STACMRx

STACMRx is a set of Matlab functions that conduct CMR analyses of state-trace data. CMR stands for “coupled monotonic regression”. This is a quick guide to its use.

***STACMRx.zip***

Unzip this file in a convenient folder. Make sure this folder and its subfolders are on the Matlab path.

***Java***

The current implementation of STACMRx calls code written in java. While this speeds up the analysis to a very great extent, it does require (at this point) some extra housekeeping. This is itemized below:

1. Java is usually pre-installed in Matlab. To check, type:  
     
   >> version –java  
   Any version on or after version 1.7 is good. If java is not installed, you will have to do this yourself from <http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html>.
2. The appropriate CMR java runtime library also needs to be acquired and stored in a convenient directory. The current version is called fxMR-0.3.18.jar.
3. Before running any of the STACMRx programs, the CMR java runtime library needs to be made available to Matlab. This requires the command:  
     
   javaclasspath(path);  
   where path is the full pathname of the library. An example (omitting stuff in the middle) is:  
   javaclasspath('C:\Users\LAPTOP\Documents\...\fxMR-0.3.18.jar');

***Input data structures***

The basic idea is that we have a set of data (called a *data structure*) that is organised as a set of observations on *d* > 1 *dependent variables* across levels of two or more *independent variables*. From the point of view of *state-trace analysis* (STA), the number of different independent variables is not important – rather, it is the total number of conditions (combinations of levels). For example, if there are two independent variables, one with 2 levels, the other with 3 levels, then there will be a total of 2 x 3 = 6 conditions. These conditions will be numbered from 1 to 6.

For this reason, STACMRx distinguishes only between conditions defined between-subjects and conditions defined within-subjects. If there are *b* between-subjects conditions and *w* within-subjects conditions then there is a total of *bw* conditions ordered by the set of between-subjects conditions. That is, the first *w* conditions correspond to the set of within-subjects conditions under the first between-subjects condition, the next *w* conditions correspond to the set of within-subjects conditions under the second between-subjects condition, and so on.

At present, STACMRx accepts two kinds of data structure:

1. **Cell array format**. In this format, the data are organised in a *b* x *d* cell array where *b* is the number of between-subjects groups and *d* is the number of dependent variables. Each component of this cell array is itself an *n* x *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-subjects conditions (fixed across groups and dependent variables). The dependent variable may be either within-subjects or between-subjects – it doesn’t matter because the correlation between dependent variables is assumed to be zero (although this might be up for grabs in future implementations).
2. **General format**. This structure is useful if the data are already in some kind of fixed column format. It is organised as a matrix in which each row corresponds to an observation and each column is defined as follows:  
   column 1 = subject number (for identification only, not used directly)   
   column 2 = between-subjects condition or group (if none, then column 2 = 1)  
   column 3 = dependent variable (1 or 2)  
   columns 4 to end = values for each within-subjects condition  
     
   While STACMRx accepts data in general format it always converts it to cell array format using the function,   
   >> y = gen2cell (data);

***Partial order***

As well as data, the various STACMRx 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 *xi* < *xj*.

A partial order is represented in STACMRx in two (equivalent) ways:

1. As a cell array containing the set of (*i*, *j*) pairs, e.g., {[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 a 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 of nodes or points that the partial order applies to. It is almost invariably the sequence of numbers, 1 to *n*, where, in this case, *n* = 4. The function adj2cell converts an adjacency matrix into its corresponding cell array.

***Models***

Although originally developed for STA, the main functions have been recently extended to conduct *signed difference analysis* (SDA) introduced by Dunn, J. C. & James, R. N. (2003). Signed difference analysis: Theory and application. *Journal of Mathematical Psychology, 47*(4), 389-416. This is further elaborated by Anderson, L. & Dunn, J. C. (in preparation). *Signed Difference Analysis: Testing for structure under monotonicity*.

For users only interested in state-trace analysis, the only thing to remember is that this is the default option for the relevant functions and if a function requires specification of a model as an argument, then it is sufficient to enter a null matrix, i.e., [].

***The principal functions***

The operation of the principal functions will be illustrated with respect to two data sets.

The first, called delay, is a data set discussed 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. It is 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

***staSTATS.m***

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

Example call:

>> delaystats = staSTATS (y, shrink);

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-subjects 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.

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

Thus, delaystats has two elements. If you look at the means, they 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

***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 *d*-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* x *d* matrix of shrinkage values (where *b* is the number of levels of the between-subject independent variable).

Try this 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, s2] = 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(s2);

0.0520 0.0491

0.0314 0.2650

***staCMRx.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. We say it fits the *conjoint partial order and monotonic model* to the data.

Example call:

>> [x, f, shrinkage] = staCMRx (data, model, E, shrink);

On the input side, data is the data structure, model is an optional model (explained above), E is an optional partial order, 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* x *d* matrix of shrinkage values.

Now try this with delay:

>> [x1, f1, s1] = staCMRx (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

***staCMRFIT.m***

This function estimates the empirical distribution (and hence *p*-value) of the fit of the conjoint monotonic and partial order model against the fit of the partial order model.

Example call:

>> [p, datafit, fits] = staCMRFIT (nsample, data, model, E, shrink, proc);

*Input:*

* nsample is the number of Monte-Carlo samples (about 10,000 should be good)
* data is a data structure (which can be in general format, cell array format, or structured output from staSTATS). If data is non-summary data in general or cell array format then the bootstrap resampling is non-parametric, otherwise it is parametric (assumes a normal distribution).
* model is an optional model (default = STA)
* E is an optional partial order (default = none)
* shrink is an optional shrinkage parameter (default = -1)
* proc is an optional specification of maximum number of processors to be used (default = maximum available)

*Output:*

* p is the estimated *p*-value
* datafit is the observed fit value (see below)
* fits is a vector of empirical fit values (see below)

**Important note:**

staCMRFIT operates on the difference in fit values between a partial order model (calculated using staMR.m, e.g., f2 above) and the combined partial order + monotonic model (calculated using staCMR.m, e.g., f1 above).

Example output (note that p and fits will be approximate only):

>> [p, datafit, fits] = staCMRFIT (10000, delay, [], E);

>> disp([p, datafit])

0.4538 1.5772

>> histogram (fits, 100)

****

***staPLOT.m***

This function generates a state-trace plot of a data structure.

Example call:

>> staPLOT (data, *optional argument pairs*);

Here, data is a data structure or a cell array of structured output from staSTATS. The optional argument pairs are of the form; *‘keyword’*, *value*. The following are the set of keywords (valid alternatives in parentheses);

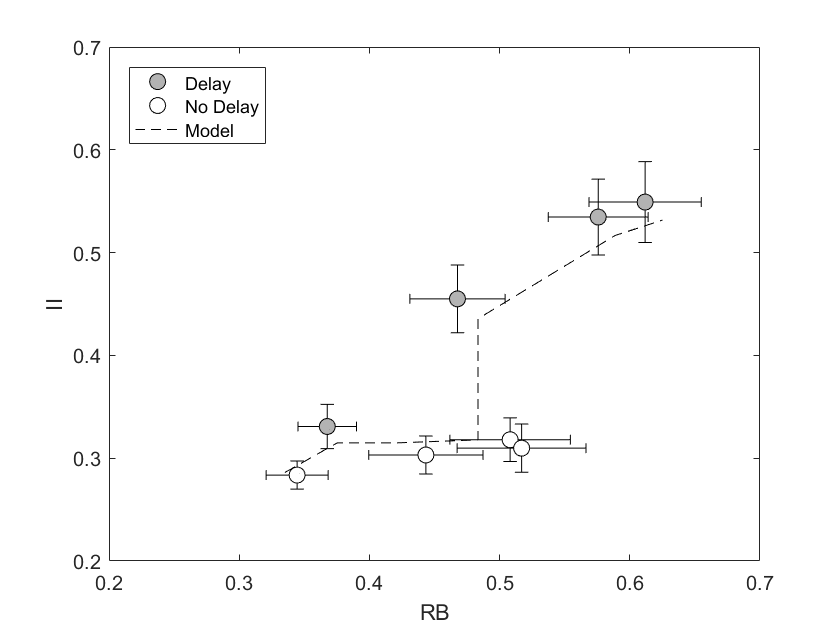
* predictions (prediction, pred, p) is a *d*-element cell array that defines a set of fitted values (output from staMR or staCMRx)
* vars (v, dv) is a 2-vector that defines two dependent variables for plotting (default = [1, 2] )
* groups (group, g) is a cell array that defines a structure on the conditions. For example if there are 2 between-subjects conditions and 3 within subjects conditions then one group structure might be {1:3, 4:6}. At present, only one factor can be identified in this way.
* labels (label, lab, l) is a cell array that defines the labels of the levels defined by groups
* axislabels (axislabel, axislab, axis) is a cell array that defines the labels of the x- and y-axes
* axislimits (axislimit, axislim, limits, limit, lim) is a cell array that defines the upper and lower limits of the axes
* axisticks (axistick, ticks, tick, t) is a cell array that defines tick marks for x- and y-axes (if specified then axislimits is redundant)
* location (loc) is a Matlab value for the location of the legend (default = ‘northwest’)
* color (col, c) is a 3-element RGB vector specifying the color of the plot of predicted values (default is black)
* line is a flag to indicate if the predicted values are connected by a line. If set a line is drawn (default) otherwise no line is drawn.

Example output:

To see how this works, we use delay and the results of the earlier call to staCMRx.

>> staPLOT(delay, 'pred', x1, 'groups', {1:4 5:8}, 'labels', {'Delay' 'No Delay'}, 'axislabels', {'RB' 'II'}, 'axisticks', {.2:.1:.7, .2:.1:.7});

This should produce the following graph:



23 November 2016

John Dunn