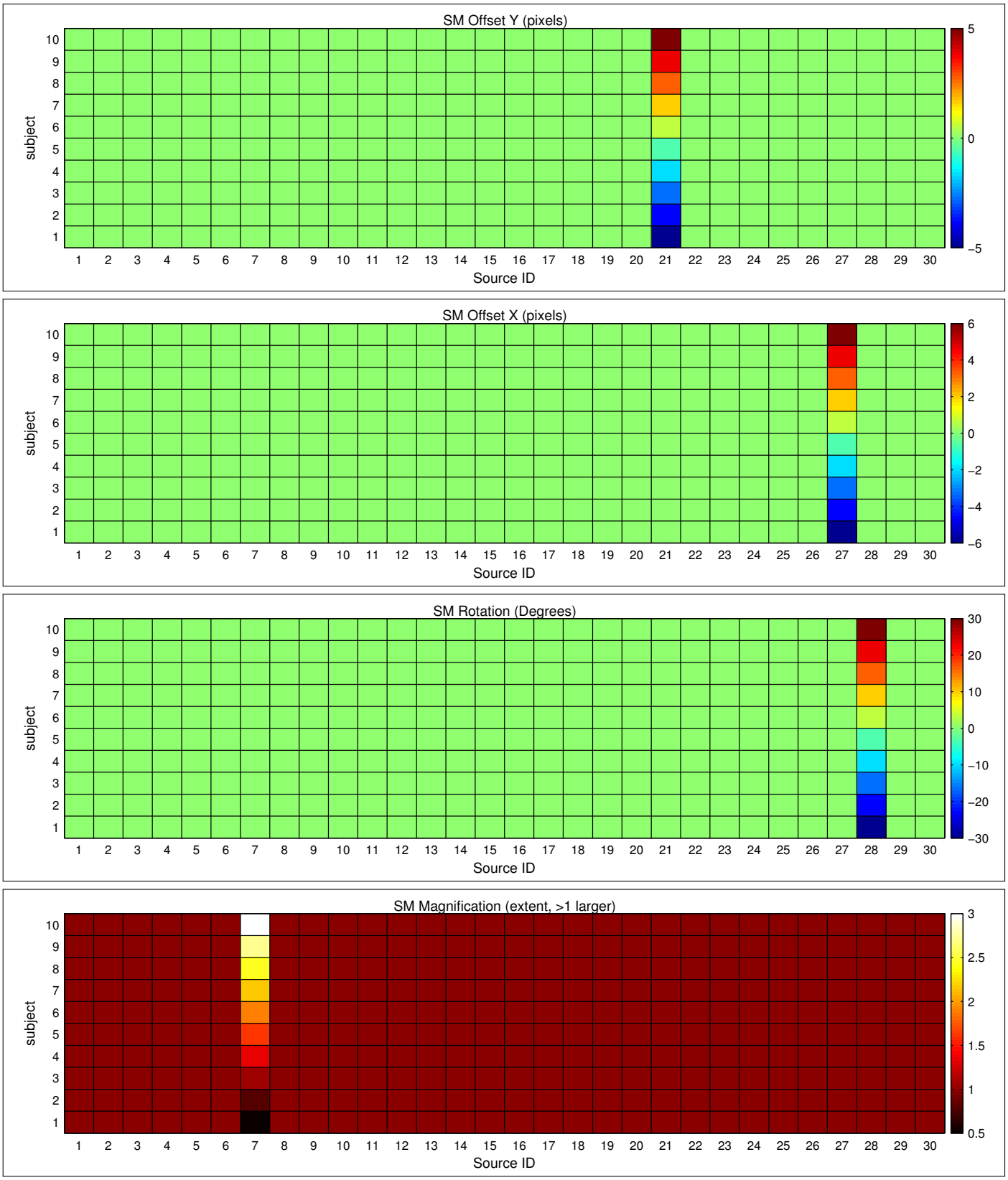


Figure 10: Output from `simtb_figure_params` showing the values for `SM_translate_y`, `SM_translate_x`, `SM_theta`, and `SM_spread`.

To assess whether the degree of spatial variability is appropriate, we can examine the actual SMs. In the following code we first generate SMs for each subject using `simtb_makeSM`, then loop over the sources of interest and plot them using `simtb_showSM`. Results are displayed in Figure 11.

```
% Generate SMs for all subjects
allSM = zeros(sP.M, sP.nC, sP.nV*sP.nV); % initialize matrix to hold all SMs
for sub = 1:sP.M                         % loop over subjects
    SMsub = simtb_makeSM(sP, sub);       % make the SMs for a subject
    allSM(sub,:,:) = SMsub;              % store in larger matrix
end

% Plot the SMs
sources_of_interest = [21 27 28 7];
for c = sources_of_interest                % loop over the sources of interest
    compIND = find(sP.SM_source_ID == c); % get the component index
    SM_c = squeeze(allSM(:,compIND,:));   % SMs for all subjects, single component
    simtb_showSM(SM_c);                   % plot the SMs
end
```

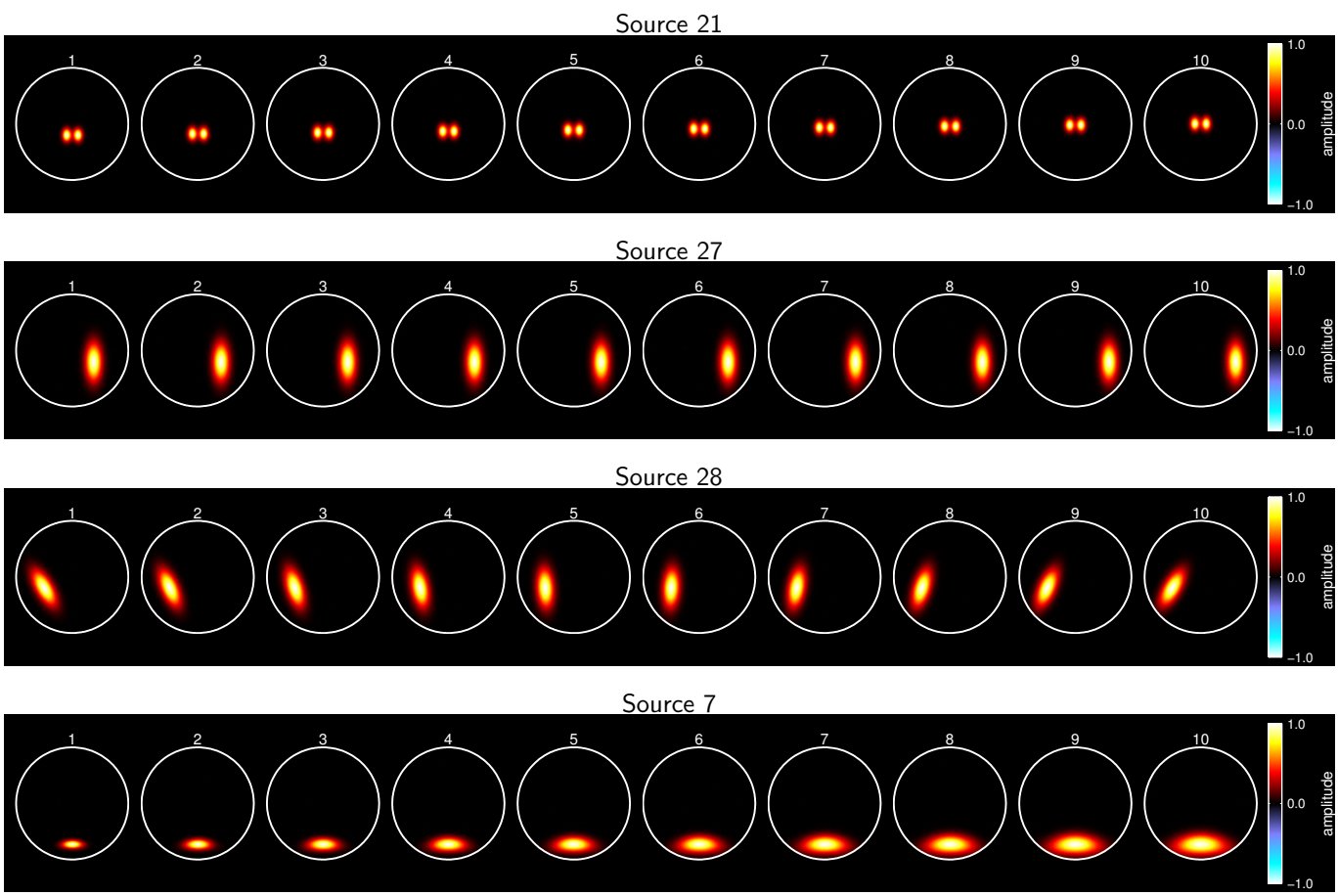


Figure 11: SMs of Sources 21, 27, 28, and 7 for subjects 1-10 (left to right). Images were produced using `simtb_showSM`.

In Figure 11, spatial variability in rotation (`SM_theta`) and size (`SM_spread`) is obvious. Variability in component translation is a little more difficult to see since the components are all plotted on different axes. To put them on the same axes we employ another plotting utility, `simtb_showSMContours`. Use of this function is demonstrated below; further details and options regarding its use can be found by typing `help simtb_showSMContours`. Figure 12 displays the output.

```
% Define preferences for simtb_showSMContours
contour_level = 0.5;      % draw contours at half the maximum value
FILL_flag = 0;            % do not fill contours
cmap = jet(sP.M+2);       % use different colors for each subject
cmap = cmap(3:sP.M+2,:); % skip first colors to avoid dark blue

% Plot the SMs
for c = sources_of_interest                % loop of the sources of interest
    compIND = find(sP.SM_source_ID == c); % get the component index
    SM_c = squeeze(allSM(:,compIND,:));   % SMs for all subjects, single component
    simtb_showSMContours(SM_c, contour_level, cmap, FILL_flag);
end
```

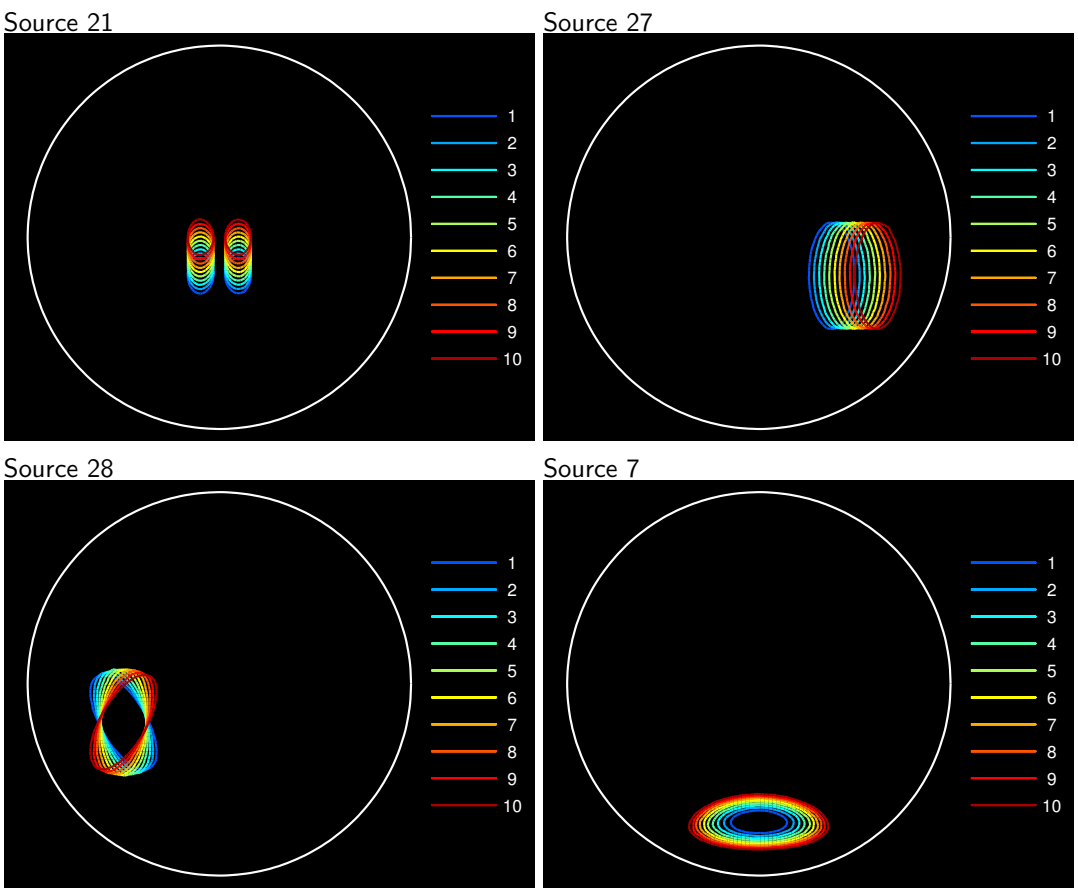


Figure 12: Contour plots of Sources 21, 27, 28, and 7 for subjects 1 (blue) to 30 (red). Images were produced using `simtb_showSMContours`.

6.3 Modify TC source parameters

In the default settings, SimTB allows TC source parameters (i.e., hemodynamic constants) to vary between components and subjects to better approximate real data. We demonstrate the effects of parameter variability by repeatedly generating a TC from a simple event time series. In the code below, we perform this procedure for TC model 1, which convolves the event time series with a canonical HRF (defined as the difference between two gamma functions). The results are displayed in Figure 13.

```
% Make a very simple event time series (single event)
TR = 1;                    % set repetition time = 1 second
nT = 35;                   % number of TRs
eTS = zeros(1,nT);         % event time series (no events yet)
eventIND = 3;              % single event at third TR
eTS(eventIND) = 1;         % make event

% Set the number of iterations and source model
nReps = 30;                % 30 iterations
allTC = zeros(nT,nReps);   % initialize matrix for TCs
sourceType = 1;            % Use TC Model 1 (canonical HRF)

% For each iteration generate a TC using a set of unique parameters
for ii = 1:nReps
    TC_ii = simtb_TCsource(eTS, TR, sourceType);
    allTC(:,ii) = TC_ii;
end

% Plot all 30 TCs
F = figure; set(F, 'Color', 'w');
plot([1:nT]-eventIND, allTC); xlabel('TR'); ylabel('amplitude')
axis tight; box off; set(gca, 'TickDir', 'out')
```

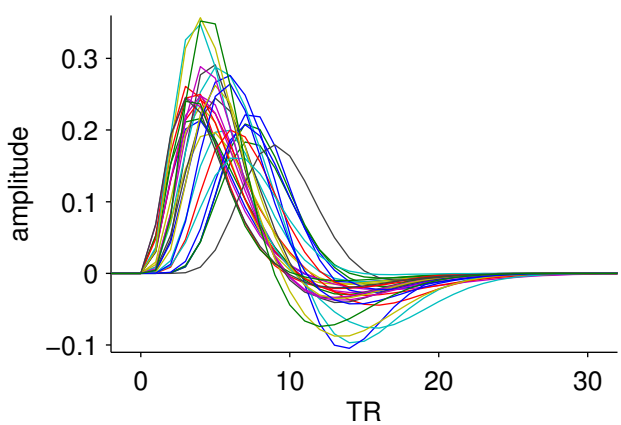


Figure 13: Variability in TCs generated with 30 repetitions of TC model 1 for a single event.

While hemodynamic variability is sometimes desired, there may be a number of reasons for users to force identical parameters or to designate particular hemodynamic constants to individual subjects or components. Here we provide several examples to give users full control over TC generation. All code blocks represent possible excerpts from a parameter file where the basic simulation parameters (e.g., number of components `nC`, number of subjects `M`, number of voxels `nV`) have been defined. Note that the `SimTB` GUI also includes options to assign identical parameters to all components in a single subject, and to assign identical parameters to all subjects (see Section 5, Step 4b). For more customized TC source parameters, GUI users may update the structure field `TC_source_params` as demonstrated below. TC source parameters are stored in the M-by-nC cell array, `TC_source_params`. To use default settings where parameters may vary between subjects and components, simply initialize `TC_source_params` to any empty array.

```
TC_source_params = cell(M,nC); % Use the default params
```

One can use the MATLAB function `deal` to assign identical parameters to multiple components and subjects. In the example below, we assume that all components use TC source model Type 1.

```
% Set the model parameters for TC model 1
P(1) = 6;       % delay of response (relative to onset)
P(2) = 16;      % delay of undershoot (relative to onset)
P(3) = 1;       % dispersion of response
P(4) = 1;       % dispersion of undershoot
P(5) = 6;       % ratio of response to undershoot
P(6) = 0;       % onset (seconds)
P(7) = 32;      % length of kernel (seconds)

% Assign identical parameters to all subjects and components
[TC_source_params{:}] = deal(P);
```

If several TC models are used, one must index each model separately. In this example we separately specify parameters for components with TC models Type 1 and Type 2. Also, instead of defining the model parameters ourselves we will use `simtb_TCsource` to generate a random set of parameters.

```
% Generate a random set of parameters for TC model 1
sourceType = 1; % canonical HRF
[tc_dummy, MDESC, P1, PDESC] = simtb_TCsource(1, 1, sourceType);

% Generate a random set of parameters for TC model 2
sourceType = 2; % balloon model
[tc_dummy, MDESC, P2, PDESC] = simtb_TCsource(1, 1, sourceType);

% Find the component indices for each model type
M1ind = find(TC_source_type == 1);
M2ind = find(TC_source_type == 2);

% Assign identical parameters for Model 1 to all subjects
[TC_source_params{:,M1ind}] = deal(P1);
% Assign identical parameters for Model 2 to all subjects
[TC_source_params{:,M2ind}] = deal(P2);
```

Finally, one may want to model components or subjects with particular parameters. In the code below, we assign each subject their own set of hemodynamic constants and specifically adjust the onset time for Source 24 so that it is delayed with respect to other components.

```
S24ind = find(SM_source_ID == 24);  % index for Source 24
M1ind  = find(TC_source_type == 1); % component indices with model 1

for sub = 1:M % loop through subjects
    sourceType = 1; % canonical HRF
    % Use simtb_TCsource to generate a random set of parameters
    [tc_dummy, MDESC, P, PDESC] = simtb_TCsource(1, 1, sourceType);

    % assign parameters to all type 1 components for this subject
    [TC_source_params{sub, M1ind}] = deal(P);

    % change the onset to make it delayed
    P(6) = P(6) + 2;
    % assign modified parameters to source 24
    TC_source_params{sub, S24ind} = P;
end
```

6.4 Create a new SM source

The toolbox includes 30 sources with spatial configurations displayed in Figure 2. Users can modify existing sources or add new SMs by editing the source definitions in `simtb_SMsource`. To demonstrate source modification we will alter Source 8, which is modeled after the default mode network (DMN). Source 8 has four distinct activation blobs, roughly corresponding to the anterior cingulate cortex (ACC), posterior cingulate cortex (PCC), and left and right angular gyri. Using the code below, we will generate and plot the default SM for Source 8. The resulting image is displayed in Figure 14.

```
% Use simtb_generateSM to create the SM with default values
sourceID = 8;
nV = 256;
[SM,TT] = simtb_generateSM(sourceID, nV);

% Plot the default SM using simtb_showSM
figure_handle = simtb_showSM(SM, [], 'Original_Source_8', [], 0);
```

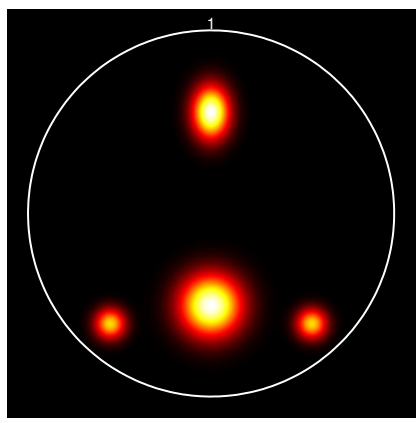


Figure 14: The original Source 8, created using `simtb_showSM`.

Source 8 is defined in Lines 107 through 119 of `simtb_SMsource`:

```
107 elseif sourceID == 8
108     %% Default Mode Network
109     theta = 0;
110     % ACC
111     z1 = make_blob(x,y,  0 + randx, .55 + randy, 10, 7, theta + randrot);
112     % PCC
113     z2 = make_blob(x,y,  0 + randx, -.5 + randy, 6, 6, theta + randrot);
114     % angular gyrus (right)
115     z3 = make_blob(x,y, 0.55 + randx, -.6 + randy, 12, 12, theta + randrot);
116     % angular gyrus (left)
117     z4 = make_blob(x,y, -0.55 + randx, -.6 + randy, 12, 12, theta + randrot);
118     z = z1 + z2 + 0.7*z3 + 0.7*z4;
119     TT = 3;
```

Each activation blob is defined in a separate line using the sub-function `make_blob`, which creates a 2-D Gaussian at a specified location and size, as explained in Section 2.1. Individual activations may be weighted (as in this case where the angular gyri `z3` and `z4` are scaled by 0.7) and are summed together to form the SM. We will modify Source 8 by increasing the size and weights of the angular gyri relative to the other activations. These modications are implemented in the lines of code below, and the outcome is displayed in Figure 15.

```
113     % angular gyrus (right), enlarged by decreasing width parameter from 12 to 8
114     z3 = make_blob(x,y, 0.55 + randx, -.6 + randy, 8, 8, theta + randrot);
115     % angular gyrus (left), enlarged by decreasing width parameter from 12 to 8
116     z4 = make_blob(x,y, -0.55 + randx, -.6 + randy, 8, 8, theta + randrot);
117     z = z1 + z2 + 1.0*z3 + 1.0*z4; % z3 and z4 are no longer scaled by 0.7
```

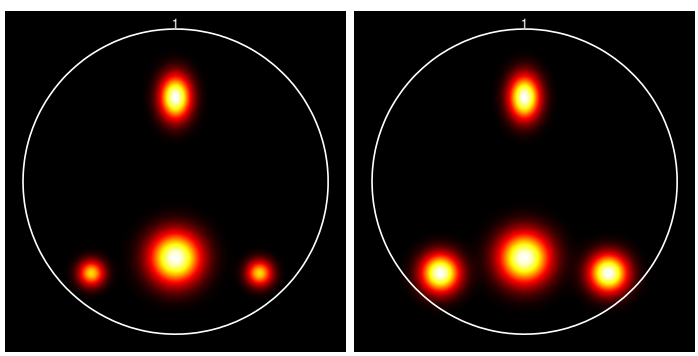


Figure 15: The original Source 8 (left) and modification (right).

We also may decide to translate one of the activation blobs, for example moving the ACC to more anterior position. This can be done by changing a single line of `simtb_SMsource` (below), the outcome of which is displayed in Figure 16.

```
    % ACC, translated to be more anterior (range, domain are [-1,1])
    z1 = make_blob(x,y,  0 + randx, .80 + randy, 10, 7, theta + randrot);
```

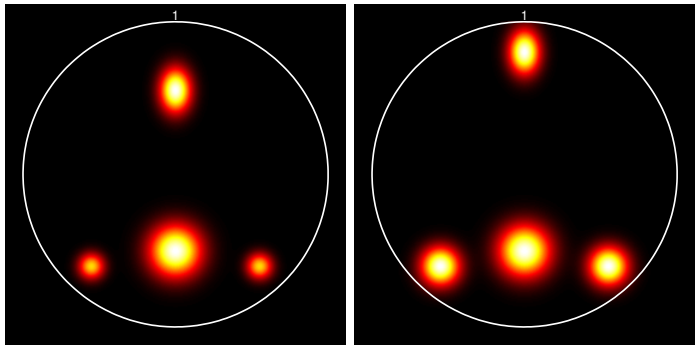


Figure 16: The original Source 8 (left) and modification, including enlargement of left and right angular gyri and anterior translation of the ACC (right).

Note that when modifying or adding components, it may be important to view all sources together to assess component overlap. This can be done using the helper function `simtb_figure_drawSMs`:

```
figure_handle = figure; set(figure_handle, 'Color', 'k') % Create a black figure
H = axes; set(H, 'Position', [0.05 0.05 0.9 0.9]);       % Create empty axes
simtb_figure_drawSMs(H, nV);                             % Draw the SM contours
```

As seen in Figure 17, there is significant overlap between anterior sources. Depending on the purpose and parameters of the simulation, overlap may or may not be problematic. Regardless, it is always a good idea to check the source configuration following modifications.

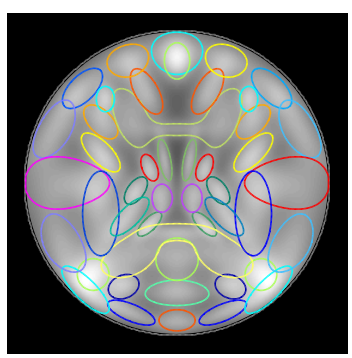


Figure 17: Output from `simtb_figure_drawSMs` shows contours of all sources, following modifications to Source 8.

To create a new source, one simply adds a definition to the bottom of the source list. In `simtb_SMsource`, we define Source 31 by adding another conditional statement with `elseif`. When adding a new source, one must define a `[nV x nV]` image of activations (`z`) as well as a tissue type (`TT`). Initially, we will model a circular blob ($w_x = w_y = 6$) of gray matter (TT = 3), centered at the origin ($x_0 = y_0 = 0$). The source will become progressively more complex in subsequent steps.

```
elseif sourceID == 31
    %% New Source: gray matter blob at the origin
    theta = 0;
    z = make_blob(x,y, 0 + randx, 0 + randy, 6, 6, theta + randrot);
    TT = 3;
```

To display our new source, we again use `simtb_generateSM` and `simtb_showSM`. The resulting image is displayed in the first panel of Figure 18.

```
sourceID = 31;
nV = 256;
[SM,TT] = simtb_generateSM(sourceID, nV);                    % Create the SM
figure_handle = simtb_showSM(SM, [], 'New_Source_v1', [], 0); % Plot the SM
```

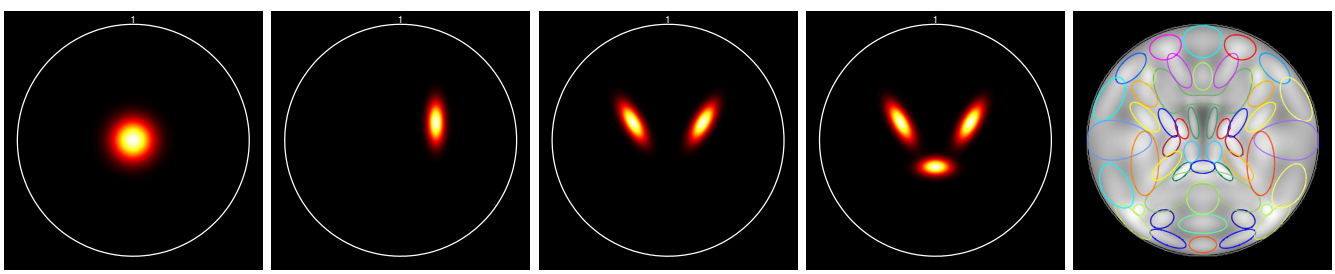


Figure 18: Progressive development of the new source (panels 1 through 4). The final source fits in well the default SMs (right).

As a first step, we will change the width parameters to elongate the blob in the y-direction by increasing w_x from 6 to 15. In addition, we will translate the blob rightwards 0.3 units and upwards 0.15 units. The effect of these transformations is shown in panel two of Figure 18.

```
elseif sourceID == 31
    %% New Source: gray matter ellipse at (0.3, 0.15)
    theta = 0;
    z = make_blob(x,y, .3 + randx, .15 + randy, 15, 6, theta + randrot);
    TT = 3;
```

Next, we will rotate the blob slightly ($\theta = \pi/6$ radians), and will add a second blob that is the mirror image of the first. The outcome is shown in the third panel of Figure 18.

```
elseif sourceID == 31
    %% New Source: two gray matter ellipses, rotated pi/6 radians
    theta = pi/6;
    % right blob
    z1 = make_blob(x,y, .3 + randx, .15 + randy, 15, 6, theta + randrot);
    % left blob
    z2 = make_blob(x,y, -.3 + randx, .15 + randy, 15, 6, -theta + randrot);
    % equivalently: z2 = fliplr(z1);
    z = z1 + z2;
    TT = 3;
```

As a last step, we will add a third blob that is centrally located. This blob will be created using a slightly different sub-function, `make_blob_grad`. The sub-function `make_blob_grad` creates a 2-D Gaussian that is also multiplied by a weak gradient in the x-direction. Here, we would like the gradient to appear in the y-direction, thus we rotate the blob by $\theta = \pi/2$ radians.

```
elseif sourceID == 31
    %% New Source: three activation blobs
    theta = pi/6;
    % right blob
    z1 = make_blob(x,y, .3 + randx, .15 + randy, 15, 6, theta + randrot);
    % left blob
    z2 = make_blob(x,y, -.3 + randx, .15 + randy, 15, 6, -theta + randrot);
    % central blob
    theta2 = -pi/2;
    z3 = make_blob_grad(x,y, 0 + randx, -.23 + randy, 15, 8, theta2 + randrot);
    z = z1 + z2 + z3;
    TT = 3;
```

The resulting SM is shown in the fourth panel of Figure 18. Note that while `make_blob_grad` provides a very mild departure from a simple 2-D Gaussian, advanced users can easily define their own sub-functions to create activations blobs with any 2-D distribution. Finally, we use the code below to examine the spatial configuration of all sources. In the rightmost panel of Figure 18 we observe minimal overlapping between SMs.

```
figure_handle = figure; set(figure_handle, 'Color', 'k') % Create a black figure
H = axes; set(H, 'Position', [0.05 0.05 0.9 0.9]);       % Create empty axes
simtb_figure_drawSMs(H, nV);                             % Draw the SM contours
```

6.5 Create a new TC source model

The toolbox includes three models to generate TCs from event time series (see Section 2.2). Users can modify existing models or create their own by editing the function `simtb_TCsource`. For a new model, one adds another conditional statement (`elseif`) to the bottom of the list of model definitions. New models must define a TC given the event time series, repetition time and set of model-specific parameters. A template for defining and describing a model is provided in the code block below. This particular model requires 3 parameters in addition to the event time series and repetition time. Note that the actual model is implemented in a user-defined function, `my_model_function`.

```
elseif sourceType == 4
    MDESC = 'My new TC generation model';
    PDESC = sprintf([...
        '\tP(1): description of parameter 1\n',...
        '\tP(2): description of parameter 2\n',...
        '\tP(3): description of parameter 3']);
    % Generate the TC from eTC, TR and P
    tc = my_model_function(eTC, TR, P);
    % eTC: [nT x 1] event time series
    % TR:  repetition time
    % P:   [1 x 3] vector of model parameters
    % tc:  [nT x 1] component TC
```

In addition to defining and labeling the model, one must provide default parameters. This is done by adding a conditional statement to the model list in the sub-function `get_default_params`. For the model defined above, one might define default parameters as the following.

```
elseif sourceType == 4
    % My new TC generation model
    % P: 1x3 vector of parameters for my model
    P(1) = 1;               % parameter 1, fixed
    P(2) = 0 + randn(1);    % parameter 2, varies
    P(3) = rand(1);         % parameter 3, varies
```

If new models are added correctly, they should appear when calling `simtb_countTCmodels`, a helper function which displays descriptions of models and their parameters.

```
simtb_countTCmodels;
```

```
 SOURCE TYPE: 1
  MODEL DESC: Convolution with canonical HRF (difference of two gamma functions)
  PARAM DESC: [7 parameters]
    P(1): delay of response (relative to onset)
    P(2): delay of undershoot (relative to onset)
    P(3): dispersion of response
    P(4): dispersion of undershoot
    P(5): ratio of response to undershoot
    P(6): onset (seconds)
    P(7): length of kernel (seconds)

 SOURCE TYPE: 2
  MODEL DESC: Windkessel Balloon Model, see Friston et al., Neuroimage (2000)
```

```
 PARAM DESC: [7 parameters]
   P(1): 1/(signal decay)                                (1/Ts)
   P(2): 1/(autoregulation)                              (1/Tf)
   P(3): transit time                                    (t0)
   P(4): stiffness                                       (alpha)
   P(5): resting oxygen extraction fraction              (E0)
   P(6): echo time (seconds)                             (TE)
   P(7): neural efficacy                                 (epsilon)

SOURCE TYPE: 3
 MODEL DESC: Convolution with fast spike (difference of two gamma functions)
 PARAM DESC: [7 parameters]
   P(1): delay of response (relative to onset)
   P(2): delay of undershoot (relative to onset)
   P(3): dispersion of response
   P(4): dispersion of undershoot
   P(5): ratio of response to undershoot
   P(6): onset (seconds)
   P(7): length of kernel (seconds)

SOURCE TYPE: 4
 MODEL DESC: My new TC generation model
 PARAM DESC: [3 parameters]
   P(1): description of parameter 1
   P(2): description of parameter 2
   P(3): description of parameter 3
```

6.6 Modify the tissue baseline

As a default, the SimTB toolbox defines four different tissue types (TT) representing signal dropout (Type 1), white matter (WM, Type 2), gray matter (GM, Type 3) and cerebral spinal fluid (CSF, Type 4). Individuals using the SimTB GUI can choose whether or not to use the tissue type (TT) model with TT levels set to their default levels. To change default levels, users may edit the helper function `simtb_countTT` which determines the number of defined TTs and their defaults. Lines 37 to 44 of `simtb_countTT` provide TT definitions:

```
37 %% edit here to change defaults for the TT_levels
38 % As a default, levels 1-4 are defined.
39 % Any additional TT_levels are set to 1.
40 TT_levels = ones(1,TT_count);
41 TT_levels(1) = 0.3; %signal dropout
42 TT_levels(2) = 0.7; %WM
43 TT_levels(3) = 1.0; %GM
44 TT_levels(4) = 1.5; %CSF
```

Suppose that we would like to use the TT model, but only want to include WM, GM, and CSF (no signal dropout). Furthermore, we would like to change the WM and CSF levels to be more similar to GM. We then change `simtb_countTT` to the following:

```
41 TT_levels(1) = 1.0; %signal dropout (no dropout in this case)
42 TT_levels(2) = 0.8; %WM
43 TT_levels(3) = 1.0; %GM
44 TT_levels(4) = 1.3; %CSF
```

Now any sources defined with `TT = 1` in `simtb_SMsource` will have baseline intensities identical to those defined with `TT = 3`. The original baseline and modified baseline are displayed in Figure 19 and were created using the display function `simtb_figure_model`:

```
model_figure = simtb_figure_model(sP, 1, [], 2);
```

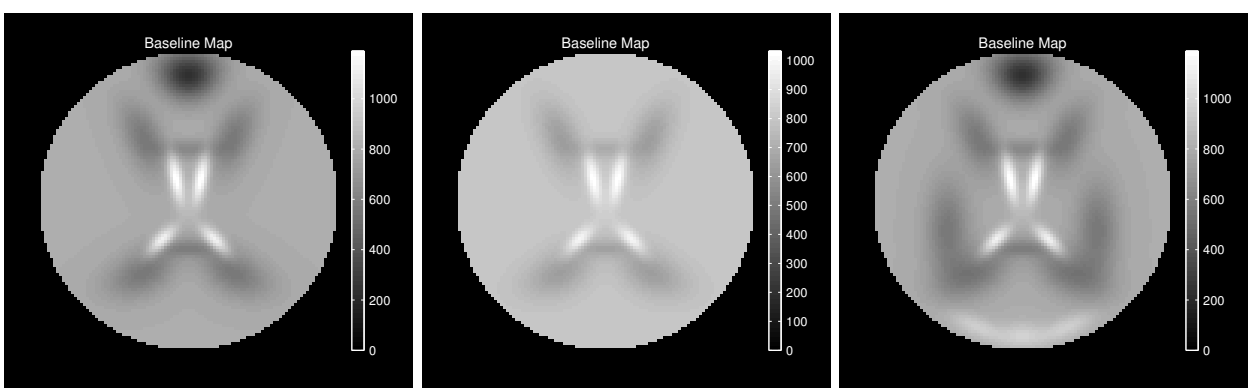


Figure 19: Default (left) and modified (center, right) baselines.

Individuals using scripts to define parameters can simply designate TT levels in the parameter file.

```
D_TT_FLAG = 1;                        % Choose to use the TT model
D_TT_level = [1.0, 0.8, 1.0, 1.3];  % [GM, WM, GM, CSF] (no signal dropout)
```

Note that one may also load in a previously saved parameter structure and change TT levels at the command line. For example,

```
load('X:\MyData\Simulations\sim_PARAMS.mat')  % load a parameter structure 'sP'
sP.D_TT_level = [1.0, 0.8, 1.0, 1.3];         % [GM, WM, GM, CSF] (no signal dropout)
[errorflag, Message] = simtb_checkparams(sP); % check the parameter structure
save('X:\MyData\Simulations\sim_PARAMS','sP') % save the modified parameters
```

If a user would like to create more TT levels or to alter the TTs of particular sources, modifications should be made in `simtb_SMsource`. In the following excerpts from `simtb_SMsource`, we assign a new TT (Type 5) to Sources 3 and 11, modeling field inhomogeneity where intensity is greater in posterior regions. We also change the TT of Sources 27 and 28 from GM (Type 3) to WM (Type 2).

```
elseif sourceID == 3
    %% Bilateral Visual - more posterior
    theta = -pi/8;
    % right
    z1 = make_blob(x,y, 0.3 + randx, -.85 + randy, 5, 15, theta + randrot);
    % left
    z2 = make_blob(x,y, -0.3 + randx, -.85 + randy, 5, 15, -theta + randrot);
    z = z1 + z2;
    TT = 5; % Changed from 3 (normal GM) to 5 (GM of increased intensity)
```

```
elseif sourceID == 11
    %% Medial Visual cortex
    theta = 0;
    z = make_blob(x,y, 0 + randx, -0.9 + randy, 7, 12, theta + randrot);
    TT = 5; % Changed from 3 (normal GM) to 5 (GM of increased intensity)
```

```
elseif sourceID == 27
    %% Left Auditory
    theta = 0;
    z = make_blob(x,y, 0.5 + randx, -0.2 + randy, 7, 3, theta + randrot);
    TT = 2;  % Changed from 3 (normal GM) to 2 (WM)
elseif sourceID == 28
    %% Right Auditory
    theta = 0;
    z = make_blob(x,y, -0.5 + randx, -0.2 + randy, 7, 3, theta + randrot);
    TT = 2;  % Changed from 3 (normal GM) to 2 (WM)
```

The newly defined TT will automatically be assigned a default level of 1.0, which can be seen by using `simtb_countTT`:

```
simtb_countTT;
```

```
Number of defined TT levels: 5
Default TT levels: [0.3, 0.7, 1.0, 1.5, 1.0]
```

We can adjust the level of the new TT by defining `D_TT_level` in the parameter file as shown below or by updating defaults in `simtb_countTT`.

```
D_TT_FLAG = 1;                          % Choose to use the TT model
D_TT_level = [0.3, 0.7, 1.0, 1.5, 1.2]; % [dropout, WM, GM, CSF, GM_brighter]
```

The baseline tissue map resulting from these modifications is displayed in the rightmost panel of Figure 19.

7 Functions

A list of MATLAB functions for `simtb_v18` follows. Note that HTML versions of function documentation can be accessed within MATLAB by typing `simtb_doc 'function_name'` and online at http://mialab.mrn.org/software.

Main directory .

`simtb()` -- Initializes the simtb GUI

Subsequent directories:

- GUI
- analyze
- display
- examples
- helper
- htmldoc
- io
- sim

Subdirectory `sim`: the workhorse functions

`simtb_SMsource()` -- Contains the definitions of the 2-D SMs and their tissue types

`simtb_TCsource()` -- Contains the definitions for the generation of TCs

`simtb_addMotion()` -- Simulates motion (translation/rotation) of dataset

`simtb_balloon_model()` -- Nonlinear balloon model to generate TCs

`simtb_create_sP()` -- Initializes parameter structure

`simtb_generateSM()` -- Generates a SM based on definitions in `simtb_SMsource()`

`simtb_main()` -- Main function for performing simulations

`simtb_makeBaseline()` -- Generate baseline intensity map for each subject

`simtb_makeMotParams()` -- Generates motion time series

`simtb_makeSM()` -- Makes component SMs (spatial maps)

`simtb_makeTC()` -- Makes component TCs (time courses)

`simtb_makeTC_block()` -- Builds task blocks

`simtb_makeTC_event()` -- Builds event time courses

`simtb_padSM()` -- Pads SM dimensions to accomodate head translation

`simtb_rand_seed()` -- Set random number generator (RNG) seeds

`simtb_spm_Gpdf()` -- Probability Density Function (PDF) of a Gamma distribution

`simtb_spm_hrf()` -- Returns a hemodynamic response function

Subsequent directories:

- `spm`

Subdirectory `helper`: functions that help the user

`simtb_checkparams()` -- Checks for appropriate and consistent simulation parameters

`simtb_countSM()` -- Determines the number of sources defined in `simtb_SMsource()`

`simtb_countTCmodels()` -- Determines the number of TC models defined in `simtb_TCsource()`

`simtb_countTT()` -- Determines the number and default levels of Tissue Types

`simtb_doc()` -- Opens HTML documentation for a given function

`simtb_makefilename()` -- Makes filenames for various file types

`simtb_params()` -- Provides information on all simulation parameters

`simtb_returnFileIndex()` -- Returns the file index for naming

Subdirectory `io`: functions for managing input/output

`simtb_createmask()` -- Creates data mask with ones inside the head and zeros outside.

`simtb_saveMOT()` -- Writes motion parameters to file

`simtb_saveasnii()` -- Save data matrix as an .nii file

Subdirectory `display`: functions used to display simulation features

`simtb_figdimension()` -- Computes position of figure given desired aspect ratio and scale

`simtb_figure_drawSMs()` -- Shows contours for all defined SMs

`simtb_figure_model()` -- Shows SM contours and/or Tissue Model

`simtb_figure_output()` -- Shows TC/SM output for a given subject

`simtb_figure_params()` -- Shows simulation parameters

`simtb_figure_pickedSM()` -- Draws and fills SM contours for selected sources

`simtb_inpoly()` -- Tests whether a point is inside a polygon

`simtb_lighten_color()` -- Lightens color (reduces saturation)

`simtb_movie()` -- Displays a movie of a subject dataset at 10 frames a second

`simtb_pcolor()` -- Produces pcolor-like images of data matrices

`simtb_pickSM()` -- GUI for selecting sources

`simtb_showSM()` -- Plot SMs

`simtb_showSMContours()` -- Plot SMs as contours

`simtb_showTC()` -- Show generation of TCs (time courses)

`simtb_show_motion()` -- Shows translational/rotational motion of subjects

Other Matlab-specific files in this directory:

- `CM_coldhot_256.mat`

Subsequent directories:

- `cm_and_cb_utilities`
- `freezeColors`

Subdirectory `analyze`: functions used to load and analyze simulation features

`simtb_est_accuracy()` -- Determines the accuracy of the estimated features

`simtb_group_getSM()` -- Loads saved subject SMs (spatial maps)

`simtb_group_getTC()` -- Loads saved subject TCs (time courses)

`simtb_match_EST2TRUE()` -- Matches estimated to true SMs based on maximum R^2 statistics

`simtb_regression()` -- Performs least squares multiple linear regression

8 Example Parameter Files

8.1 experiment_params_block.m

The script below represents a minimal parameter file. Just a few parameters are defined to implement a block design and modify experiment length and CNR. Most parameters will retain their default values, which can be viewed using `simtb_params`.

```
1  %-------------------------------------------------------------------------
2  % experiment_params_block.m
3  % simtb_v18 03/28/11
4  % We define parameters to:
5  %       1. Change the output path and file prefix
6  %       2. Increase the number of time points from 150 (default) to 260
7  %       3. Implement a block design with 2 conditions
8  %       4. Increase the CNR for all subjects from 1 (default) to 2
9  % Parameters that are not defined here will take on their default values.
10 %
11 % To create the simulation parameter structure:
12 % >> sP = simtb_create_sP('experiment_params_block', M, nC);
13 %    Simulation can be executed with any number of subjects, M, or components, nC,
14 %    though nC should be ≥ 4 given task modulation amplitudes (see Lines 46-49).
15 % To run the simulation:
16 % >> simtb_main(sP)
17 %-------------------------------------------------------------------------
18
19 %% OUTPUT PARAMETERS
20 %-------------------------------------------------------------------------
21 % Directory to save simulation parameters and output
22 out_path = 'X:\MyData\Simulations\';
23 prefix = 'block';  % Prefix for saving output
24 %-------------------------------------------------------------------------
25
26 %% RANDOMIZATION
27 %-------------------------------------------------------------------------
28 seed = round(sum(100*clock));   % randomizes parameter values
29 simtb_rand_seed(seed);          % set the seed
30 %-------------------------------------------------------------------------
31
32 %% SIMULATION DIMENSIONS
33 %-------------------------------------------------------------------------
34 nT = 260; % Increase the length of the experiment
35 %-------------------------------------------------------------------------
36
37 %% EXPERIMENT DESIGN
38 %-------------------------------------------------------------------------
39 % BLOCKS
40 TC_block_n = 2;           % Number of blocks [set = 0 for no block design]
41 TC_block_same_FLAG = 0;   % 1 = block structure same for all subjects
42                           % 0 = block order will be randomized
43 TC_block_length = 20;     % length of each block (in samples)
44 TC_block_ISI    = 15;     % length of OFF inter-stimulus-intervals (in samples)
45 TC_block_amp    = zeros(nC, TC_block_n); % initialize [nC x TC_block_n] matrix
```

```
TC_block_amp(3,1) = 2;   % Comp 3 is strongly modulated by condition 1
TC_block_amp(3,2) = 0.5; % Comp 3 is weakly modulated by condition 2
TC_block_amp(4,1) = -1;  % Comp 4 is negatively modulated by condition 1
TC_block_amp(4,2) = 1.5; % Comp 4 is strongly modulated by condition 2
%-------------------------------------------------------------------------

%% NOISE
%-------------------------------------------------------------------------
D_CNR = 2*ones(1,M);   % Increase the CNR to be 2 for all subjects
%-------------------------------------------------------------------------
% END of parameter definitions
```

8.2 experiment_params_aod.m

This script represents a full parameter file. Every simulation parameter is defined. The simulation is fairly complex and implements an event-related task design, multiple TC generation models, different tissue baselines, and subject motion.

```
%-------------------------------------------------------------------------
% experiment_params_aod.m
% simtb_v18 03/28/11
% Example script for setting/modifying simtb simulation parameters, from
%   "SimTB, a simulation toolbox for fMRI data
%     under a model of spatiotemporal separability"
%   E.B. Erhardt, E.A. Allen, Y. Wei, T. Eichele, V.D. Calhoun
%
% To create the simulation parameter structure:
% >> sP = simtb_create_sP('experiment_params_aod');
%
% To run the simulation:
% >> simtb_main(sP)
%
% Futher information on any parameter can be found by typing:
% >> simtb_params(param_name)
%-------------------------------------------------------------------------

%% OUTPUT PARAMETERS
%-------------------------------------------------------------------------
% Directory to save simulation parameters and output
out_path = 'X:\MyData\Simulations\';
% Prefix for saving output
prefix = 'aod';
% FLAG to write data in NIFTI format rather than matlab
saveNII_FLAG = 0;
% Option to display output and create figures throughout the simulations
verbose_display = 1;
%-------------------------------------------------------------------------

%% RANDOMIZATION
%-------------------------------------------------------------------------
%seed = round(sum(100*clock));  % randomizes parameter values
seed = 3571;                   % choose seed for repeatable simulation
```

```
simtb_rand_seed(seed);          % set the seed
%-------------------------------------------------------------------------------

%% SIMULATION DIMENSIONS
%-------------------------------------------------------------------------------
M  = 5;   % number of subjects
% nC is the number of components defined below, nC = length(SM_source_ID);
nV = 148; % number of voxels; dataset will have [nV x nV] voxels.
nT = 150; % number of time points
TR = 2;   % repetition time
%-------------------------------------------------------------------------------

%% SPATIAL SOURCES
%-------------------------------------------------------------------------------
% Choose the sources. To launch a stand-alone GUI:
% >> simtb_pickSM
SM_source_ID = [    2  3  4  5  6  7  8  9     ...
                11 12    14 15 16 17 18 19 20  ...
                21 22 23 24 25 26 27 28 29 30]; % all but (1, 10, 13)

nC = length(SM_source_ID);  % number of components

% LABEL COMPONENTS
% Here, we label components or component groups that may be used later
% Auditory: strong positive activation for all task events
comp_AUD1  = find(SM_source_ID == 27);
comp_AUD2  = find(SM_source_ID == 28);
% DMN: negative activation to task events
comp_DMN   = find(SM_source_ID ==  8);
% Bilateral frontal: positive activation to for targets and novels
comp_BF    = find(SM_source_ID == 24);
% Frontal: 1 second temporal delay from bilateral frontal
comp_F1    = find(SM_source_ID ==  4);
comp_F2    = find(SM_source_ID ==  5);
% Precuneus: activation only to targets
comp_P     = find(SM_source_ID ==  7);
% Dorsal Attention Network: activation to novels more than targets
comp_DAN   = find(SM_source_ID == 18);
% Hippocampus: activation only to novels
comp_H1    = find(SM_source_ID == 29);
comp_H2    = find(SM_source_ID == 30);
% (Sensori)Motor: activation to targets and novels (weakly)
comp_M1    = find(SM_source_ID == 22);
comp_M2    = find(SM_source_ID == 23);
% CSF and white matter: unaffected by task, but has signal amplitude differences
comp_CSF1  = find(SM_source_ID == 14);
comp_CSF2  = find(SM_source_ID == 15);
comp_WM1   = find(SM_source_ID == 16);
comp_WM2   = find(SM_source_ID == 17);
% Medial Frontal: has lower baseline intensity (signal dropout)
comp_MF    = find(SM_source_ID ==  6);

% compile list of all defined components of interest
complist = [comp_AUD1 comp_AUD2 comp_DMN comp_BF  comp_F1 comp_F2 ...
```

```
            comp_P    comp_DAN  comp_H1  comp_H2  comp_M1 comp_M2 ...
            comp_CSF1 comp_CSF2 comp_WM1 comp_WM2 comp_MF];
%-------------------------------------------------------------------------------

%% COMPONENT PRESENCE
%-------------------------------------------------------------------------------
% [M x nC] matrix for component presence: 1 if included, 0 otherwise
% For components not of interest there is a 90% chance of component inclusion.
SM_present = (rand(M,nC) < 0.9);
% Components of interest (complist) are included for all subjects.
SM_present(:,complist) = ones(M,length(complist));
%-------------------------------------------------------------------------------

%% SPATIAL VARIABILITY
%-------------------------------------------------------------------------------
% Variability related to differences in spatial location and shape.
SM_translate_x = 0.1*randn(M,nC);  % Translation in x, mean 0, SD 0.1 voxels.
SM_translate_y = 0.1*randn(M,nC);  % Translation in y, mean 0, SD 0.1 voxels.
SM_theta       = 1.0*randn(M,nC);  % Rotation, mean 0, SD 1 degree.
%                Note that each 'activation blob' is rotated independently.
SM_spread = 1+0.03*randn(M,nC); % Spread < 1 is contraction, spread > 1 is expansion.
%-------------------------------------------------------------------------------

%% TC GENERATION
%-------------------------------------------------------------------------------
% Choose the model for TC generation.  To see defined models:
% >> simtb_countTCmodels

TC_source_type = ones(1,nC);    % convolution with HRF for most components
% to make statistical moments of data look more like real data
TC_source_type([comp_CSF1 comp_CSF2]) = 3; % spike model for CSF

TC_source_params = cell(M,nC);  % initialize the cell structure
% Use the same HRF for all subjects and relevant components
P(1) = 6;    % delay of response (relative to onset)
P(2) = 16;   % delay of undershoot (relative to onset)
P(3) = 1;    % dispersion of response
P(4) = 1;    % dispersion of undershoot
P(5) = 6;    % ratio of response to undershoot
P(6) = 0;    % onset (seconds)
P(7) = 32;   % length of kernel (seconds)
[TC_source_params{:}] = deal(P);

% Implement 1 second onset delay for components comp_F1 and comp_F2
P(6) = P(6) + 1;  % delay by 1s
[TC_source_params{:,[comp_F1 comp_F2]}] = deal(P);

sourceType = 3; % CSF components use spike model
% Generate a random set of parameters for TC model 3
[tc_dummy, MDESC, P3, PDESC] = simtb_TCsource(1, 1, sourceType);
% Assign identical parameters for model 3 to all subjects
[TC_source_params{:,[comp_CSF1 comp_CSF2]}] = deal(P3);
%-------------------------------------------------------------------------------

```

```
%% EXPERIMENT DESIGN
%-------------------------------------------------------------------------------
% BLOCKS
% No blocks for this experiment
TC_block_n = 0;           % Number of blocks [set = 0 for no block design]
% Note that if TC_block_n = 0 the rest of these parameters are irrelevant
TC_block_same_FLAG = 0;  % 1 = block structure same for all subjects
                         % 0 = otherwise order will be randomized
TC_block_length = 10;    % length of each block (in samples)
TC_block_ISI    = 10;    % length of OFF inter-stimulus-intervals (in samples)
TC_block_amp    = [];    % [nC x TC_block_n] matrix of task-modulation amplitudes

% EVENTS
TC_event_n = 4;           % Number of event types (0 for no event-related design)
                          % 1: standard tone
                          % 2: target tone
                          % 3: novel tone
                          % 4: 'spike' events in CSF (not related to task)
TC_event_same_FLAG = 0;  % 1=event timing will be the same for all subjects

% event probabilities (0.6 standards, 0.075 targets and novels, 0.05 CSF spikes)
TC_event_prob = [0.6, 0.075, 0.075, 0.05];  % an 8:1:1 ratio

% initialize [nC x TC_event_n] matrix of task-modulation amplitudes
TC_event_amp = zeros(nC,TC_event_n);
% event type 1: standard tone
TC_event_amp([comp_AUD1 comp_AUD2],              1) = 1.0; % moderate task-modulation
TC_event_amp([comp_BF comp_F1 comp_F2 comp_DAN], 1) = 0.7; % mild
TC_event_amp([comp_DMN],                         1) =-0.3; % negative weak
% event type 2: target tone
TC_event_amp([comp_AUD1 comp_AUD2],              2) = 1.2; % strong
TC_event_amp([comp_BF comp_F1 comp_F2],          2) = 1.0; % moderate
TC_event_amp([comp_DAN],                         2) = 0.8; % mild
TC_event_amp([comp_P],                           2) = 0.5; % weak
TC_event_amp([comp_M1 comp_M2],                  2) = 1.0; % moderate
TC_event_amp([comp_DMN],                         2) =-0.3; % negative weak
% event type 3: novel tone
TC_event_amp([comp_AUD1 comp_AUD2],              3) = 1.5; % very strong
TC_event_amp([comp_BF comp_F1 comp_F2],          3) = 1.0; % moderate
TC_event_amp([comp_DAN],                         3) = 1.2; % strong
TC_event_amp([comp_H1 comp_H2],                  3) = 0.8; % mild
TC_event_amp([comp_M1 comp_M2],                  3) = 0.5; % weak
TC_event_amp([comp_DMN],                         3) =-0.3; % negative weak
% event type 4: 'spikes' in CSF (not related to task)
TC_event_amp([comp_CSF1 comp_CSF2],              4) = 1.0; % moderate
%-------------------------------------------------------------------------------

%% UNIQUE EVENTS
%-------------------------------------------------------------------------------
TC_unique_FLAG = 1; % 1 = include unique events
TC_unique_prob = 0.2*ones(1,nC); % [1 x nC] prob of unique event at each TR

TC_unique_amp  = ones(M,nC);     % [M x nC] matrix of amplitude of unique events
% smaller unique activations for task-modulated and CSF components
```

```
TC_unique_amp(:,[comp_AUD1 comp_AUD2])              = 0.2;
TC_unique_amp(:,[comp_BF comp_F1 comp_F2])          = 0.3;
TC_unique_amp(:,[comp_DAN])                         = 0.5;
TC_unique_amp(:,[comp_P])                           = 0.5;
TC_unique_amp(:,[comp_M1 comp_M2])                  = 0.2;
TC_unique_amp(:,[comp_H1 comp_H2])                  = 0.4;
TC_unique_amp(:,[comp_DMN])                         = 0.3;
TC_unique_amp(:,[comp_CSF1 comp_CSF2])              = 0.05; %very small
%-------------------------------------------------------------------------------

%% DATASET BASELINE
%-------------------------------------------------------------------------------
% [1 x M] vector of baseline signal intensity for each subject
D_baseline = 800*ones(1,M); % [1 x M] vector of baseline signal intensity
%-------------------------------------------------------------------------------

%% TISSUE TYPES
%-------------------------------------------------------------------------------
% FLAG to include different tissue types (distinct baselines in the data)
D_TT_FLAG = 1;                     % if 0, baseline intensity is constant
D_TT_level = [1.15, 0.8, 1, 1.2]; % TT fractional intensities
% To see/modify definitions for tissue profiles:
% >> edit simtb_SMsource.m
%-------------------------------------------------------------------------------

%% PEAK-TO-PEAK PERCENT SIGNAL CHANGE
%-------------------------------------------------------------------------------
D_pSC = 3 + 0.25*randn(M,nC);   % [M x nC] matrix of percent signal changes

% To make statistical moments of data look more like real data
D_pSC(:,comp_CSF1) = 1.2*D_pSC(:,comp_CSF1);
D_pSC(:,comp_CSF2) = 1.2*D_pSC(:,comp_CSF2);
D_pSC(:,comp_WM1)  = 0.5*D_pSC(:,comp_WM1);
D_pSC(:,comp_WM2)  = 0.5*D_pSC(:,comp_WM2);
%-------------------------------------------------------------------------------

%% NOISE
%-------------------------------------------------------------------------------
D_noise_FLAG = 1;               % FLAG to add rician noise to the data
% [1 x M] vector of contrast-to-noise ratio for each subject
% CNR is distributed as uniform between 0.65 and 2.0 across subjects.
minCNR = 0.65;  maxCNR = 2;
D_CNR = rand(1,M)*(maxCNR-minCNR) + minCNR;
%-------------------------------------------------------------------------------

%% MOTION
%-------------------------------------------------------------------------------
D_motion_FLAG = 1;              % 1=motion, 0=no motion
D_motion_TRANSmax = 0.02;       % max translation, proportion of entire image
D_motion_ROTmax = 5;            % max rotation, in degrees
D_motion_deviates = ones(M,3);  % proportion of max each subject moves
D_motion_deviates(1,:) = 0.5;   % Subject 1 moves half as much
%-------------------------------------------------------------------------------
% END of parameter definitions
```

Acknowledgements

This research was supported by NIH 1R01-EB006841, NIH 1R01-EB005846, NIH 2R01-EB000840, NIH 1 P20 RR021938-01, and DOE DE-FG02-08ER64581 (PI: Calhoun). We thank Eswar Damaraju, Martin Havlicek, and Arvind Caprihan for providing comments that improved the toolbox. We also thank the developers of EEGLAB (Arnaud Delorme and Scott Makeig) and `m2html` (Guillaume Flandin), whose excellent toolboxes and documentation served as a model for this work.

References

Allen, E., Erhardt, E., Wei, Y., Eichele, T., Calhoun, V., 2011a. Capturing inter-subject variability with group independent component analysis of fMRI data: a simulation study. In submission.

Allen, E. A., Erhardt, E. B., Damaraju, E., Gruner, W., Segall, J. M., Silva, R. F., Havlicek, M., Rachakonda, S., Fries, J., Kalyanam, R., Michael, A. M., Caprihan, A., Turner, J. A., Eichele, T., Adelsheim, S., Bryan, A. D., Bustillo, J., Clark, V. P., Ewing, S. W. F., Filbey, F., Ford, C. C., Hutchison, K., Jung, R. E., Kiehl, K. A., Kodituwakku, P., Komesu, Y. M., Mayer, A. R., Pearlson, G. D., Phillips, J. P., Sadek, J. R., Stevens, M., Teuscher, U., Thoma, R. J., Calhoun, V. D., 2011b. A baseline for the multivariate comparison of resting state networks. Frontiers in Systems Neuroscience 5, 12.

Erhardt, E., Allen, E., Wei, Y., Eichele, T., Calhoun, V., 2011. SimTB, a simulation toolbox for fMRI data under a model of spatiotemporal separability. In submission.

Friston, K., Mechelli, A., Turner, R., Price, C., 2000. Nonlinear responses in fMRI: the Balloon model, Volterra kernels, and other hemodynamics. NeuroImage 12 (4), 466--477.

Friston, K. J., Holmes, A. P., Worsley, K. J., Poline, J. B., Frith, C. D., Frackowiak, R. S. J., et al., 1995. Statistical parametric maps in functional imaging: a general linear approach. Human Brain Mapping 2 (4), 189--210.

Gudbjartsson, H., Patz, S., 1995. The Rician distribution of noisy MRI data. Magnetic Resonance in Medicine 34 (6), 910--914.

Kiehl, K., Laurens, K., Duty, T., Forster, B., Liddle, P., 2001. An event-related fMRI study of visual and auditory oddball tasks. Journal of Psychophysiology 15 (4), 221--240.