Guide to the STACMR-Matlab package

John C. Dunn, Michael L. Kalish

13 February 2018
Revised, 26 March 2020

STACMR is a set of Matlab functions that implement the Conjoint Monotonic Regression (CMR) approach to State-Trace Analysis (STA). This is a quick guide to its use.

Contents

1 Getting started

1.1 Download STACMR software

1. Go to `https://github.com/michaelkalish/STA`. You should see something similar to Figure 1.

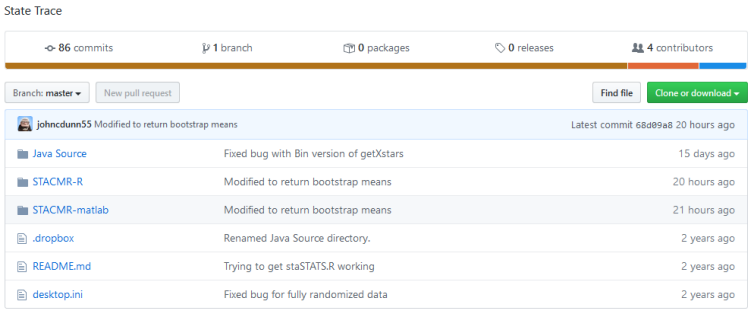

Figure 1: `https://github.com/michaelkalish/STA`.

2. Click the green box labelled **Clone or download**. You will then see the dialog box shown in Figure 2.

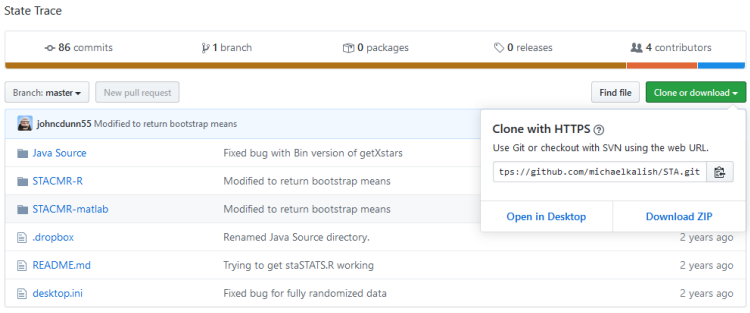

Figure 2: Click **Download ZIP**.

3. Click the option labelled **Download ZIP**. You will then see the dialog box shown in Figure 3.

4. Click **OK** and save `STACMR-master.zip` to a folder of your choice.

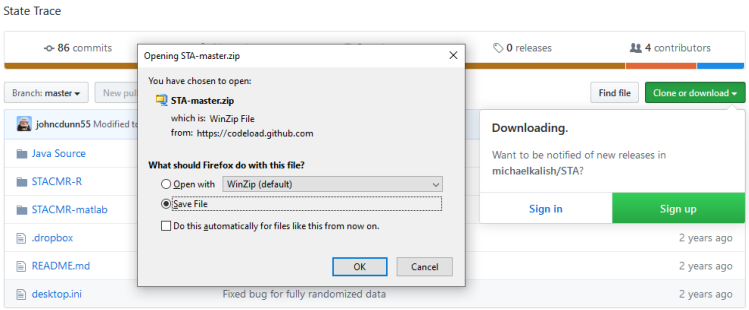

Figure 3: Click **OK**.

5. Unzip `STACMR-master.zip` and place the folder `STA-master` and its subfolders on the Matlab path. You will only need the subfolder `STACMR-matlab`.

1.2 Install Java Runtime Environment (JRE)

1. STACMR-matlab requires JRE to be installed. Go to `https://www.java.com/en/download/` and follow the prompts. Be careful to download the version appropriate to your machine and operating system.

2. Launch the JRE executable (e.g., `jre-8u241-windows-x64.exe`) that you have downloaded to finish installation.

1.3 Link STACMR to the current java library

1. In Matlab, run `staCMRsetup`. If it is successful you should see a message similar to the following:

```
STACMR program library Version 21.02.2018
Utility programs for use with the book:
Dunn, J. C. & Kalish, M. L. (2018). State-Trace Analysis. Springe

STACMR linked to java library fxMR-0.3.44.jar
```

2. As stated in the message, at the time of writing, the current java library is `fxMR-0.3.44.jar`. This will change in future releases.

3. You should now be good to go.

2 Partial order

The various STACMR functions often make use of an optional partial order. A partial order on a vector, x, is a set of pairs, (i, j), such that $x_i \leq x_j$. This is represented in STACMR in two (equivalent) ways:

1. As a cell array containing the set of (i, j) pairs, such as, {`[1 2], [2 3], [1 4]`}. There is a shorthand for a linear order such as, {`[1 2], [2 3]`}, which can be written as, {`[1 2 3]`} or, even more simply, as {`1:3`}.

2. As an adjacency matrix in which entry $(i, j) = 1$ if (i, j) is an element of the partial order, otherwise $(i, j) = 0$.

The function, `cell2adj`, converts a partial order in cell array form into its corresponding adjacency matrix form. For example,

```
>> E = {[1 2] [2 3] [1 4]};
>> A = cell2adj (1:4, E)
A =
     0     1     0     1
     0     0     1     0
     0     0     0     0
     0     0     0     0
```

In the above call to `cell2adj`, the vector, 1:4, specifies the set that the partial order applies to. It is almost invariably the sequence of numbers, 1 to k, where k is the total number of conditions. In the above example, $k = 4$. The function `adj2cell` converts an adjacency matrix into its corresponding cell array.

3 Continuous data

Continuous data has the form that each observation is a number drawn from a continuous distribution.

3.1 Input data structures

At present, STACMR accepts two kinds of data structure for continuous data:

1. **Cell array format**. In this format, the data are contained in a $b \times n$ cell array where b is the number of between-participant conditions (groups) and n is the number of dependent variables. Each component of this cell array is itself an $N \times w$ matrix of observations where N is the number of subjects (which may vary across groups and dependent variables) and w is the number of within-participant conditions (fixed across groups and dependent variables). The dependent variable may be either within-participant or between-participant it doesnt matter because the correlation between dependent variables is assumed to be zero (although this might change in future implementations).

2. **General format**. This is a fixed column format organised as a matrix in which each row corresponds to an observation and each column is defined as follows:

   column 1: Participant number (for identification only, not used directly)

   column 2: Between-participant condition or group (if none, then set this value to 1)

   column 3: Dependent variable (numbered 1, 2, and so on)

   columns 4 to end: Values for each within-participant condition

While STACMR accepts data in general format it always converts it to cell array format using the function,

```
>> y = gen2cell (data);
```

3.2 Principal functions

The principle functions are located in the folder, `.../STACMR-matlab`. Their operation is illustrated with respect to the dataset, `delay`, located in folder, `.../STACMR-matlab/Data files`.

The `delay` data set comes from the study by Dunn, J. C., Newell, B. R., & Kalish, M. L. (2012). The effect of feedback delay and feedback type on perceptual category learning. *Journal of Experimental Psychology: Learning, Memory and Cognition, 38*(4), 840-859.

Participants in this study completed one of two category-learning tasks over 4 blocks of training (a within-participant factor) under one of 2 different conditions (a between-participant factor). The dependent variables are the proportions correct for each of two tasks defined according to the category

structure that participants learned. For the rule-based (RB) group, the category structure was defined by a simple rule. For the information-integration (II) group, the category structure was more complex and could not be defined by a simple rule.

The data are in general format and is contained in the file, `delay.dat`. That is,

```
>> delay = load('delay.dat'); % read the data file
>> y = gen2cell(delay); % convert to cell array format
```

3.2.1 staSTATS.m

This function computes summary statistics of a data structure in cell array format. Example call:

```
>> delaystats = staSTATS (y, shrink);
```

Input:

Here, `y` is the delay data in cell array format and `shrink` is an optional parameter denoting how much shrinkage to apply to the estimated covariance matrix. Generally, the covariance matrix needs to be shrunk during the bootstrap cycle to avoid ill-conditioning. If `shrink = 0` then no shrinkage is applied. If `shrink = 1` then maximum shrinkage is applied. This means that the covariance matrix is diagonalized with all off-diagonal entries set to zero. If `shrink < 0` (the default) then an optimal shrinkage value is estimated for each within-participant block and applied according to an algorithm developed by: Ledoit, O. & Wolf, M. (2004). Honey, I shrunk the sample covariance matrix, *The Journal of Portfolio Management, 30*(4), 110-119.

Output:

`staSTATS` returns in the variable `delaystats` a cell array of length equal to the number of dependent variables in `y`. Each component of `delaystats` is a structured array. For the dependent variable, `ivar`:

- `delaystats{ivar}.means` = vector of means across all conditions
- `delaystats{ivar}.cov` = the covariance matrix (for information only)
- `delaystats{ivar}.regcov` = the adjusted covariance matrix following application of shrinkage

- `delaystats{ivar}.n` = matrix of number of observations (subjects) in each within-participant block
- `delaystats{ivar}.lm` = matrix of Loftus-Masson within-participant standard errors (used by staPLOT)
- `delaystats{ivar}.weights` = matrix of weights defined by: `delaystats{ivar}.n.*delaystats{ivar}.regcov^-1`
- `delaystats{ivar}.shrinkage` = a vector of length b (where b is the number of levels of the between-participant independent variable) containing the specified or estimated shrinkage values

Thus, `delaystats` has two components corresponding to the two dependent variables in `delay`. If you look at the means, they should look like this:

```
>> disp(delaystats{1}.means)
0.3676  0.4676  0.5757  0.6118  0.3445  0.4434  0.5081
0.5169
>> disp(delaystats{2}.means)
0.3308  0.4550  0.5346  0.5492  0.2836  0.3031  0.3180
0.3098
```

3.2.2 staPLOT.m

Generates a state-trace plot for continuous data.

Example call:

```
>> staPLOT(delay, 'groups', {1:4, 5:8}, 'labels', ...
     {'No delay', 'Delay'}, 'axislabels', {'RB','II'});
```

Input:

- `delay` is the name of the data set and is the one required argument. It may be in either general or cell-array format, or in summary statistics form (i.e., the output of `staSTATS` described below). If the input is a data structure then `staPLOT` invokes `staSTATS` and accepts an optional covariance shrinkage parameter (also explained above). The plot includes error bars.

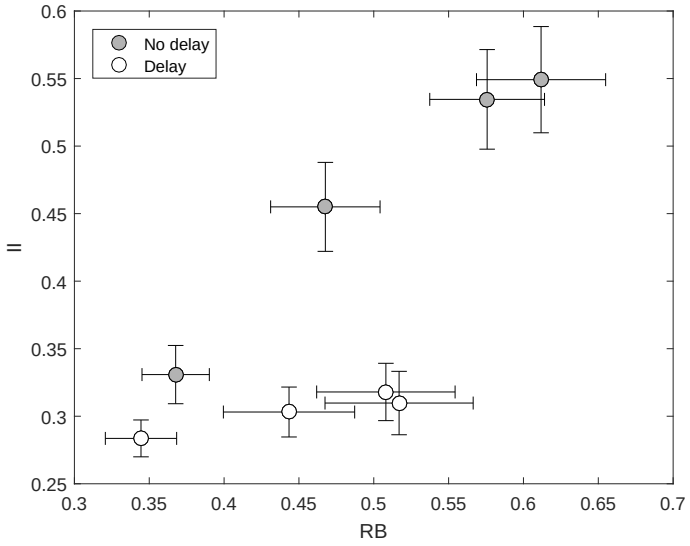

Figure 4: Output from staPLOT based on `delay` data set.

- `'groups'` identifies the conditions to be distinguished in the plot with different markers. For the delay data, conditions 1 to 4 correspond to the *no delay* group and conditions 5 to 8 correspond to the `delay` group. This is passed to `staPLOT` by setting the value of `'groups'` to the cell array, {`1:4, 5:8`}. Up to four groups can be defined in this way.

- `'labels'` is a cell array consisting of the labels for the groups defined by `'groups'` to appear in the legend. In the present example, this is specified by the cell array, {`'No delay','Delay'`}.

- `'axislabels'` is a cell array that defines the labels on the x-axis and y-axis, respectively (note: the same result can be achieved using the Matlab functions, `xlabel` and `ylabel`). In the present example, the labels are defined by the cell array, {`'RB','II'`}, for *rule based* and *information integration*, respectively.

- `'predicted'` is a cell array consisting of predicted values for each condition. Typically, this is the best-fitting values returned by `staMR` or `staCMR` (see below).

Output:

Output is shown in Figure 4.

3.2.3 staMR.m

This function conducts monotonic regression on a data structure according to a given partial order. We say it fits the partial order model to the data (i.e., the set of dependent variables).

Example call:

```
>> [x, f, shrinkage] = staMR (data, E, shrink);
```

Input:

Here, data is either a data structure (in cell array or general format) or structured output from `staSTATS`; `E` is a partial order (required) in either cell array or adjacency matrix format; `shrink` is an optional shrinkage parameter (defined previously). If data is a cell array of structured output from `staSTATS`, then the shrinkage specified by this output is used whether the argument `shrink` is specified or not.

Output:

- `x` is a n-element cell array of that contains the best-fitting values for each dependent variable
- `f` is the value of the least squares fit
- `shrinkage` is a $b \times n$ matrix of shrinkage values (where b is the number of levels of the between-participant independent variable)

Try `staMR` with `delay`. To do so, we have to specify a partial order. Use the following:

```
>> E = {1:4, 5:8, [5 1], [6 2], [7 3], [8 4]};
>> [x2, f2, shrinkage2] = staMR (delay, E);
>> disp([x2{:}]);
    0.3676    0.3308
    0.4676    0.4550
    0.5757    0.5346
    0.6118    0.5492
    0.3445    0.2830
    0.4434    0.3034
    0.5081    0.3149
    0.5169    0.3149
```

```
>> disp(f2);
    0.1721

>> disp(shrinkage2);
    0.0520    0.0491
    0.0314    0.2650
```

3.2.4 staCMR.m

This is the main function that conducts the CMR (state-trace) analysis. It takes a data structure or a cell array of structured output from `staSTATS` and an optional partial order and returns the best fitting values (to the data means) and the least squares fit. It fits the monotonic model to the data.

Example call:

```
>> [x, f, s] = staCMR (data, E, shrink);
```

On the input side, `data` is a data structure, `E` is an optional partial order, and `shrink` is an optional shrinkage parameter (defined previously).

On the output side, `x` is a cell array of the best-fitting values, `f` is the value of the least squares fit, and `s` is a structured array of fit statistics. The component, `s.shrinkage`, is a $b \times n$ matrix of shrinkage values.

Now try `staCMR` with `delay`:

```
>> [x1, f1, s1] = staCMR (delay, E);
>> disp([x1{:}]);
    0.3759    0.3150
    0.4850    0.4358
    0.5898    0.5167
    0.6265    0.5318
    0.3353    0.2856
    0.4186    0.3150
    0.4806    0.3227
    0.4850    0.3227

>> disp(f1);
    1.7493

>> disp(s1.shrinkage);
    0.0520    0.0491
    0.0314    0.2650
```

3.2.5 staMRFIT.m

This function tests the fit of the partial order model.

Example call:

```
>> [p, datafit, fits, pars]=staMRFIT(data, name-value pairs
);
```

Input:

The `staMRFIT` function takes the following arguments:

- `data` is the name of a data structure (either in general format, cell array format), or structured output from `staSTATS`. If a data structure is specified then the bootstrap re-sampling is non-parametric. If only the summary statistics are provided (e.g., `delaystats`) then the bootstrap is parametric and assumes that observations are distributed normally for each dependent variable in each condition.
- `'partial'`, `'part'`, `'p'` is a required partial order. This may be in either cell array or adjacency matrix form.
- `'nsample'`, `'ns'`, `'n'` is the number of bootstrap samples to be drawn in computing the empirical sampling distribution of the fit value.
- `'shrink'` is the optional shrink parameter, defined previously.

Output:

- `p` is the estimated p-value for the hypothesis that the fit of the model is zero. It is the proportion of bootstrap fit values that are greater than or equal to the observed fit value. Note that it will be be different from run to run, as it is a bootstrap estimate.
- `datafit` is the observed fit value. It is the same as `f1` returned by `staMR` above.
- `fits` is a vector of length `nsample` of computed bootstrap fit values. Thus, `p` is the proportion of values of `fits` that are greater than or equal to `datafit`.
- `pars` is a cell array of length n, the number of dependent variables. Each component of `pars` is an `nsample` $\times$ `ncond` matrix of means

for each bootstrap sample and each condition (where `ncond` is the number of conditions).

Applying `staMRFIT` as follows produced the following outputs:

```
>> [p, datafit, fits, pars] = staMRFIT(delay, 'partial', E, ...
'nsample', 10000);
>> disp([p, datafit])
    0.7470    0.1721
>> size(fits)
ans =
    10000     1
>> pars
pars =
    [10000x8 double]    [10000x8 double]
```

3.2.6 staCMRFIT.m

This function estimates the empirical distribution (and hence p-value) of the difference in the fit of the conjoint monotonic and the fit of the partial order model. The function call is analogous to `staMRFIT`:

`>> [p,datafit,fits,pars]=staCMRFIT(data,` *name-value pairs* `);`

Input:

- `data` is the name of the data structure (in either general or cell array format) or the name of structured output from `staSTATS`). If specified as a data structure then the bootstrap re-sampling is non-parametric, otherwise the bootstrap is parametric and assumes that observations are distributed normally for each dependent variable in each condition.
- `'partial'` is an optional partial order in either cell array or adjacency matrix form.
- `'nsample'` is the number of bootstrap samples to be drawn in computing the empirical sampling distribution of the fit value. We recommend using about 10,000 for estimating p to the nearest 100th.
- `shrink` is the optional shrink parameter, defined previously.

Output:

- `p` is the estimated p-value for the hypothesis that the difference in fit between the monotonic model and the partial order model is zero. It is the proportion of differences of bootstrap fits values that are greater than or equal to the observed difference in fit value.
- `datafit` is the observed difference in fit value between the monotonic model and the partial order model.
- `fits` is a vector of length `nsample` of computed differences in bootstrap fit values. Thus, `p` is the proportion of components of `fits` that are greater than or equal to `datafit`.
- `pars` is a cell array of length n, the number of dependent variables. Each component of `pars` is an `nsample` × `ncond` matrix of means for each bootstrap sample and each condition (where `ncond` is the number of conditions).

Applying staCMRFIT to the delay data produces the following output:

```
>> [p,datafit,fits]=staCMRFIT(delay, 'partial', E,...
'nsample', 10000);
>> disp([p datafit])
    0.1753    1.5772
>> size(fits)
ans =
    10000     1
>> pars
pars =
    [10000x8 double]    [10000x8 double]
```

4 Binary data

Binary data has the form that each observation is either a 'success' or a 'failure'. STACMR assumes that these observations have been aggregated into counts of the total number of successes and failures for each participant and each condition.

4.1 Input data structures

At present, STACMR accepts two kinds of data structure for binary data:

1. **Cell array format**. In this format, the data are contained in a $N \times n$ cell array where N is the number of observation units (typically, participants) and n is the number of dependent variables. Each component of this cell array is itself an $k \times 2$ matrix of counts of successes and failures, respectively, where k is the number of conditions.

2. **General format**. This is a fixed column format organised as a matrix in which each row corresponds to a particular observation unit (participant) and condition. Each column is defined as follows:

   column 1: Participant number

   column 2: Condition number (if none, then set this value to 1)

   column 3: Dependent variable (numbered 1, 2, and so on)

   column 4: Count of number of successes

   column 5: Count of number of failures

Data in general format are converted to cell array format using the function,

```
>> y = BNgen2cell (data);
```

4.2 Principle functions

The principle functions are located in the folder, `.../STACMR-matlab`. Their operation is illustrated with respect to the dataset, `dfie`, located in folder, `.../STACMR-matlab/Data files`.

We illustrate the binary STACMR functions using The `dfie` data set reported by Prince, M., Hawkins, G., Love, J., & Heathcote, A. (2012). An R package for state-trace analysis. *Behavior Research Methods, 44*(3), 644-655.

In this study, the dependent variables were accuracy of memory for pictures of faces and accuracy of memory for pictures of houses. There were $k = 6$ conditions defined by the combination of two factors; stimulus orientation (upright, inverted), and study duration (short, medium, and long). There were $N = 18$ participants each of whom were tested under all six conditions on both dependent variables. The data from each participant therefore can be analyzed individually.

The data (counts of hits and misses) for each participant are contained in the text file, `dfie.dat`:

```
>> dfie = load('dfie.dat'); % read general format
>> y = BNgen2cell(dfie); % convert to cell array
```

4.2.1 staSTATSBN.m

This function returns summary statistics for binary data.

Example call:

```
>> dfiestats = staSTATSBN (y);
```

The output, `dfiestats`, consists of a $N \times n$ cell array each component of which is a participant (indexed by `isub`) and a dependent variable (indexed by `dvar`). Each such component consists of the following:

- `dfiestats{isub,dvar}.count` = matrix of counts of successes and failures for each condition.
- `dfiestats{isub,dvar}.means` = vector of 'means' corresponding to the proportion of successes for each condition.
- `dfiestats{isub,dvar}.weights` = vector of weights corresponding to the number of trials (i.e., number of successes + number of failures) for each condition.
- `dfiestats{isub,dvar}.n` = vector of counts corresponding to the number of trials for each condition. Identical to weights.

4.2.2 staPLOTBN.m

Generates a state-trace plot for binary data.

Example call:

```
>> staPLOTBN(dfie, 'subj', 1, 'groups', {1:3, 4:6}, ...
 'labels', {'Upright', 'Inverted'}, 'axislabels', ...
 {'Faces','Houses'});
```

This command produces the plot shown in Figure 5.
Input:

- `dfie` is the name of the data set and is the one required argument. It may be in either general or cell-array format, or in summary statistics form (i.e., the output of `staSTATSBN`).
- `'subj'` identifies the participant number to be plotted.

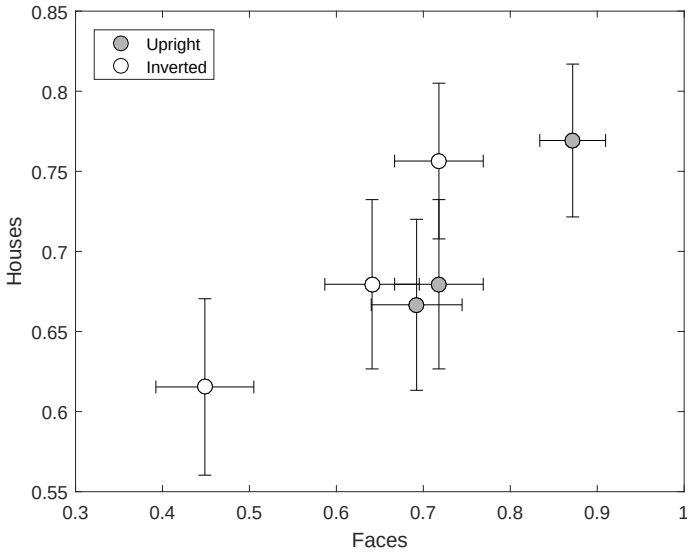

Figure 5: Output from staPLOTBN for Participant 1 from the `dfie` data set.

- `'groups'` identifies the conditions to be distinguished in the plot with different markers. For the dfie data, conditions 1 to 2 correspond to the upright orientation and conditions 4 to 6 correspond to the inverted orientation.
- `'labels'` is a cell array consisting of the labels for the groups defined by `'groups'` to appear in the legend.
- `'axislabels'` is a cell array that defines the labels on the x-axis and y-axis, respectively.
- `'predicted'` is a cell array consisting of predicted values for each condition (not shown here).

4.2.3 staMRBN.m

The function `staMRBN` fits a partial order model to a binary data structure.

Example call:

```
>> [x, f, g] = staMRBN (data, E);
```

Input

- `data` is the name of the binary data structure.

- `E` is a partial order (required) in either cell array or adjacency matrix format.

Output:

- `x` is an $N \times n$-element cell array, where N is the number of observation units (participants) and n is the number of dependent variables. Each component of `x` is a k-element vector containing the best-fitting values for the corresponding participant and dependent variable.
- `f` is a N-vector containing the values of the least squares fit for each participant.
- `g` is a N-vector containing the G^2 fit for each participant.

Execute the following commands:

```
>> E = {1:3,4:6};
>> [x1, f1, g1] = staMRBN (dfiestats, E);
```

For participant 1, the observed means are:

```
>> [dfiestats{1,1}.means, dfiestats{1,2}.means]
ans =
    0.7179    0.6795
    0.6923    0.6667
    0.8718    0.7692
    0.4487    0.6154
    0.6410    0.6795
    0.7179    0.7564
```

The fitted means are:

```
>> [x1{1,:}]
ans =
    0.7051    0.6731
    0.7051    0.6731
    0.8718    0.7692
    0.4487    0.6154
    0.6410    0.6795
    0.7179    0.7564
```

The least-squares fit and G^2 values for this participant are:

```
>> [f1(1), g1(1)]
ans =
    0.0321    0.1525
```

4.2.4 staCMRBN.m

This function fits the monotonic model to a binary data structure.
Example call:

```
>> [x, f, g] = staCMRBN (data, E);
```

Input:

- `data` is the name of a binary data structure.
- `E` is an optional partial order in either cell array or adjacency matrix format.

Output:

- `x` is an $N \times n$-element cell array, where N is the number of observation units (participants) and n is the number of dependent variables. Each component of `x` is a k-element vector containing the best-fitting values for the corresponding participant and dependent variable.
- `f` is a N-vector containing the values of the least squares fit for each participant.
- `g` is a N-vector containing the G^2 fit for each participant.

Execute the following commands:

```
>> [x2, f2, g2] = staCMRBN (dfie, E);
```

For participant 1, the fitted means are:

```
>> [x2{1,:}]
ans =
    0.7051    0.6752
    0.7051    0.6752
    0.8718    0.7692
    0.4487    0.6154
    0.6410    0.6752
    0.7179    0.7564
```

The least-squares fit and G^2 values for this participant are:

```
>> [f2(1), g2(1)]
ans =
    0.0342    0.1622
```

4.2.5 staMRFITBN.m

This function tests the fit of the partial order model to binary data.
Example call:

`>> [p, datafit, fits, pars] = staMRFITBN (data,` *name-value pairs* `);`

Input:

- `data` is the name of the binary data structure.
- `'partial'`, `'part'`, `'p'` is the partial order in cell array or adjacency matrix form.
- `'nsample'`, `'ns'`, `'n'` is the number of bootstrap samples.

Output:

- `p` is an N-vector containing the estimated p-values for each observation unit (participant).
- `datafit` is an N-vector containing the observed G^2 value for each each observation unit (participant).
- `fits` is an $N\times$ `nsample` matrix of computed bootstrap G^2 values. Each row corresponds to each of N observation units (participants) and each row consists of a vector of length `nsample`.
- `pars` is a cell array of size $N \times n$, where n is the number of dependent variables. Each component of `pars` is an `nsample` $\times$ `ncond` matrix of means for each bootstrap sample and each condition (where `ncond` is the number of conditions).

Example output (note that `p` and `fits` will be differ from run-to-run as they are bootstrap estimates):

```
>> [p, datafit, fits, pars] = staMRFITBN (dfie, 'partial', E, ...
 'nsample', 10000);
>> disp([p, datafit])
   0.7361    0.1525
>> size(fits)
ans =
    18    10000
```

```
>> size(pars)
ans =
    18     2
>> size(pars{1,1})
ans =
    10000     6
```

4.2.6 staCMRFITBN.m

This function tests the difference between the fit of the partial order model and the fit of the monotonic model to binary data.
Example call:

`>> [p, datafit, fits, pars] = staCMRFITBN (data,` *name-value pairs* `);`

Input:

- `data` is the name of the binary data structure.
- `'partial'`, `'part'`, `'p'` is an optional partial order in cell array or adjacency matrix form.
- `'nsample'`, `'ns'`, `'n'` is the number of bootstrap samples.

Output:

- `p` is an N-vector containing the estimated p-values for each observation unit (participant).
- `datafit` is an N-vector containing the observed G^2 value for each each observation unit (participant).
- `fits` is an $N\times$ `nsample` matrix of computed bootstrap G^2 values. Each row corresponds to each of N observation units (participants) and each row consists of a vector of length `nsample`.
- `pars` is a cell array of size $N \times n$, where n is the number of dependent variables. Each component of `pars` is an `nsample` $\times$ `ncond` matrix of means for each bootstrap sample and each condition (where `ncond` is the number of conditions).

Example output (note that `p` and `fits` will be differ from run-to-run as they are bootstrap estimates):

```
>> [p, datafit, fits] = staCMRFITBN (dfie, 'p', E,...
'n', 10000);
>> disp([p, datafit])
   0.8207    0.0098
>> size(fits)
ans =
    18    10000
>> size(pars)
ans =
    18     2
>> size(pars{1,1})
ans =
    10000     6
```