StateTrace: An R Package for State-Trace Analysis

Melissa Prince, Guy Hawkins,
Jonathon Love & Andrew Heathcote
School of Psychology, The University of Newcastle, Australia

Abstract

This introduction to **StateTrace**, a freely available GUI driven package for the R language, is an extended version of Prince, Hawkins, Love and Heathcote (2011), which provided an abridged description of the application of this package.

State-trace analysis (Bamber, 1979) is a graphical approach that can determine whether one or more than one latent variable mediates an apparent dissociation between the effects of two experimental manipulations. State-trace analysis makes only ordinal assumptions, and so is not confounded by range effects that plague alternative methods, especially when performance is measured on a bounded scale, such as accuracy. **StateTrace** automates many aspects of a state-trace analysis of accuracy and other binary response data, including implementing Bayesian methods quantifying evidence about the outcomes of a state-trace experiment and the creation of customisable graphs.

Keywords: state-trace analysis, dimensional analysis, latent variables, **StateTrace**

1. Introduction

One of the most fundamental questions asked in experimental psychology and the neurosciences concerns whether a single latent (i.e., not directly observable) variable mediates the relationship between two or more experimental factors. For example, is recognition memory mediated by one (*memory strength*) or two (*familiarity* and *recollection*) processes (Dunn, 2004, 2008)? Researchers often seek particular patterns of data, called *dissociations*, in order to answer this question (Shallice, 1988) and so to infer the existence of functionally independent neural (e.g., Teuber, 1955) or cognitive (e.g., Glanzer & Cunitz, 1986) systems. Typically a one-dimensional or one-system account is rejected when a dissociation is observed, as quantified by a significant interaction test.

However, it has been repeatedly shown that dissociations quantified by an interaction are problematic. This is because they cannot compel the rejection of a one-dimensional explanation without making strong and debateable assumptions that are difficult, if not impossible to test (e.g., Bogartz, 1976; Busemeyer & Jones, 1983; Dunn, 2003; Dunn & Kirsner, 1988; Henson, 2006; Poldrack, 2006; Loftus, 1978, 1996). For example, when the response measure is bounded (e.g., accuracy), an observed interaction, or equally the failure to observe an interaction, might be scale dependent (e.g., confounded by floor and ceiling effects) if the function mapping the latent variable to the bounded response scale is nonlinear (Loftus, 1978; see also Prince, Brown & Heathcote, 2011).

State-trace analysis (Bamber, 1979), which is also known as *dimensional analysis* (Loftus, Oberg & Dillon, 2004), avoids the problems that plague traditional dissociation methods (see Newell & Dunn, 2008, for a recent summary). It does so by making only the weak and arguably plausible assumption that latent variables have a *monotonic* effect on performance; that is, that the direction of the latent variable's effect does not change with its magnitude. Latent dimensionality can be assessed through a simple graph: a plot of one response measure against another. This *state-trace plot* must be monotonic if both measures are mediated by the same latent variable. In contrast, if the plot is not monotonic then more than one latent variable must be in play.

Although state-trace analysis has been applied to a diverse range of topics and areas (see Prince, Brown & Heathcote, 2011, for an exhaustive summary), no textbooks cover relevant methodological issues, such as the need to calibrate state-trace designs in order to produce data diagnostic of dimensionality. Furthermore, standard statistical methods cannot be used to make inferences about dimensionality without requiring additional assumptions that detract from the relatively general and assumption-free nature of state-trace analysis. Prince, Brown and Heathcote (2011) addressed both gaps. First, they identified methodological issues specific to state-trace experiments and recommended an iterative process of design refinement to avoid them. Second, they developed Bayesian procedures for binary data analysis that not only quantifies evidence about dimensionality but also aids in the process of design refinement. These statistical procedures add only the assumption that a binomial process generates the binary data. In this paper (which is an extended version of Prince, Hawkins, Love & Heathcote, 2011), we provide a detailed tutorial of how to use **StateTrace**, a freely available GUI driven package for the R language (R Development Core Team, 2007), which implements Prince, Brown and Heathcote's (2001) approach in a way that automates the computationally demanding aspects of the analysis.

2. State-Trace Analysis

To illustrate state-trace analysis, we use the example data included in the **StateTrace** package, which concerns recognition memory judgements indicating whether a test item was previously studied (*old*) or not (*new*). In this experiment, items were studied one at a time and then tested as pairs of items, one of which was old and one of which was new (i.e., a *two-alternative forced choice*, or *2AFC*, judgement). The aim of this experiment was to determine whether faces are encoded on an extra latent dimension (commonly called *configural* encoding; Maurer, LeGrand & Mondloch, 2002), that is not available to non-face stimuli. Here we used houses as the non-face stimuli as they match faces on characteristics such as being familiar, complex and mono-oriented (i.e., they have a specific *upright* orientation).

Like conventional dissociation analyses, state-trace analysis is primarily concerned with the interaction between two experimental factors, which we designate the *state* and *dimension* factors. Our example and software address the common case where both factors have two levels and come from a completely repeated-measures design. A state-trace plot is created by plotting dependent variable measurements from one level of the state factor against measurements from the other level. In our example the dependent measurements (recognition accuracy) differ only in the type of test items that elicited them, pictures of faces or pictures of houses.

The dimension factor in our example concerns whether items are presented upright or inverted. It has been proposed that upright faces can be encoded both in terms of their constituent features (i.e., a feature dimension) and in terms of the configuration of those features (i.e., a configural dimension), whereas non-face stimuli and inverted faces can only be encoded on the feature dimension (Rakover, 2002). This proposal is based on the robust empirical finding that inversion disproportionately affects recognition performance for faces when compared to non-face stimuli; known as the *Disproportionate Face Inversion Effect* (*DFIE*; Valentine, 1988; Yin, 1969). Therefore, the manipulation of orientation potentially influences dimensionality differentially for each level of the state factor (i.e., for houses and faces), leading to our dimension factor designation. In our 2AFC experiment, items could be studied and tested either upright or inverted; for each participant an item was always studied and tested in the same orientation and both members of a test pair had the same orientation.

With two exceptions, a single latent variable can be responsible for generating the data in a state-trace plot *if and only if* the orders of points on each axis are exactly the same as, or exact opposite of, each other. Graphically this is equivalent to being able to join all points with a curve that never decreases or never increases. The first exception concerns plots with only two points (as would be produced by our current 2(state: face, house) x 2(dimension: upright, inverted) design), where the condition on orders necessarily holds; that is, two points can always be joined by a curve that always increases or always decreases. Hence a state-trace plot must have at least three points to be diagnostic of dimensionality. This can be achieved by a dimension factor with more than two levels, but more commonly a third *trace* factor enters the design. In our example, the trace factor manipulates the time allowed to study each item (i.e., study duration). Our software and analyses can accommodate any number of trace-factor levels, but we recommend at least three and not many more (see Prince, Brown & Heathcote, 2011, for reasons).

To describe the second exception it is convenient to refer to lines joining points which have different trace-factor levels but share the same dimension-factor levels, as *data traces*: when data traces fail to overlap, it is possible for the data to fall on a single monotonic curve even if multiple dimensions do exist. Hence the data traces must overlap on at least one axis for the state-trace plot to be diagnostic of dimensionality. Typically, performance is better for one level of the dimension factor than the other, in which case overlap can be achieved by counteracting the difference using a non-factorial trace-factor manipulation. The second exception, therefore, provides the main reason why state-trace experiments and conventional fully factorial experiments have different requirements: In our example, upright performance is better than inverted performance for all mono-oriented stimuli (i.e., the *inversion* effect; Rock, 1974), and so we used longer study durations for inverted (200, 600, and 1800ms) than upright (66, 200, and 600ms) items. Greater overlap is better, so an iterative process may be required to select the best set of trace-factor levels for each dimension factor level.

Although not required for valid inference about dimensionality, the trace factor must have a monotonic effect on the dependent measure in order for non-monotonicity in a state-trace plot (and hence the need for more than one latent variable) to be unambiguously attributed to the interaction between the state and dimension factors. In some cases a monotonic trace-factor effect can be assumed based on prior evidence (e.g., increased study time typically always leads to increased accuracy). Alternatively, examining the monotonicity of points in each data trace provides a direct check. The chances of passing this check can be increased by choosing increments in trace-factor levels that lead to substantial and equal increments in performance, which, like ensuring data-trace overlap, may require iterative design refinement. For example, the durations for our 2AFC experiment increased by a constant multiple and were selected based on prior evidence that recognition memory accuracy increases linearly with log study time (e.g., Loftus et al., 2004). Prince, Brown and Heathcote's (2011) analysis supports both types of design refinement by providing methods that quantify evidence about data-trace overlap and a monotonic trace-factor effect.

To summarise the design of our example data, the state factor defining the axes of the state-trace plot corresponds to the type of stimulus presented (faces or houses). The dimension factor, which has the potential to differentially influence the underlying latent variable(s), corresponds to the orientation (upright or inverted) of the study presentation. The trace factor, which creates variation in the data points within each level of the dimension factor, corresponds to the study duration. Finally, lines joining sets of points from the same dimension-factor level are called data traces.

2.1. Quantifying Evidence

Although graphically simple, the visual inspection of state-trace plots can sometimes be misleading due to measurement noise. This is particularly the case for the analysis of individual participant data where levels of measurement noise can be high. However, individual analysis is required to make strong inferences based on state-trace analysis because neither the monotonicity nor non-monotonicity of state-trace plots is necessarily preserved when averaged over participants (see Prince, Brown & Heathcote, 2011, for examples). Hence a quantitative approach to state-trace analysis provides an important complement to visually assessing the state-trace plot.

StateTrace provides inference about individual participant experimental results by quantifying evidence about four mutually exclusive models:

1) *Non-trace*: The trace factor does not always have a monotonic effect (i.e., one or more data traces are non-monotonic) and hence non-monotonicity in the state-trace plot cannot be unambiguously attributed to the interaction between the state and dimension factors.

2) *No-overlap*: Data traces do not overlap (i.e., even though the state-trace plot is monotonic no conclusions can be made about dimensionality).

3) *Uni-dimensional*: Performance is mediated by a single latent variable (i.e., the state-trace plot is monotonic and provides a valid basis for inference about dimensionality).

4) *Multi-dimensional:* Performance is mediated by more than one latent variable (i.e., the state-trace plot is non-monotonic).

Each of these models specifies a set of order restrictions on the points in a state-trace plot. Evidence for each model is quantified using a Bayes factor (BF; Kass & Raftery, 1995). The BF quantifies the change in relative beliefs (odds) about two models, which is caused by observing the data. That is, it is the change from prior odds (the odds before seeing the data) to posterior odds (the odds after seeing the data). For example, if two models, M1 (the numerator model) and M2 (the denominator model), are initially considered equally likely, then BF = 10 implies M1 is ten times more likely than M2 after observing the data.

Formally, the BF is the ratio of the average likelihood of each model, where the average takes into account uncertainty about the true values of the model's parameters and the model's flexibility (i.e., its ability to fit any data pattern). Bayes factors can be thought of as measures of relative goodness-of-fit that compensate for differences in model flexibility. In the current context the denominator (M2) is always an *encompassing* model under which all orders are equally likely. Hence, a BF > 1 favours the less flexible order-restricted (numerator) model over the encompassing (denominator) model, which by definition fits any data perfectly. Conventions vary, but evidence for the numerator model can be considered weak for BF < 3, positive when BF > 3, strong when BF > 20 and very strong when BF > 100.

If it is assumed that one model amongst a set is the model that generated the data (i.e., the *true* model) Bayes factors for the set of models can be combined to calculate posterior model probabilities, p, which quantify evidence about each model in the set in a relative manner (as the set of probabilities sum to one). High probabilities provide evidence favouring a model and low probabilities against. Table 1 provides some conventions aiding interpretation. Note these conventions are similar to those suggested for the Bayes factors as p = BF / (1 + BF), given BF = 1 for the encompassing model. If the true-model assumption is not made, a posterior probability still provides a number on an easy to interpret zero-to-one scale that quantifies the relative evidence for a model within a set of models; however, that number can no longer be interpreted as a probability.

Table 1: Conventions to aid interpretation of the posterior model probabilities (after Raftery, 1995)

Favouring model		Against model
$p > .99$	Very strong evidence	$p < .01$
$.95 < p \leq .99$	Strong evidence	$.01 \leq p < .05$
$.75 < p \leq .95$	Positive evidence	$.05 \leq p < .25$
	$.25 \leq p \leq .75$ Equivocal evidence	

Note also that models can be excluded from the set under consideration when previous evidence is deemed strong enough to reject them without examining the current experimental data. For example, trace-factor manipulations are usually selected based on prior evidence that they have a monotonic effect. In this *trace-true* case the non-trace model can be excluded from the set, allowing the relative evidence for each of the remaining three models to be compared. **StateTrace** can output both Bayes factors, to quantify absolute evidence about whether the data provide sufficient support for clear conclusions about each model, as well as posterior model probabilities, to quantify the relative evidence for models that are considered plausible a priori.

2.2. Computing Bayes Factors

Prince, Brown and Heathcote (2011) used posterior sampling methods proposed by Klugkist, Kato and Hoijtink (2005) and Klugkist, Laudy and Hoijtink (2005) to compute Bayes factors for the four models. Although conceptually straightforward (i.e., they simply count the frequency with which different orders occur), these methods are computationally expensive. In order to make them practical Prince, Brown and Heathcote used two types of sampling:

1) Faster Monte-Carlo (MC) sampling, used to obtain samples from a model that makes no order constraints, which we refer to as *encompassing sampling*, and

2) Slower Markov-Chain Monte-Carlo (MCMC) sampling, used to obtain samples under the trace model constraint, which we call *trace model sampling*.

In principle MC methods could be used for all required sampling, but the yield of samples relevant to models (2) – (4) is often so low that this would be far too inefficient to be used in practice. Although slower per sample, the MCMC method always yields relevant samples after an initial “burn-in” period and so in practice is a better option.

As described in the next section of this paper, **StateTrace** manages the process of computation, enabling estimates to be automatically refined to a specified level of accuracy and for computation to be limited to convenient time periods (e.g., overnight runs). Users may output posterior model probabilities that instantiate two model-selection strategies, either simultaneous selection among all four models (an *exhaustive strategy*) or a *trace-true strategy* that excludes the non-trace model a priori. Raw counts can also be accessed to support other strategies, such as sequential model selection (see Prince, Brown & Heathcote, 2011, for details).

3. Using StateTrace: An Example Analysis

The **StateTrace** package runs under R, a free software environment for statistical computing and graphics (R Development Core Team, 2007). R is available for Windows, Mac OS X and Linux, can be downloaded from http://www.r-project.org/. When R expects an input command, it issues the > prompt. Experienced R users can execute the functions instantiated in **StateTrace** from the command line, call them from their own functions and modify and incorporate the code in their own functions as required, subject to the requirements of the software license agreement. Here we describe how to use **StateTrace** through a GUI (guided user interface) suitable for users less experienced with R, allowing them to view default values of function arguments and enter alternative values via widgets, such as text entry boxes, slider bars, true/false check boxes, and multi-option lists.

Hoffman and Laird’s (2009) **fgui** package provides the GUI functionality and is available from the package repository on the R homepage. To obtain the **fgui** package, select ‘CRAN’ from the menu on the left of the R homepage and choose the appropriate CRAN mirror for your location.

Select “Packages” from the menu on the left, search for “fgui” and download the corresponding package or binary file for your operating system. This process should also be repeated to obtain Plummer, Best, Cowles and Vine’s (2006) *coda* package. **StateTrace**, *fgui* and *coda* are installed once into an instance of R by typing

```
> install.packages("fgui","coda","StateTrace")
```

on the R command line (assuming package files are located in the working directory). Subsequently, each time R is invoked **StateTrace** functionality is made available by typing

```
> library("StateTrace")
```

On some operating systems equivalent menu based methods can be used for both of these steps; for example, for Windows users a package can be installed by selecting *‘Packages > Install package(s) from local zip files...’* from the menu toolbar and then navigating to and selecting the downloaded file, while the functionality of **StateTrace** can be made available by selecting *‘Packages > Load Package...’* from the menu toolbar and selecting “StateTrace” from the displayed list.

The main **StateTrace** GUI (see Figure 1) can be invoked by typing

```
> guista()
```

and provides access to each of the **StateTrace** functions: **stFirst**, **stSample**, **stSummary**, **stProbplot**, **stBootav**, **stPlot** and **staManage**. Clicking the corresponding function buttons will open further GUI windows. Alternatively, these GUIs can be called directly by typing “gui” followed by the function name and parentheses (e.g., `guistFirst()`). In general terms, command line users can remove “gui” from the start of the function and enter argument values within the parentheses (e.g., `stFirst(staname="DFIE.sta", fnams="DFIE.txt", multiparticipant=T)`). Details on the available arguments can be obtained by consulting the function’s help documentation, which is called by typing a `?` prior to the function name on the command line (e.g., `?guistFirst` or `?stFirst`).

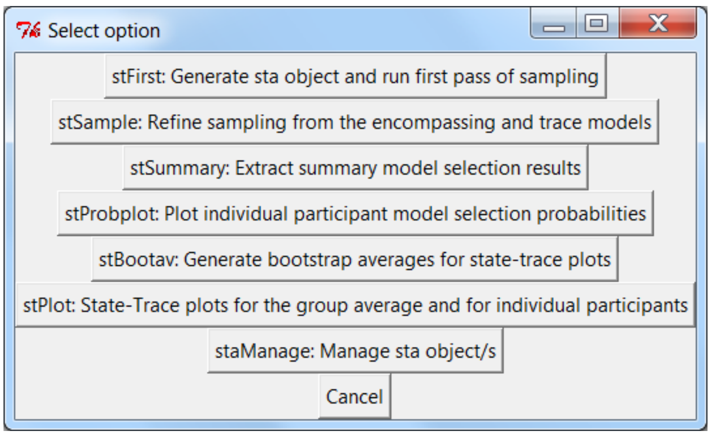

Select option
stFirst: Generate sta object and run first pass of sampling
stSample: Refine sampling from the encompassing and trace models
stSummary: Extract summary model selection results
stProbplot: Plot individual participant model selection probabilities
stBootav: Generate bootstrap averages for state-trace plots
stPlot: State-Trace plots for the group average and for individual participants
staManage: Manage sta object/s
Cancel

Figure 1: GUI window for `guista()`, which contains a button linking to a customised GUI for the seven functions available in **StateTrace**.

3.1. Overview of Functions

The **stFirst** function performs an initial analysis, first reading in data for one or more participants from one or more text files, and then making a quick preliminary assessment of the results based on a limited number of posterior samples. It then creates an object of class *sta* in the R environment, which is named by the user. The *sta* object is used to encapsulate data and the numerical results produced by posterior sampling as well as analyses of the posterior samples. Once **stFirst** is complete the *sta* object can be saved from the R environment to a file in a compressed format and then restored in a later R session using the R `save` and `load` functions respectively.

Three **StateTrace** functions allow the contents of an *sta* object to be displayed. The **stSummary** and **stProbplot** functions provide, respectively, tabular and graphical summaries of the Bayesian analysis. While the **stPlot** function makes state-trace plots. All three functions will work to some degree with an *sta* object just created by **stFirst**. However, for more accurate results from the Bayesian analysis, and to access the full range of state-trace plot options, two other functions may have to be run: **stSample** and **stBootav**. These functions perform time-consuming computations whose results are stored in the *sta* object. The **stSample** function collects enough extra posterior samples to reach a specified level of accuracy in the Bayesian analysis. Some of these extra samples can also be stored in the *sta* object so that a line representing the trace or monotonic model that is best supported by the data can be added to a state-trace plot. Additionally, the **stBootav** function must also be run to enable lines representing trace and monotonic models to be added to state-trace plots averaged over participants.

The main GUI also provides access to the **staManage** function, which allows users to manage and export posterior sample stored in an *sta* object. An *sta* object is an R list that can be directly accessed by users, but **staManage**, and other functions, are designed so this should not be necessary. Samples are stored in an *sta* object to enable efficient generation of graphical summaries of uncertainty in estimation (i.e., credible regions, the Bayesian equivalent analogue of confidence intervals). However, this can sometimes cause an *sta* object to become so large that it takes a long time to load and save. Hence, it can be necessary to use **staManage** to remove samples after the graphics have been generated. The **staManage** function also allows users to export a list containing posterior samples with one entry for each participant. Each entry in the list contains samples in a format (the `mcmc.list` class) for which **coda** provides many easy to use analysis methods (e.g., `plot` and `summary` functions).

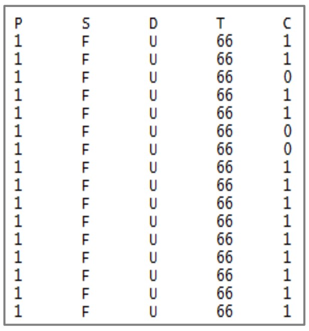
3.2. Data Input Formats

StateTrace reads data from text files in two formats, both of which (a) can contain data from either one or more participants, (b) have the same number of columns in each row, (c) can contain a header row and (d) have initial columns containing numbers or character strings indexing the design cell referred to by that row. They differ in that:

1. *Trial data files* have a row for each trial ending with a binary response indicator (e.g., correct and error responses coded as 1 and 0 respectively)

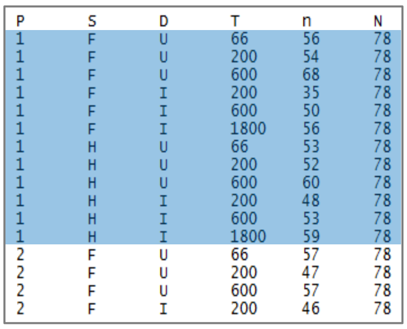
2. *Summary data files* have a row for each design cell ending with the summed binary response frequencies and number of trials for each cell.

Figures 2a and 2b show our example 2AFC data in trial and summary formats respectively. For both formats, the first column (P) contains identifiers unique to each participant, which may either be a numeric (e.g., $1 : n$) or character string (e.g., “MP”, “GH”) value. Where the file has data for only a single participant, this column is used to assign a participant identifier or it may be omitted. When omitted participants are given integer identifiers in the order files are read; however, file names are also stored and can be displayed later using the **stSummary** function. The next three columns contain indicators for the levels of the state (S), dimension (D) and trace (T) factors. Again these levels may be specified as numeric values (e.g., 1:3 for a trace factor defined by three study duration levels) or as character values (e.g., “F” and “H” for a state factor defined by the type of stimulus presented, faces and houses). The only restriction for these indicators being that the trace-factor identifiers within each combination of state and dimension levels must sort (using R’s `sort()` function) into the order assumed by the trace model to produce increasing performance.



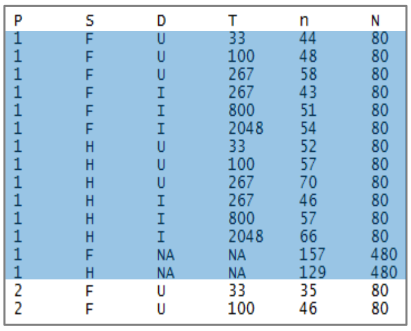
P	S	D	T	C
1	F	U	66	1
1	F	U	66	1
1	F	U	66	0
1	F	U	66	1
1	F	U	66	1
1	F	U	66	0
1	F	U	66	0
1	F	U	66	1
1	F	U	66	1
1	F	U	66	1
1	F	U	66	1
1	F	U	66	1
1	F	U	66	1
1	F	U	66	1
1	F	U	66	1
1	F	U	66	1

(a)



P	S	D	T	n	N
1	F	U	66	56	78
1	F	U	200	54	78
1	F	U	600	68	78
1	F	I	200	35	78
1	F	I	600	50	78
1	F	I	1800	56	78
1	H	U	66	53	78
1	H	U	200	52	78
1	H	U	600	60	78
1	H	I	200	48	78
1	H	I	600	53	78
1	H	I	1800	59	78
2	F	U	66	57	78
2	F	U	200	47	78
2	F	U	600	57	78
2	F	I	200	46	78

(b)



P	S	D	T	n	N
1	F	U	33	44	80
1	F	U	100	48	80
1	F	U	267	58	80
1	F	I	267	43	80
1	F	I	800	51	80
1	F	I	2048	54	80
1	H	U	33	52	80
1	H	U	100	57	80
1	H	U	267	70	80
1	H	I	267	46	80
1	H	I	800	57	80
1	H	I	2048	66	80
1	F	NA	NA	157	480
1	H	NA	NA	129	480
2	F	U	33	35	80
2	F	U	100	46	80

(c)

Figure 2: Example 2AFC data files in (a) trial and (b) summary formats and (c) yes-no data in summary format. In (b) and (c), rows corresponding to one participant’s data have been highlighted.

Each of the two formats has a different structure for the remaining columns, with a total of six columns in summary files and five columns in trial files. For the trial format the final column (C) contains the response made to each trial. For example, for 2AFC judgements (e.g., Figure 2a) this column records whether the response was correct (1) or an error (0). In contrast, if the experiment tested single items and therefore required a *yes-no* judgement, this column records a yes (1) or no (0) response choice. For the summary format, the fifth column contains the number of “successes” (n) per design cell. This corresponds to the summed number of correct identifications for 2AFC data (e.g., Figure 2b). For yes-no data this column contains the summed frequency for making a “yes” response (e.g., Figure 2c). In both cases, the sixth column for summary files contains the total number of observations (N) per design cell. For example, in our 2AFC experiment participants attended three 1hour sessions over which they completed 52 blocks of 18 study images and 18 test pairs, yielding 78 observations per design cell, as shown in Figure 2b.

For our 2AFC example data in summary format (Figure 2b), there is one row for each of the 2(state: face, house) x 2(dimension: upright, inverted) x 3(trace: study duration) = 12 design cells per participant. However, the summary yes-no data (Figure 2c) has an additional two rows per participant corresponding to the baseline conditions for each level of the state factor. In this experiment, images of faces and houses were presented either upright or inverted at study and then all tested upright in a yes-no recognition task. As a yes-no task was used, participants were required to respond to new items during the test phase and it is the false alarm rates (i.e., incorrect "yes" judgements; FAR) for the new faces and new houses recorded in the extra rows.[1] Therefore, the yes-no summary data has a total of 2 x 2 x 3 + 2 = 14 rows per participant. This latter type of design is referred to by Prince, Brown and Heathcote (2011) as a *state baseline*, or *B2*, design. While the former type is referred to as a *no baseline*, or *B0*, design. As the baseline conditions do not have corresponding dimension or trace factor levels, these indicators can be left blank or specified as "NA". **StateTrace** uses the presence of such indicators to automatically distinguish B0 from B2 designs.

Note that no explicit provision is made for designs where accuracy in all conditions is measured relative to a single baseline. Such data can be treated as coming from either a B0 or B2 design. In the former case, the baseline data can be omitted (e.g., only the "yes" data is analysed). In the latter case, the baseline data are included twice. The outcome of state-trace analysis is the same in both cases, but the B2 treatment may be preferred as it enables accuracy to be displayed in graphs in terms of a difference between non-baseline and baseline results.

The data for our 2AFC example can be made available with a

```
> data(DFIE)
```

command, which will create an object called 'DFIE' in the R workspace and can be viewed using

```
> fix(DFIE)
```

To demonstrate **StateTrace**'s data input capabilities, this data needs to be saved in a text file outside of the R environment using

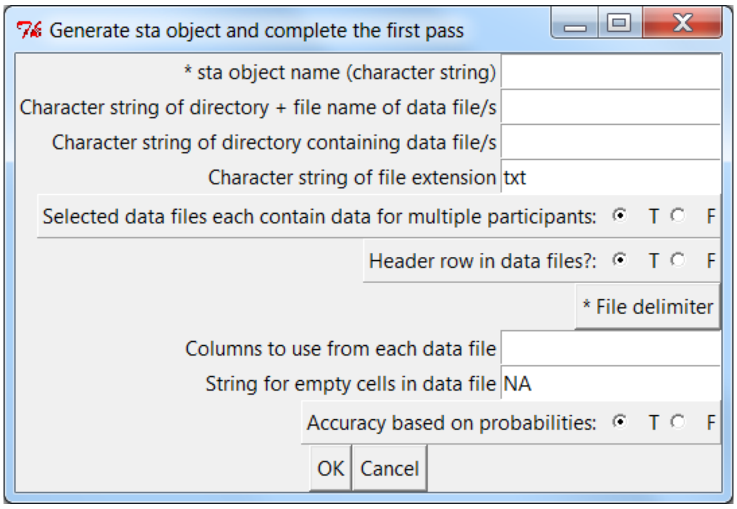
```
> write.table(DFIE, file="DFIE.txt", sep="\t", row.names=F)
```

which will save the data in a tab-delimited text file called 'DFIE.txt' in the current working directory. Note that the location of the current working directory can be obtained by typing `getwd()` on the command line. If the working directory needs to be changed, this can be done with a `setwd()` command and including the path of the desired directory within the parentheses (e.g., `setwd("C:/User/Desktop/statetrace")`). Alternatively, on some operating systems the working directory can be changed by selecting '*File > Change dir...*' from the R menu toolbar and navigating to the desired directory.

[1] Note that unambiguous inference about dimensionality for yes-no responses requires that accuracy for both levels of the dimension factor be assessed against a common baseline (i.e., relative to the same new items). This is achieved in our yes-no example as all items were tested upright.

3.3. First Steps

It is important to note that multiple passes of sampling from the data will likely be required, which will update user specified aspects of the analysis. The **stFirst** function therefore provides users with a function specifically designed for commencing the sampling process. **stFirst** is a wrapper for several other functions that are called with some arguments fixed at values minimally sufficient for an initial analysis. The GUI shown in Figure 3 illustrates the remaining arguments that can be set by the user. For this function (and all other GUIs presented) parameter values that must be set are marked by a '*'. Note also that in all GUIs, parameter descriptions containing the term "character string" must have their entries enclosed in quotes. Chief among these is the *sta object name*, which must not begin with a number or contain any spaces; the **stFirst** function will create an *sta* object with this name in the R environment, overwriting any existing object without a warning. For this example analysis, we assigned the name "DFIE.sta". It is not required for the name of this object to be followed by the extension '.sta' however, we tend to use this naming convention for ease in distinguishing the *sta* object from other formats that may have the same name and be saved in the current workspace.


Generate sta object and complete the first pass

* sta object name (character string)	
Character string of directory + file name of data file/s	
Character string of directory containing data file/s	
Character string of file extension	txt
Selected data files each contain data for multiple participants:	[✓ T] F
Header row in data files?:	[✓ T] F
* File delimiter	
Columns to use from each data file	
String for empty cells in data file	NA
Accuracy based on probabilities:	[✓ T] F

OK Cancel

Figure 3: GUI for **stFirst**, which is used to read-in the raw data files, create an *sta* object, perform an initial pass of sampling, and create preliminary state-trace plots.

The following three **stFirst** arguments specify different ways of loading data. The first argument, *Character string of directory + file name of data file/s*, allows one or more directory and file-name combinations to be entered as a character vector. If no directory is entered then R's working directory is assumed. Otherwise, the R convention of a forward slash, /, should be used when specifying the path of a file; for example, `"datafiles/DFIE.txt"` specifies a single file in the directory 'datafiles' below the working directory, or `c("DFIE-P1.txt", "DFIE-P2.txt")` specifies two files in the working directory. Alternatively, the next two arguments enable loading of all files contained in a specified directory (*Character string of directory containing data file/s*) that have a particular extension (*Character string of file extension*; by default *.txt). If all three of these arguments are left blank when **stFirst** is run, R's `choose.files()` method of selection (through a file list dialog) is invoked for Windows and Mac users. Using this method, Windows users can specify multiple files by holding the 'Ctrl' key when selecting the data files, whereas Mac users may only select one file (note this method is not available on Linux).

The next five arguments specify the data file format. First, *Selected data files each contain data for multiple participants?*, indicates whether the data file(s) contain data for multiple participants (“T”) or a single participant (“F”). It can also be specified whether the data files contain a header row by checking either true (“T”) or false (“F”) for the argument *Header row in data files?* The next button, *File delimiter,* is used to indicate what delimits the columns. This value can be specified by clicking the button and selecting an option (“tab”, “space” or “comma”) from the displayed list.

Data files will also often contain more information than required by **StateTrace** and therefore the argument, *Columns to use from each data file,* specifies the relevant columns in the order of participant identifier (if included), state, dimension, trace and response columns either by indicating the column position (as an integer vector) or column name (as a character vector). This can be done using R’s `combine` function: `c(…)`, where … are the column numbers or names to be included; for example, `c(1,2,4,7)` could be used for a trial data file that does not have a participant identifier column but does have additional non-relevant columns to be excluded, or `c(3,1,2,4)` could be used for a trial data file with no participant identifier column that had the state, dimension and trace columns in the incorrect order. The next argument, *String for empty cells in data file/s*, allows any non-relevant rows to be excluded from analysis depending on the contents of the last (*C*) or second last (*n*) columns for trial and summary formats respectively. For example, in a recognition memory task participants may be required to make study-trial responses that are recorded on separate rows to the test-trial responses, or participants may fail to make a test response on some trials. Such rows can be excluded by specifying the character or symbol that identifies them (by default “NA”).

At the end of the initial pass, **stFirst** will generate a state-trace plot for each individual participant and for the group average. The final argument, *Accuracy based on z transformation?*, specifies whether accuracy measures based on probabilities (“F”) or on the inverse cumulative normal (*z*) transformation of probabilities (“T”) should be plotted. For the former option the state-trace plot will contain an estimate of the proportion of correct responses for B0 designs and the hit rate (HR; the proportion of correct ‘yes’ responses) minus the false alarm rate (FAR; the proportion of incorrect ‘yes’ responses) for B2 designs. Although state-trace analysis largely avoids the scale dependent caveats that can confound bounded response measures, some users may still wish to normalise their data. The latter option, therefore, specifies *z*(proportion correct) for B0 designs and the signal detection theory measure $d' = z(\text{HR}) - z(\text{FAR})$ for B2 designs.

For this example analysis, most of the **stFirst** arguments can remain at their default values. However, we assigned “DFIE.sta” as the *sta* object name, entered the file name “DFIE.txt” for the second argument value (assuming that the text file is located in the working directory) and specified that a probability measure should be used when generating the state-trace plot (i.e., the proportion correct as the example data comes from a B0 design). Clicking the ‘OK’ button with then run **stFirst**.

Before sampling begins, a number of checks are run to ensure the selected data files are compatible with **StateTrace** (e.g., that the state and dimension factors each have only two levels) and that multiple data files are compatible with each other (e.g., all have either a B0 or B2 design). If any of the files are incompatible, an error message will direct the user to the aspect of the data file(s) that produced that problem. For example, if one participant's data had three trace levels and another had four trace levels the message "Participants have different numbers of trace levels" would be printed.

Using the raw data, the initial pass will then create the *sta* object. This object is essentially a list of lists, which are empty by default but will hold information about the raw data (e.g., whether the design is a B0 or B2, as well as the number of and expected order for the trace levels) as well as the sample counts and prior and posterior estimates (see `?staMake` for a detailed breakdown of the *sta* object). **StateTrace** will then complete two "runs" per participants sampling from the encompassing posterior. This sampling is done for each independent set of design cells; four sets for B0 designs corresponding to each combination of the state by dimension factor levels and two sets for B2 designs as there is a baseline condition effectively tying the dimension levels together for each state level.

For each run, **StateTrace** draws S = 100,000 samples from the encompassing model assuming no orders and counts the number of these samples that respect the trace model order s_T. It then calculates the posterior proportion for the trace model (T) relative to the unrestricted (U) encompassing model using $\hat{\Pi}_{T,U} = (s_T + 1)/(S + 2)$, estimates the 95% credible interval for the posterior estimates and where the precision of the interval is less than 0.0005 marks sampling for a set as being complete. Otherwise it estimates the time required to get enough samples to narrow the interval sufficiently, based on the time taken for the initial 100,000 samples and using a line search between 100,000 and 10^{14} samples. Note the time estimates are only approximate and will vary if sampling is completed on a different computer.

Next an order constrained Gibbs sampler (Gelfland, Smith & Lee, 1992) is used to draw two sequences of 5,000 MCMC samples ("chains") from the trace model per participant. For each chain S_T = 5,000 samples are drawn from the trace posterior and **StateTrace** counts the number of these samples that have a monotonic order, s_M and that are non-overlapping, s_{NO}. The proportion of trace samples following the no-overlap, $\hat{\Pi}_{NO,T} = (s_{NO} + 1)/(S_T + 2)$, uni-dimensional, $\hat{\Pi}_{UD,T} = (s_M - s_{NO} + 1)/(S_T + 2)$, and multidimensional, $\hat{\Pi}_{MD,T} = (S_T - s_M + 1)/(S_T + 2)$, model orders are tabulated and the corresponding 95% credible intervals calculated. Next the precision of these intervals is assessed, with sampling for a participant marked as complete if all intervals are less than 0.005 and the additional time required estimated otherwise. Note that a smaller interval criterion is used for the encompassing than trace sampling as the encompassing proportion estimates have greater potential to reduce precision overall because they multiple the trace proportions in the calculation of Bayes factors.

Although all of the above sampling is essentially happening "behind the scenes" **StateTrace** prints a number of details in the R console, which allows the progress of the sampling to be monitored. First the data sources of each data file loaded are reported:

```
> guistFirst()
[1] "stFirst"
Close ' Generate sta object and complete the first pass ' window to allow entering
commands in the R console;   that window has gone modal. Note that the ' Generate
sta object and complete the first pass ' window may be   hidden from view (esp. in
windows), and you may have to find it in the taskbar and close it.

Reading in data file DFIE.txt
```

If the data files are compatible and compiled to create the *sta* object, a summary of the design is then reported:

```
Object DFIE.sta has state levels F,H dimension levels I,U and 3 trace levels, with
baseline design B0
```

As seen above, this summary notes that our DFIE example, has “F” and “H” (i.e., faces and houses) as the state factor levels, “I” and “U” (i.e., inverted and upright) are the dimension factor levels, there are three levels for the trace factor and the data has a B0 design.

Next updates are printed relating to the sampling from the encompassing model including (a) the current precision of the credible interval of the trace posterior proportion for each independent set as well as the (b) estimated time remaining (in minutes) for each participant:

```
UPDATING REMAINING TIME FOR ENCOMPASSING MODEL SAMPLING (minutes)
Participant 1 (Chain CI-Size: 1=0.0046 2=0.006 3=0.0059 4=0.006 ): 0.18
Participant 2 (Chain CI-Size: 1=0.0059 2=0.0023 3=0.0062 4=0.0055 ): 0.11
Participant 3 (Chain CI-Size: 1=0.0053 2=0.006 3=0.0055 4=0.0054 ): 0.14
        .                                              .
        .                                              .
        .                                              .
Participant 17 (Chain CI-Size: 1=0.0059 2=0.0052 3=0.0043 4=0.0059 ): 0.11
Participant 18 (Chain CI-Size: 1=0.006 2=0.0058 3=0.0055 4=0.0062 ): 0.13
TOTAL TIME REMAINING FOR ENCOMPASSING SAMPLING: 2.31

WORKING ON ENCOMPASSING MODEL FOR REMAINING PARTICIPANTS
Participant 1 (Chain CI-Size: 1=0.0032 2=0.0042 3=0.0042 4=0.0042 ): 0.2
Participant 2 (Chain CI-Size: 1=0.0042 2=0.0016 3=0.0044 4=0.0039 ): 0.13
Participant 3 (Chain CI-Size: 1=0.0038 2=0.0042 3=0.0039 4=0.0038 ): 0.17
        .                                              .
        .                                              .
        .                                              .
Participant 17 (Chain CI-Size: 1=0.0042 2=0.0037 3=0.0031 4=0.0042 ): 0.15
Participant 18 (Chain CI-Size: 1=0.0042 2=0.0041 3=0.0039 4=0.0044 ): 0.17
```

It should be noted that when sampling from the encompassing posterior the time remaining relates to the appropriate computation time required for the set with the “worst” precision to meet the specified criterion. For example, for participant 1 above, the time remaining of 0.18minutes at the end of the first run relates to the third set which has the widest credible interval. These details are printed for each individual participant, followed by an estimate of the total time remaining at the end of the first run (here 2.31minutes) and then the individual participant updates are printed for the second run of sampling from the encompassing posterior.

Similar updates are then printed for the MCMC sampling from the trace posterior including (a) the current precision of the credible intervals for the no-overlap, uni-dimensional and multidimensional posterior proportions and (b) the estimated time remaining for all three estimates to satisfy the required precision criterion:

```
UPDATING REMAINING TIME FOR TRACE MODEL SAMPLING (minutes)
Participant 1(CI-Size: NoOverlap=0.0031 1D=0.0135 MD=0.0138): 0.35
Participant 2(CI-Size: NoOverlap=0.0043 1D=0.0129 MD=0.0135): 0.35
Participant 3(CI-Size: NoOverlap=7e-04 1D=0.0068 MD=0.0068): 0.35
        .                                               .
        .                                               .
        .                                               .
Participant 17(CI-Size: NoOverlap=7e-04 1D=0.0137 MD=0.0137): 0.35
Participant 18(CI-Size: NoOverlap=0.0011 1D=0.004 MD=0.004): 0
TOTAL TIME REMAINING FOR TRACE SAMPLING: 5.62

WORKING ON TRACE MODEL FOR REMAINING PARTICIPANTS
Participant 1(CI-Size: NoOverlap=0.0025 1D=0.0094 MD=0.0097): 0.46
Participant 2(CI-Size: NoOverlap=0.0031 1D=0.0092 MD=0.0096): 0.46
Participant 3(CI-Size: NoOverlap=4e-04 1D=0.0048 MD=0.0048): 0
        .                                               .
        .                                               .
        .                                               .
Participant 17(CI-Size: NoOverlap=4e-04 1D=0.0098 MD=0.0098): 0.46
Participant 18(CI-Size: NoOverlap=5e-04 1D=0.0023 MD=0.0023): 0
```

It can therefore be seen that at the end of the second run participant 1 also requires a further 0.35minutes of computation time for the three estimates to satisfy the precision criterion. Finally, at the end of the second run of trace sampling, an estimate is reported for the overall time remaining for sampling from the encompassing posterior, the trace posterior as well as all sampling:

```
TOTAL TIME REMAINING FOR ENCOMPASSING SAMPLING (minutes): 2.96
TOTAL TIME REMAINING FOR TRACE SAMPLING (minutes): 5.53
TOTAL TIME REMAINING FOR ALL SAMPLING (minutes): 8.49
```

For the trace model sampling, initial (burn-in) samples are discarded because it can take some time for the MCMC process to converge to the target distribution (see Gilks, Richardson & Spiegelhalter, 1996). Our experience is that for Prince, Brown and Heathcote's (2011) method, convergence is very fast and that at most 100 initial samples need to be discarded. However, this may not be the case in all applications, so **StateTrace** provides facilities to check. Once the two chains sampling from the trace model have run, **stFirst** automatically calculates one check for whether the MCMC process has worked properly (i.e., has "converged") using Gelman's multivariate "R-hat" statistic as provided by Plummer et al.'s (2006) **coda** package:

```
Gelman Diagnostic for Convergence (multivariate)
   1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16
1.00 1.00 1.00 1.01 1.00 1.00 1.00 1.00 1.01 1.00 1.01 1.00 1.00 1.01 1.00 1.01
  17   18
1.00 1.00
```

This statistic requires more than one chain therefore, to ensure convergence can always be assessed at the end of **stFirst** for all participants, this first pass will force two runs from the trace posterior for all participants (*cf.* sampling from the encompassing posterior, where the second run is only completed for those participants where sampling is not complete). As seen above, the first row of the output corresponds to the participant number and the second row is Gelman's R-hat statistic, where values close to one indicate good convergence.

In the final stage of computation **stFirst** draws 10,000 bootstrap average samples and uses the two-dimensional density estimator provided by Wand and Ripley's (2009) **KernSmooth** package (with its default parameters) to calculate the posterior modes (measures of central tendency) and 68% credible regions around the modes:

```
Calculating bootstrap average over participants:
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18

Calculating encompassing bootstrap average
```

These calculations are used to display an average state-trace plot, which provides the user with an immediate view of results averaged over participants, as well as state-trace plots for each individual participant. Additionally the corresponding posterior mode estimates are output in a tabular form to the R console; for example:

```
Participant 10 posterior mode
p(posterior region)=0.68

, , State = F  (HR)

     Dimension
Trace         I         U
    1 0.6407435 0.5386162
    2 0.4630981 0.6809448
    3 0.6132080 0.6370516

, , State = H  (HR)

     Dimension
Trace         I         U
    1 0.5523747 0.4844325
    2 0.5285221 0.5976829
    3 0.6480587 0.5489388

 Average
0.577806
```

Once the first pass has completed, **stFirst** suggests directions for the next stage of analysis:

```
First pass completed!
 Run stSample to obtain accurate posterior model probability estimates
 then examine results with stSummary and stProbplot.
 Run stBootav to obtain participant results for average trace and
 monotonic plots with stPlot
 Use staManage to join sta objects and manage stored samples
```

Note also that **stFirst** can be called repeatedly to add additional participants to the *sta* object; it should just be ensured that the argument *sta object name* is provided with the name of the existing object to which the additional data should be appended to. A warning will be issued if duplicate data sources are mistakenly specified however, this data will still be added to the object without replacing the old entries.

3.4. Saving and Loading *sta* Objects

Before running any subsequent passes, the user may wish to save the *sta* object to the current stage of analysis. This can be done by typing

```
> save.image()
```

which will save the *sta* object (and everything else in the environment) in the current R workspace. However, there may be times where it is more appropriate to save the *sta* object outside the R workspace. This can be done using the `save()` command by providing the name of the object to be saved and assigning a file name within the parentheses. For example, typing

```
> save(DFIE.sta, file="DFIE.sta")
```

will save the example *sta* object as a compressed file called "DFIE.sta" in the working directory. Note this command behaves in much the same way as 'zipping' a file and so can also be used to save multiple *sta* objects to a single external file (e.g., `save(c(DFIE1.sta, DFIE2.sta), file="sta objects")`).

When saved as an external file, the `load()` command can be used to load the *sta* objects into any R workspace. If the file has been copied to the new working directory, then simply including the file name within the parentheses will direct R to the file to load; for example

```
> load("DFIE.sta")
```

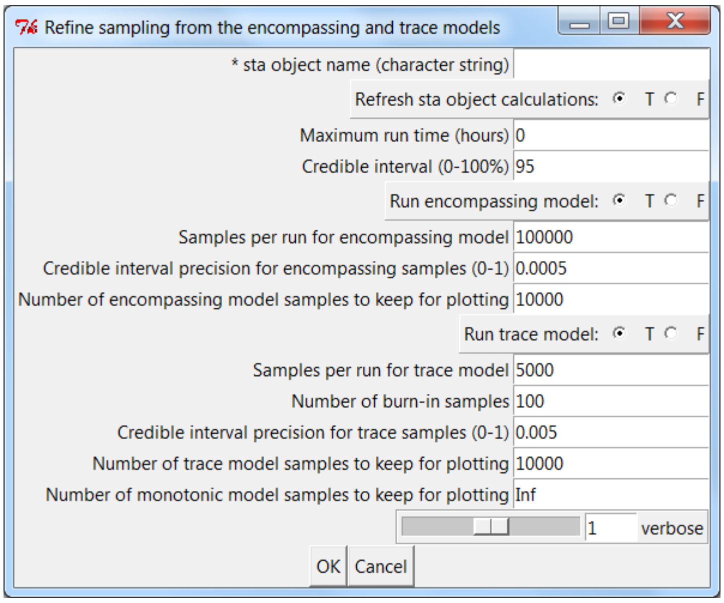
Similarly, `load("sta objects")` will 'unzip' and load both of the *sta* objects. Alternatively, if the file is not located in the working directory, typing

```
> load(choose.files())
```

will open a file list dialog from which the user can navigate to and select the file to load.

3.5. Refining Estimates

Further sampling may be required if the analysis did not reach completion during the initial pass or the user wishes to alter the credible interval and precision criteria from those used by **stFirst**. This is achieved using **stSample** (see Figure 4), which allows fine-grained control over the defaults used by **stFirst**. The only required input (* *sta object name(character string)*) is the name of an existing *sta* object (e.g., "DFIE.sta"). Because obtaining enough samples to fulfil stricter criteria can be time consuming, **stSample** has a 'refresh' mode (*Refresh sta object calculations = T*), which allows the predicted time to completion to be calculated for different criteria; that is, it will re-assess the observed credible interval and corresponding precision against the new criteria and then estimates how much more computation time is required. This refresh mode is fast to run as no actual sampling is done, unless none has been done yet, in which case a single pass is completed to get the necessary timing information. Note that as this timing information will vary depending on the computer used, if the most recent pass (e.g., running **stFirst**) was run on a different computer it is useful to turn off the refresh mode but leave the maximum run time at zero, which will cause a single pass to be run and update the timing information for the new computer.



Refine sampling from the encompassing and trace models

* sta object name (character string)	
Refresh sta object calculations:	[✓ T] F
Maximum run time (hours)	0
Credible interval (0-100%)	95
Run encompassing model:	[✓ T] F
Samples per run for encompassing model	100000
Credible interval precision for encompassing samples (0-1)	0.0005
Number of encompassing model samples to keep for plotting	10000
Run trace model:	[✓ T] F
Samples per run for trace model	5000
Number of burn-in samples	100
Credible interval precision for trace samples (0-1)	0.005
Number of trace model samples to keep for plotting	10000
Number of monotonic model samples to keep for plotting	Inf
verbose	1

Figure 4: GUI for the **stSample** function, which is used to refine the posterior estimates and sampling parameters used.

Once a sampling plan is determined the refresh mode can be turned off, a suitable maximum run time entered and sampling initiated. The *Maximum run time (hours)* can be set to any feasible number of hours by entering an integer value in the text box. As noted above, by leaving this argument at the default ‘0’ value, **stSample** will complete a single pass of sampling per participant. Alternatively, the timing estimated provided by **stFirst** can be used to inform this parameter value. For example, the output from the initial pass of the DFIE data estimated that a further 8.49minutes of computation time was required; hence the maximum run time could be set to 0.14hours. However, given that this value is only an estimate that may fluctuate as the sampling progresses, it is typically wiser to set a larger value if it is desired for the sampling to run to completion (when there are a large number of participants we have found overnight runs to be a good solution; i.e., maximum run time = 8hours).

stSample will divide the maximum time allowed between the types of sampling (from the encompassing and/or trace model) and then further between each participant that has not been completed. Note however, that sampling will terminate when the precision criteria are satisfied, and so **stSample** may complete before the maximum run time has elapsed and one type of sampling may complete before the other. The progress of this sampling can be monitored using the final **stSample** argument (*verbose*), which controls information printed to the R console during sampling. The slider for this argument can either be dragged along the width of the bar or a numeric value (0, 1, 2) entered in the adjacent text box: 0 is silent, 1 prints the estimated total time remaining after each run for all participants and 2 adds estimated timings per participant (the **stFirst** output is provided by a verbose value of 2).

Sampling is completed in a series of runs, and users may choose to sample only from the encompassing model (*Run encompassing model*), the trace model (*Run trace model*) or both. The number of samples for each run of each type of sampling (*Samples per run for encompassing model* and *Samples per run for trace model*) is chosen to satisfy a trade-off between optimising computational speed and obtaining a sufficient number of samples to count those respecting a model's order. Although fast, small values inherit a small cost in housekeeping between runs and initial *burn-in* samples on each trace model run are lost (*Number of burn-in samples*). In contrast, large values cost more in memory and can result in more samples being taken than is necessary to achieve the required precision. A larger value is advisable for encompassing than trace model sampling, as encompassing sampling is usually an order of magnitude faster. In our applications we have found the defaults work well (100,000 samples per run for the encompassing model and 5,000 per run for the trace model sampling with 100 burn-in samples) and that little is gained in particular cases by altering them. Nevertheless, these values can be modified by entering a different integer value in the appropriate text box.

Similarly the accuracy criteria (*Credible interval (0-100%)*, as well as *Credible interval precision for encompassing samples (0-1)* and *Credible interval precision for trace samples (0-1)*) can be set by altering the integer value in the appropriate text box. Again we have found the default criteria (95% credible interval and precision of 0.0005 for encompassing samples and 0.005 for trace samples) strike an appropriate balance between computation time and the accuracy of Bayes factor and posterior model probability estimates.

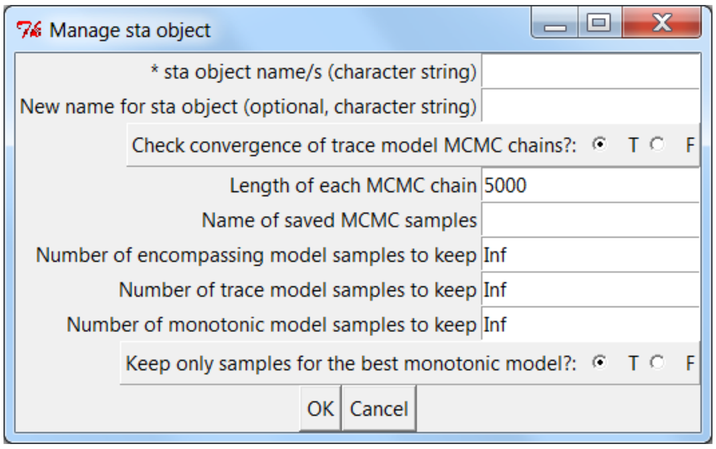
A second reason for running **stSample** is to collect samples that can be used in visualising each model; that is, samples that follow the order(s) dictated by a model. We have found the default value of 10,000 encompassing samples (*Number of encompassing model samples to keep for plotting*), which were collected by **stFirst**, are sufficient for accurate visualisation of central tendencies and credible regions, although large regions (which require estimation of distribution tails) can require more. The 10,000 trace samples (*Number of trace model samples to keep for plotting*) collected by **stFirst** are also usually more than sufficient given they are only used to estimate central tendency. The **stSample** function also collects a particular type of trace sample, monotonic model samples (*Number of monotonic model samples to keep for plotting*), which may be relatively rare, especially when the data are far from monotonic. Monotonic samples are used to plot the central tendencies of the uni-dimensional or no-overlap models, with the latter type of sample often being extremely rare unless the data are strongly non-overlapping. Given this, by default **stSample** keeps all (i.e., `Inf`) monotonic samples.

As for **stFirst** once the desired parameters have been set, clicking 'OK' will initiate **stSample**; for the current example all defaults were used except the refresh mode was turned off and the maximum run time set to '8' (although sampling was complete within 20minutes). This process of running a subsequent pass can be repeated as many times as is necessary. When all sampling is complete, the overall time remaining for both the encompassing and trace models will be recorded as '0'.

3.6. Managing the *sta* Object

Storing large sets of samples for each participant can greatly increase the size of *sta* objects. The **stSample** defaults (assuming stored monotonic samples are not allowed to grow too large) do not cause problems, but if an object contains data from a large number of participants, issues may arise, such as very slow loading and saving times for *sta* objects. The **staManage** function (see Figure 5) can be used to reduce the number of stored samples in such cases (*Number of encom-*

passing model samples to keep, *Number of trace model samples to keep* and *Number of monotonic model samples to keep*). It also includes the option to keep only samples for the "best" (i.e., most frequently occurring, and hence most probable) monotonic order rather than all samples with monotonic orders (*Keep only samples for the best monotonic model?*).


Manage sta object

* sta object name/s (character string)	
New name for sta object (optional, character string)	
Check convergence of trace model MCMC chains?:	[✓ T] F
Length of each MCMC chain	5000
Name of saved MCMC samples	
Number of encompassing model samples to keep	Inf
Number of trace model samples to keep	Inf
Number of monotonic model samples to keep	Inf
Keep only samples for the best monotonic model?:	[✓ T] F

Figure 5: GUI for the **staManage** function, which can be used to combine *sta* objects, check the convergence of MCMC trace chains and reduce the number of stored samples.

As **stSample** can add extra stored samples **staManage** also allows for Gelman's multivariate convergence analysis to be run on the expanded sets (*Check convergence of trace model MCMC chains?*). However, as **stSample** also provides the option of modifying the number of trace samples drawn per chain, the *Length of each MCMC chain* must be specified; this value must be some multiple of the total number of trace samples and result in there being at least two chains. **staManage** also allows the expanded sample sets to be exported as a list of `mcmc.list` objects[2] by specifying a name for the object (*Name of saved MCMC samples;* e.g., "DFIE.mcmc"). Exporting this object allows the user to run further convergence checks provided by Plummer et al.'s (2006) **coda** package (see the package documentation for details). For example, visual inspection of the chain convergence can be assessed using

```
> plot(DFIE.mcmc[[1]])
```

which will plot the results for the first participant. If the chains have converged then this plot should look like a "fat hairy caterpillar" and the separate chains should not be distinguishable from each other (see also `summary(DFIE.mcmc[[1]])` to obtain additional summary statistics from **coda**).

Finally, **staManage** can be used to bind multiple *sta* objects. In contrast to the previous functions the required input of an *sta* object name (** sta object name/s (character string)*) may either specify a single character name, in which case the stored samples are managed, or a vector of character names, in which case the objects are joined and saved to a new object. For example, it can be

[2] The `mcmc.list` format requires equal chain lengths for all of the chains it contains. This may not be the case if **stSample** is not run with the default value for *Samples per run of the trace model* used by **stFirst** (i.e., 5,000). In this case it is best to use **staManage** to remove the trace-model samples stored by **stFirst** before running **stSample** with new parameter values.

computationally efficient to divide a very large sample and run the sampling for sub-groups of participants on separate machines. Once completed these *sta* objects can be combined to examine the group aggregate results. By default, the combined object is saved back into the first element of the ** sta object name/s (character string)* value. However, a new object will be created if an alternate character name is entered for the *New name for sta object (optional, character string)* argument.

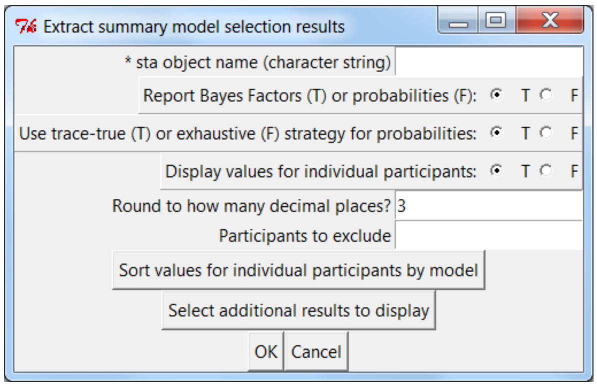
3.7. Extracting Results

The **stSummary** and **stProbplot** functions display model selection results, and the **stPlot** function creates state-trace plots. The **stSummary** function also provides information about the status of sampling for an *sta* object (i.e., whether it is complete or if not how much more computation time is required according to the criteria stored in the object). All three functions may be run as soon as **stFirst** is complete, but **stSample** should also be run to completion when final results are required. The **stBootav** function must also be run in order to calculate averages based on samples stored by **stSample** before making participant-average state-trace plots. The **stFirst** function runs **stBootav** on encompassing model samples, but **stBootav** must be run separately to plot participant averages for other models.

3.7.1. Model Selection Results

The **stSummary** GUI (Figure 6) controls output of tabular model-selection results to the R console. The only required argument for this function is the name of an *sta* object (** sta object name (character string)*). Results can be output in terms of Bayes factors or posterior model probabilities using the first true/false argument, *Report Bayes Factors (T) or probabilities (F)*. For the latter option, the next argument, *Use Trace-true (T) or Exhaustive (F) strategy for probabilities*, can specify if the probabilities are calculated based on all four models (i.e., an *exhaustive* selection strategy) or by excluding the non-trace model from the set (i.e., the *trace-true* strategy). By default, **stSummary** reports results summarised over participants based on group Bayes factors, which are the product of each participants' Bayes factors and assume each participant contributes independent evidence (see Prince, Brown & Heathcote, 2011). Note these group results are not obtained by averaging data over participants, as Prince, Brown and Heathcote showed that neither monotonicity nor non-monotonicity are necessarily preserved by averaging.

However, in some cases the group results can be inappropriately influenced by outlying individual participant results (e.g., most participants are uni-dimensional but a few strongly multi-dimensional participants dominate the group results). We therefore, recommend that users always output individual participant results in order to check this possibility using the *Display values for individual participants* option supplied in the **stSummary** GUI. When outliers occur the corresponding participant's integer or character identifier can be entered for the argument *Participants to exclude*, to exclude them from the calculation of group results; for example, to exclude the third and seventh participant we would type `3 7` in the text box. For readability the number of decimal places printed (*Round to how many decimal places?*) and different ways of sorting individual participant results (based on results for a particular model; *Sort values for individual participants by model*) can also be specified.


Extract summary model selection results

* sta object name (character string)

Report Bayes Factors (T) or probabilities (F): [✓ T] F

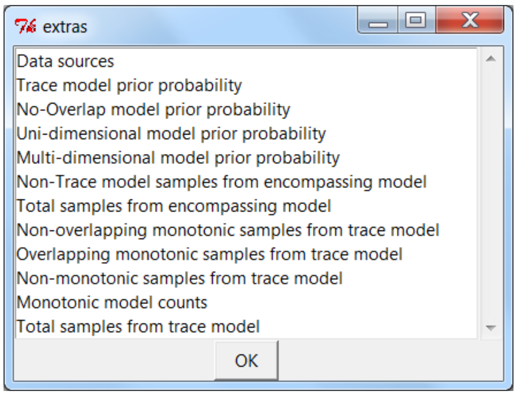
Use trace-true (T) or exhaustive (F) strategy for probabilities: [✓ T] F

Display values for individual participants: [✓ T] F

Round to how many decimal places? 3

Participants to exclude

Sort values for individual participants by model
Select additional results to display
OK
Cancel


extras

- Data sources
- Trace model prior probability
- No-Overlap model prior probability
- Uni-dimensional model prior probability
- Multi-dimensional model prior probability
- Non-Trace model samples from encompassing model
- Total samples from encompassing model
- Non-overlapping monotonic samples from trace model
- Overlapping monotonic samples from trace model
- Non-monotonic samples from trace model
- Monotonic model counts
- Total samples from trace model

(a) (b)

Figure 6: (a) GUI for the **stSummary** function and further customisations that are available by clicking the (b) “Select additional results to display” button.

Users may also output a large range of additional results, using a multi-option list GUI (Figure 6b) that opens after pressing the *Select additional results to display* button shown in Figure 6a. When each participant’s data were read from a separate file the “Data Sources” option outputs the file names, which can be useful in linking this information to the participant identifiers used by **StateTrace**. The remaining options output prior probabilities for each model (calculated analytically; see Prince, Brown & Heathcote, 2011), as well as the total number of encompassing and trace model samples and raw counts of the number of times the orders specified by each model were sampled. The latter results can be used to instantiate different model selection strategies without obtaining new samples (see Prince, Brown & Heathcote for details; a brief example is also provided below).

When the desired parameters have been set, clicking ‘OK’ will execute the **stSummary** function and the appropriate output will be printed to the R console. For our example analysis, we asked **stSummary** to output posterior probabilities using the exhaustive model selection strategy, and to include the individual participant results. We also asked for **stSummary** to output the prior probability for the trace model as well as the total number of samples from the encompassing model and the number of these samples that respected the order of the non-trace model.

The **stSummary** output first prints whether the sampling is complete and the accuracy criteria used, as well as the number of participants included in the group level results. Below we can see that for our DFIE example sampling is complete using a 95% credible interval and precision of 0.0005 and 0.005 for encompassing and trace model sampling respectively. Moreover, all 18 participants have been included in the group results:

```
Sampling complete at 95% credible interval with precision based on:
Encompassing samples=5e-04 and Trace samples=0.005

Number of participants=18
```

If however, sampling were incomplete, **stSummary** would print the accuracy criteria specified as well as the estimated computation time remaining.

Next **stSummary** outputs the group followed by individual participant results and in both cases records that the posterior model probabilities are presented:

```
Group level analysis (posterior model probabilities)
       Non-Trace        No-Overlap   Uni-dimensional Multi-dimensional
              0                  0                 0                 1


Individual participant analysis (posterior model probabilities)
                      1     2     3     4     5     6     7     8     9
Non-Trace         0.003 0.119 0.030 0.315 0.029 0.021 0.005 0.017 0.017
No-Overlap        0.257 0.316 0.014 0.079 0.031 0.055 0.158 0.015 0.013
Uni-dimensional   0.434 0.313 0.236 0.249 0.600 0.218 0.315 0.203 0.526
Multi-dimensional 0.306 0.252 0.720 0.357 0.339 0.707 0.522 0.766 0.443
                     10    11    12    13    14    15    16    17    18
Non-Trace         0.667 0.001 0.280 0.017 0.136 0.006 0.010 0.041 0.018
No-Overlap        0.016 0.004 0.012 0.477 0.222 0.130 0.017 0.006 0.034
Uni-dimensional   0.153 0.467 0.054 0.175 0.271 0.187 0.383 0.561 0.065
Multi-dimensional 0.163 0.527 0.654 0.331 0.370 0.677 0.589 0.393 0.882
```

For our DFIE example we see that with the exception of participant “10”, no participants showed positive support favouring the non-trace model, which is consistent with the group result, $gp_{NT,U} < .001$. Therefore our manipulation of study duration was largely successful in producing a monotonic effect on performance. Similarly, no support was found for the no-overlap model, $gp_{NO,U} < .001$, suggesting that our use of longer study durations for inverted than upright presentations achieved strong overlap of the data traces and hence a diagnostic design. The final two models can inform us of the underlying latent dimensionality. For our DFIE data we see that the multi-dimensional model received very strong support, $gp_{MD,U} > .999$, while the uni-dimensional model received little support, $gp_{UD,U} < .001$. However, the individual participant results were not always as decisive in their evidence favouring either a multi-dimensional or uni-dimensional account.

Following the model selection results, **stSummary** outputs any additional results that were specified; here we asked for the trace model prior probability, the number of encompassing samples that respected the non-trace model and the total number of samples drawn from the encompassing model:

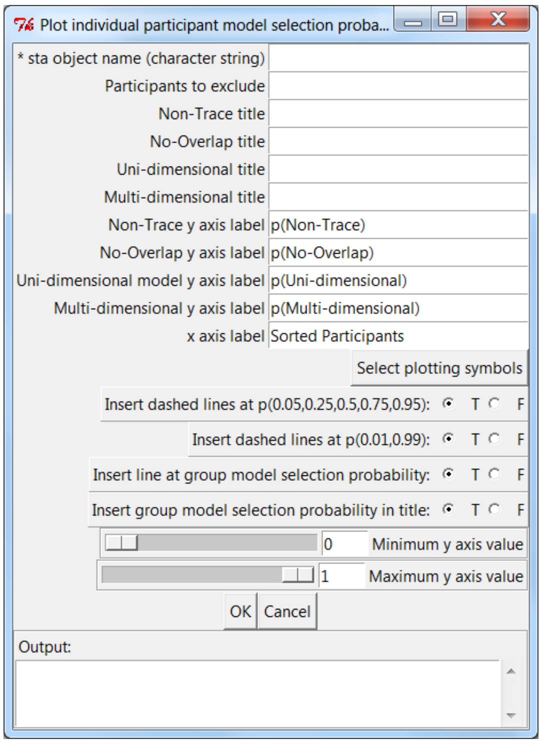
```
Trace model prior probability
[1] 0.0007716049


Non-Trace model samples from encompassing model
           1       2       3       4       5       6       7       8
D1S1 1358738 9120271 8630085 9020214 9109822 6653134 8665316 8114288
D2S1 9073437 2121706 9098084  775329 7168134 8179696 7505780 7702157
D1S2 4764984 7873208 8888863 2486003  580354 4576258 7892558  603246
D2S2 9075527 8818766 3049042 4147521 7765561 6958374 2677412 9107518
           9      10      11      12      13      14      15      16
D1S1 7478322  594330  260441 4978041 9117771 1840771 7808696 8534738
D2S1 8412627 8925902  908049 5950154 8527622 6683316 2919946 5309888
D1S2   73000 9128638 9166930 1840517 6583839 8775568 9158786 2777556
D2S2 7799845 8112082 2184966 4302973 5221332 9042383 5129042 9095012
          17      18
D1S1 4739738 9067263
D2S1 8434533 4480842
D1S2 6440382 8812980
D2S2 9149426 7555127
```

```
Total samples from encompassing model
            1        2        3        4        5        6        7
D1S1  8400000 14200000 11400000 12600000 13400000  7800000 15000000
D2S1 14200000  2200000 14300000  6500000  8600000 15300000 15400000
D1S2 13900000 15400000 12200000  2600000  5700000 13700000 15400000
D2S2 14400000 12000000 11800000  4500000  9600000 15300000 11200000
            8        9       10       11       12       13       14
D1S1 10300000  9100000   600000  3900000 14100000 14200000  1900000
D2S1 15400000 10900000 12400000  7000000  6800000 15200000 15200000
D1S2  5800000  2100000 13700000 13900000  1900000  7700000 14900000
D2S2 14000000 15400000 10300000 10300000 13400000 14300000 12700000
           15       16       17       18
D1S1 15400000 11200000 13900000 14500000
D2S1 11600000 14400000 11000000 13600000
D1S2 13800000 11400000  7500000 12000000
D2S2 14200000 14300000 14000000 15400000
```

Although the latter two sample counts are reported for each independent chain, note that *State-Trace* is not interested in the probability that the non-trace model is true for 'chain one' but not 'chain two' and so on, but rather the probability that it is true for all chains. Therefore, the posterior proportions are first calculated for the trace model at the level of the independent chains and then combined to calculate the posterior proportion for the complementary non-trace model. For example, consider participant "1": the posterior proportion of the trace model for chain one is $\hat{\Pi}_{T(1),U} = (8,400,000 - 1,358,738 + 1) / (8,400,000 + 2) = 0.8382$, and the combined trace posterior proportion is $\hat{\Pi}_{T,U} = 0.8382 \times 0.3610 \times 0.6572 \times 0.3698 = 0.0735$. The complementary non-trace model posterior proportion is therefore, $\hat{\Pi}_{NT,U} = 1 - 0.0735 = 0.9265$, and the non-trace Bayes factor is $BF_{NT,U}$ = 0.9265 / (1-0.00077) = 0.927.

As noted above these expensive computations (sampling and counting orders) can be re-used to assess many other model selection strategies. For example, Prince, Brown and Heathcote (2011) also suggested a sequential strategy, which first compares the trace and non-trace models. Here a Bayes factor would be calculated for both the trace model ($BF_{T,U}$ = 0.0735 / 0.00077 = 95.45, for participant "1") and the non-trace model ($BF_{NT,U}$ = 0.927), and then the non-trace posterior model probability calculated, $p_{NT,\{NT,T\}} = BF_{NT,U} / (BF_{NT,U} + BF_{T,U})$ = 0.0096. Given that the non-trace model is not supported, a similar comparison could then be made between the multi-dimensional and monotonic models and then (if warranted) between the uni-dimensional and no-overlap models (see Prince, Brown & Heathcote, for further details).

In additional to the tabular output of **stSummary**, the **stProbplot** GUI (Figure 7) allows the distribution over participants of posterior probabilities for each model to be inspected graphically. Again the only required input for this function is the name of the *sta* object (**sta object name (character string)*) and outlying participants can be excluded using the *Participants to exclude* parameter. **stProbplot** also allows the annotation within the plots to be extensively customised. A title can be added above each panel using the next four arguments (*Non-Trace title, No-Overlap title, Uni-dimensional title, Multi-dimensional title*), while the following five text-box arguments can customise the labels for y-axes (*Non-Trace y axis label, No-Overlap y axis label, Uni-dimensional y axis label, Multi-dimensional y axis label*) and for the x-axis (*x axis label*), which is common to all four panels. Note that if no label is desired, the default text should be replaced with a pair of double quotation marks. The *Select plotting symbols* button will link to a multi-option list GUI, which contains a range of filled and unfilled shapes, participant numbers and letters that can be used to denote each participant's value.



Plot individual participant model selection proba...

* sta object name (character string)	
Participants to exclude	
Non-Trace title	
No-Overlap title	
Uni-dimensional title	
Multi-dimensional title	
Non-Trace y axis label	p(Non-Trace)
No-Overlap y axis label	p(No-Overlap)
Uni-dimensional model y axis label	p(Uni-dimensional)
Multi-dimensional y axis label	p(Multi-dimensional)
x axis label	Sorted Participants

Select plotting symbols

Insert dashed lines at p(0.05,0.25,0.5,0.75,0.95):	[✓ T] F
Insert dashed lines at p(0.01,0.99):	[✓ T] F
Insert line at group model selection probability:	[✓ T] F
Insert group model selection probability in title:	[✓ T] F
Minimum y axis value	0
Maximum y axis value	1

Figure 7: GUI for the **stProbplot** function, which creates a graphical display of posterior model probabilities.

Dotted horizontal lines corresponding to the evidence categories in Table 1 can be customised using the next two true/false arguments, with *Insert dashed lines at p(0.05, 0.25, 0.5, 0.75, 0.95)* including the criteria for equivocal, positive and strong evidence and *Insert dashed lines at p(0.01, 0.99)* including the criterion for very strong evidence. The posterior probability for the group (based on the group Bayes factors) can also be displayed as a heavy dashed line on the plot (*Insert line at group posterior probability*) and as a numeric value in the title (*Insert group posterior probability in title*). Finally the range of values on the y-axis can be modified by dragging the slider bars corresponding to the *Minimum y axis value* and *Maximum y axis value* arguments.

Unlike the GUIs for non-graphical functions, the **stProbplot** GUI does not close after 'OK' is clicked. This allows the user to progressively customise the plots without re-calling the function and re-entering the parameter values each time. However, it is important to note that each time the plot is generated, the new plot will over-write the previous unless it is specified that R should record all of the plots generated in the graphics device, which is done by selecting *History* > *Recording* from the R console. Clicking the 'Cancel' button will dismiss the **stProbplot** GUI but does not "cancel" the most recent parameter values assigned or output produced.

As shown by the output for the example DFIE data in Figure 8, the **stProbplot** plot includes a panel for each model. Participants in each panel are sorted by their results, allowing those with extreme values to be easily identified. Figure 8 was made using the default values, which produces appropriate annotation in most cases, and by choosing the option to use integer plot symbols in order to easily identify each participant's result.

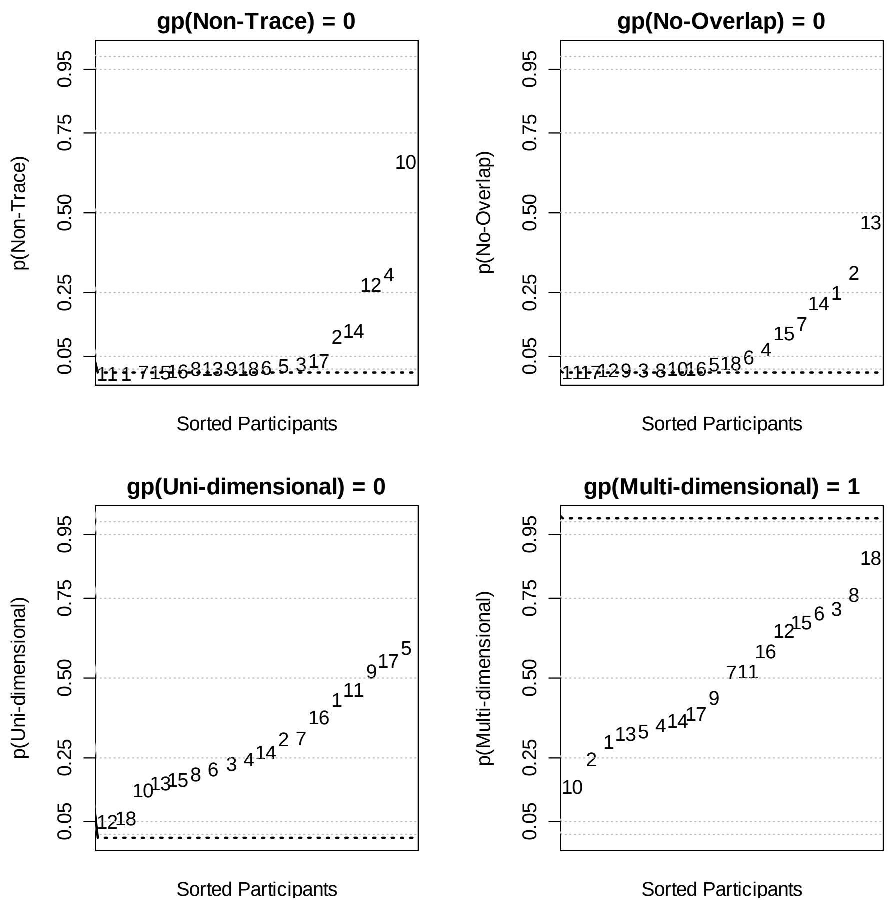
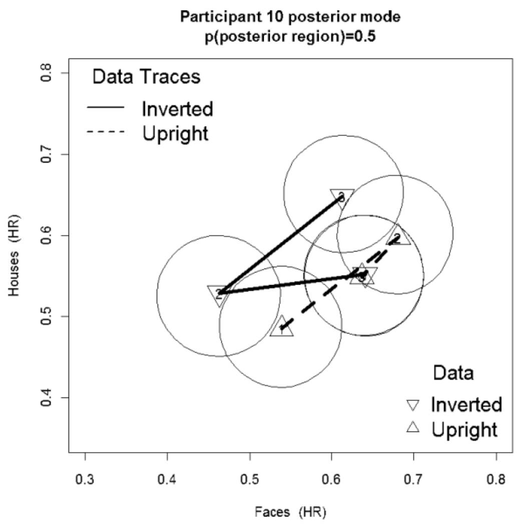


Figure 8: Posterior model probabilities for each participant (denoted by their participant number) for the DFIE example data for each of the four diagnostic models (panels). Group posterior model probabilities for each model are indicated in panel titles and plotted as a heavy dashed line. Within each panel, participant results are sorted in ascending order of their probability estimates and faint dashed lines demarcate the categories in Table 1.

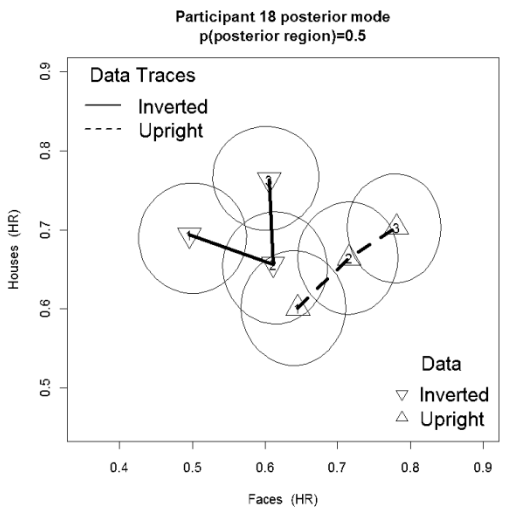
For example, Figure 8 clearly suggests that participant “10” provides an outlying result in favour of the non-trace model, as was also noted from the **stSummary** output. However, follow-up analysis excluding participant “10” revealed little influence on the group posterior model probability (*gp* in the panel titles). As previously noted, overall these results show positive or greater evidence for the trace model and for data-trace overlap (i.e., low posterior probabilities for the non-trace and no-overlap models). Evidence is weaker and individual variability greater in relation to the dimensionality results, but the group evidence clearly supports a multi-dimensional outcome. With respect to individual variability in dimensionality, the **stProbplot** plots can also easily reveal a potential mixture of uni-dimensional and multi-dimensional sub-groups, but this is not indicated in Figure 8.

3.7.2. The State-Trace Plot

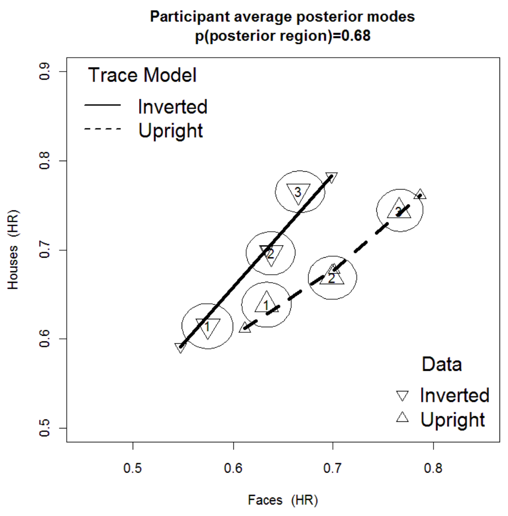
Figure 9 shows examples of state-trace plots produced using the **stPlot** GUI. Although we could create state-trace plots using the current example *sta* object, which has now been run to



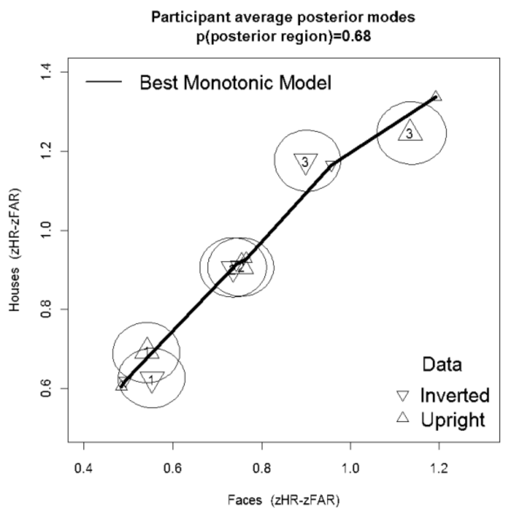
(a)



(b)



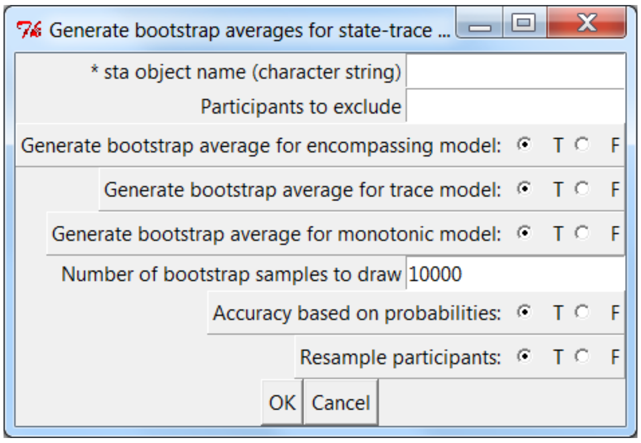
(c)



(d)

Figure 9: Accuracy state-trace plots showing modes of the posterior estimates from the encompassing model (large symbols) for the 2AFC DFIE data set for (a) participant 10, (b) participant 18, and (c) on average, as well as (d) on average for the yes-no data set. Accuracy is indicated by plotting the hit rate (HR) for the 2AFC DFIE data and *d'* (i.e., *z*(HR) – *z*(FAR), where FAR is the false alarm rate) for the yes-no example. For (a) and (b) lines are data traces, joining the posterior modes of the encompassing model samples, and ellipses represent 50% credible regions. For (c) lines with small symbols join posterior modes of the trace model and for (d) they join the highest posterior probability (i.e., most frequently sampled) monotonic model. The ellipses in (c) and (d) represent 68% credible regions.

completion using **stSample**, in order to illustrate the full range of state-trace plot options **stBootav** should also be run to obtain bootstrap averages. Again the only required input for the **stBootav** GUI (Figure 10) is the name of the *sta* object (**sta object name (character string)*) and the option is also provided to exclude participants from the bootstrap averages (*Participants to exclude*). The **stBootav** GUI allows users to choose to calculate bootstrap averages (based on the stored posterior samples for each participant) for one or more of the encompassing, trace and monotonic models (*Generate bootstrap average for encompassing model*, *Generate bootstrap average for trace model*, and *Generate bootstrap average for monotonic model* respectively). Note the bootstrap average for the encompassing model is calculated at the end of **stFirst**, which enables preliminary state-trace plots to be generated prior to running **stBootav**.


Generate bootstrap averages for state-trace ...

* sta object name (character string)
Participants to exclude
Generate bootstrap average for encompassing model: [✓ T] F
Generate bootstrap average for trace model: [✓ T] F
Generate bootstrap average for monotonic model: [✓ T] F
Number of bootstrap samples to draw 10000
Accuracy based on probabilities: [✓ T] F
Resample participants: [✓ T] F

Figure 10: GUI for the **stBootav** function, which is used to obtain the bootstrap participant averages.

A set of bootstrap averages is created by repeatedly randomly selecting with replacement one sample from each participant's set of posterior samples for a given model and taking their mean. By default, these averages are based on 10,000 bootstrap samples (*Number of bootstrap samples to draw*). Each time **stBootav** is invoked it can compute averages for only one type of accuracy measure (i.e., based on probabilities ("F") or *z*-transformed probabilities ("T"); *Accuracy based on z transformation?*). Finally, the **stBootav** function also allows participants to be selected at random with replacement on each bootstrap repetition (*Resample participants* = "T"); this produces a set of averages with the same central tendency but greater variability that is appropriate when the participants are treated as a sample from a population.

For our DFIE example, **stBootav** was run using the default values, with the exception of setting the *Resample participants* argument to "F". Once executed, **stBootav** prints a number of updates in the R console, including the participant identifiers for those included in the bootstrap average and a record of the models for which the bootstrap averages were calculated:

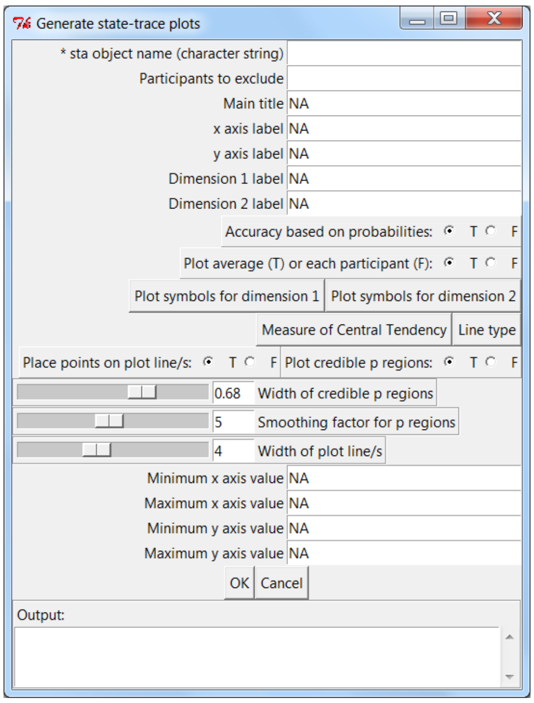
```
Calculating bootstrap average over participants:
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18

Calculating encompassing bootstrap average

Calculating trace bootstrap average

Calculating best monotonic bootstrap average
```

Note that monotonic samples may be relative rare for some data, and so the averages can be unreliable; to alert users to this possibility, when asked to calculate the bootstrap average for the monotonic model, **stBootav** also reports the number of participants who have less than 100 samples:

```
Less than 100 monotonic samples available for participants:
  6  8 12 18
 39 55 22  8
```

A participant with no monotonic samples is excluded from the average. The problem of a lack of monotonic samples might be addressed by calling **stSample** again with a stricter criterion, but usually a lack of monotonic samples indicates the monotonic model is not appropriate for the data and so there is no point plotting it (e.g., monotonic samples may rare for our DFIE data as it is non-monotonic).

The **stPlot** GUI (Figure 11) operates much like the **stProbplot** GUI. The name of an *sta* object (** Character string of sta object name*) must be provided and participants may again be excluded from the generated plots by specifying the corresponding participant identifiers (*Participants to exclude*). Note that this latter value must match the participants excluded when **stBootav** was run, as must the accuracy measure to use in the state-trace plots (*Accuracy based on z transformations?*).


Generate state-trace plots

* sta object name (character string)
Participants to exclude
Main title NA
x axis label NA
y axis label NA
Dimension 1 label NA
Dimension 2 label NA
Accuracy based on probabilities: [✓ T] F
Plot average (T) or each participant (F): [✓ T] F
Plot symbols for dimension 1 | Plot symbols for dimension 2
Measure of Central Tendency | Line type
Place points on plot line/s: [✓ T] F
Plot credible p regions: [✓ T] F
0.68 Width of credible p regions
5 Smoothing factor for p regions
4 Width of plot line/s
Minimum x axis value NA
Maximum x axis value NA
Minimum y axis value NA
Maximum y axis value NA
OK | Cancel
Output:

Figure 11: GUI for the **stPlot** function used to generate state-trace plots.

stPlot also allows for extensive customisation of the annotation in the state-trace plots. A title can be included above the state-trace plot (*Main title*) and labels can be specified for each level of the state factor (*x axis label* and *y axis label*) as well as for each level of the dimension factor (*Dimension 1 label* and *Dimension 2 label*), which correspond to the axes and legend entries respectively. By default, **stPlot** will fill these latter four values with the appropriate level obtained from the raw data files (e.g., ‘F’, ‘H’, ‘I’, and ‘U’ respectively for the DFIE data as shown in Figure 2).

However, alternate labels that are more informative of the experimental design can be specified by modifying the value in the appropriate text box (e.g., for the DFIE plots in Figure 9, we entered ‘Faces’, ‘Houses’, ‘Inverted’ and ‘Upright’ in the four text boxes respectively). Note also that the accuracy measured specified will automatically be included in parentheses after the axis label (e.g., HR or *z*HR for 2AFC data and HR – FAR or *z*HR – *z*FAR for yes-no data).

Additionally, the symbols used to represent each level of the dimension factor can be specified using the buttons *Plot symbols for dimension 1* and *Plot symbols for dimension 2*. Note there is no option to use a filled symbol as *stPlot* will include a numeric value within each shape to identify the corresponding trace levels. The range of values for the x and y axes can also be modified by entering a value in the text boxes corresponding to the arguments *Minimum x axis value, Maximum x axis value, Minimum y axis value* and *Maximum y axis value*. However, leaving these values as the default “NA”, **stPlot** will automatically scale the axes to fit the data and credible regions.

The **stPlot** function represents accuracy data using any one of three measures of central tendency (the mode, mean or median) applied to posterior samples from the encompassing model (*Statistic to use for model plotting*). As the encompassing model makes no order assumptions these central tendency measures (the large symbols in Figure 9) provide a model-free estimate of the observed data. The default choice used to create Figure 9 (the mode) produces estimates that are usually equivalent to the familiar maximum-likelihood formula (e.g., n / N for the 2AFC hit rate, where n is the number of correct in N trials).

The other central tendency measures usually produce similar results, at least for reasonable sample sizes not subject to floor or ceiling effects. For example, for the hit rate and uniform prior used by **StateTrace**, the mean of a large sample from the encompassing posterior is equivalent to $(n + 1) / (N + 1)$. For other accuracy measures such simple formulae are not available. This is also the case for any accuracy measures for any of the order restricted models. Hence, estimates based on posterior samples have the advantage of providing an easily applied and general approach.

The **stPlot** function uses the same approach to display the degree of uncertainty in central tendency estimates, by drawing contours around regions containing a specified percentage of the posterior encompassing-model samples (*Plot credible p regions*). Estimating Bayesian credible regions in this way works with all accuracy measures in a way that takes account of any floor and ceiling effects, which can be very influential when contours are near bounds in an accuracy measure. The regions, and modes, are estimated using the **bkde2D** function in Wand and Ripley’s (2009) **KernSmooth** package, which is included by default in R.

The **stPlot** GUI allows users to choose the percentage contained by the regions (*Width of credible p regions*) and the degree of smoothing (*Smoothing factor for p regions*), as a multiple of the maximum over data points of the values provided by the **dpik** function: this **KernSmooth** function, and **bkde2D**, are called with default values. The default multiplier of five used by **stPlot** was chosen to produce very smooth contours even for large regions, which are otherwise often irregular because they require estimation of the tails of the posterior distributions; users are encouraged to experiment with the multiplier in their application.

Users are also able to create either an average state-trace plot or a plot for each individual participant (*Plot average (T) or each participant (F)*). Figure 9a and 9b were generated by setting this argument to "F" and plot the results for participant "10", who had the strongest evidence for a violation of the trace model in Figure 8, as well as participant "18", who had the strongest evidence for the multi-dimensional model; both state-trace plots are clearly consistent with the model-selection analyses. Individual participant data are typically quite noisy, so for clarity the credible regions in these plots contain only 50% of the posterior samples. The lines in these plots are *data traces* (the default *Line type* value), which join points with the same dimension factor level (upright or inverted; NB., the weight of the plot lines can be specified using *Width of plot line/s*). Both data sets display strong data-trace overlap, consistent with the results for the no-overlap model in Figure 8. Note that plots such as 9a and 9b can be made as soon as **stFirst** is run, as **stFirst** stores sufficient samples from the encompassing model for each participant (10,000 for each) and runs the **stBootav** function necessary to produce averages.

Figure 9c was generated by setting the *Plot average (T) or each participant (F)* argument to "T" and is a state-trace plot of the average over all participants in the example DFIE data. Data points in average plots represent the central tendency of the set of bootstrap averages. Variability among the averages is used to construct credible regions in the same way as for individual participants. These credible regions reflect uncertainty in the estimated average over the particular set of participants in an experiment. Reflecting the reduction in uncertainty associated with an average, the credible regions are much smaller in Figure 9c, even though they contain the default value of 68% of the posterior samples (corresponding to the proportion of a normal distribution contained by conventional standard error bars).

The lines in Figure 9c join the modes of the average of samples from the trace model. The points joined by the lines (*Place points on plot line/s* = "T") are different from the large symbols, which are estimated based on encompassing model samples, as the encompassing model admits samples that violate the trace model. However, in this case the difference is not large, reflecting the fact that for most participants the trace model provides an excellent description of this data.

Figure 9d plots average results for the yes-no experiment using the signal-detection theory *d'* measure of accuracy (i.e., z(HR) – z(FAR)). The lines in Figure 9d join the modes of the most commonly occurring order[3] for monotonic MCMC samples (the *best* monotonic model); that is, MCMC samples from either the uni-dimensional or no-overlap models. Because this data is well described by a uni-dimensional model the difference between the best monotonic model and encompassing model modes is relatively small. The order in which the points are joined, according with increasing trace factor levels, reflects almost perfect overlap between data traces in the average data. Note that Figures 9c and 9d were created after first running **stSample** to the default criterion then **stBootav** to average the trace and monotonic samples stored by **stSample**.

[3] It is important to note that the best order may differ between participants. Before interpreting the best (most frequently occurring) monotonic model in the average, such as is plotted in Figure 9d, it is advisable to use **stSummary** to examine the degree of variability in the best orders over participants, as strong individual difference may mean that taking an average is not sensible. In this case, despite considerable individual variability, the best monotonic order for the average is the same as the most frequently occurring order for individual participants.

4. Summary and Conclusions

The question of latent dimensionality (i.e., whether a single latent variable mediates the relationship between the effect of two experimental factors) has pervaded not only basic research in areas of memory, perception and categorisation but it also an important consideration in applied settings such as clinical psychology, human factors, aging and development (see Prince, Brown & Heathcote, 2011). Traditionally, this question is addressed using an ANOVA interaction test, with any one-dimensional account rejected when a significant interaction is observed. However, it is widely known that this approach requires strong assumptions to be made that are difficult or impossible to test.

In contrast, state-trace analysis provides a graphical method for addressing latent dimensionality, which makes only ordinal assumptions and so avoids confounds from range effects that can distort other methods of assessing latent dimensionality when performance is measured on a bounded scale. Although state-trace analysis requires researchers to consider different methodological issues, Prince, Brown and Heathcote (2011) have provided detailed guidance to help develop and refine a state-trace experiment. Furthermore, they proposed an inferential method for state-trace data, based on Klugkist, Kato and Hoijtink's (2005) and Klugkist, Laudy and Hoijtink's (2005) encompassing prior method for estimating Bayes factors. These inferential tests can be used to not only assess dimensionality (i.e., to estimate the probability that a one-dimensional or multidimensional model is best able to account for the data) but also to help refine experimental methodology and to check the validity of the dimensionality assessment.

A Bayesian approach is particularly suited to state-trace analysis for two main reasons. First, the four models proposed by Prince, Brown and Heathcote (2011; the non-trace, no-overlap, unidimensional and multi-dimensional models) differ tremendously in their ability to fit data by chance. The Bayes factor compensates for these differences, and so does not inappropriately favour more flexible models. Second, in contrast to null hypothesis statistical testing, a Bayesian analysis can quantify evidence in favour of a simpler "null" model (e.g., a one-dimensional account) as well as evidence against it. This feature of the Bayesian approach is ideally suited to state-trace analysis because, as Loftus (2002) notes, state-trace analysis is an example of a class "equivalence" method that treat simplicity (i.e., invariances) and differences as equally important. This even-handed approach has proven fruitful in many other areas of science, and stands in stark contrast to the focus on often inconsequential "significant" differences (i.e., Meehl's, 1990, "crud factors") encouraged by the widely acknowledged limitations of null hypothesis testing.

However, these Bayesian procedures have the potential to narrow the focus of state-trace applications to only researchers that are familiar with the sampling and estimation techniques required for calculating Bayes factors and corresponding posterior model probabilities. In this paper we have, therefore, provided users with a software package, **StateTrace**, to aid the broader adoption of these methods. In particular, we provide a GUI for each of the functions in order to make these procedures more user-friendly for those who are also not familiar with the R language. Each of the available GUIs provides users with a description of the arguments and allows argument values to be entered via widgets including text boxes, slider bars, true/false check boxes and multi-option lists. Moreover, most functions require very minimal input (often just the name of the *sta* object) in order to execute a function and still obtain meaningful results. The other arguments then enable users to customise parameter values for their own needs.

The **StateTrace** package offers very general purpose data input capabilities, such that very little parsing should be required of a raw data file before it is ready for analysis. The **stFirst** function allows users to specify the relevant columns from a data file, the order in which columns should be read-in, and any non-relevant rows that should be excluded. Moreover, **stFirst** is customised for initiating an analysis and so will also perform the necessary procedures to allow for a preliminary examination of results; including calculating posterior proportions for the four models and estimating how much additional computation time may be required, as well as checking the convergence of the MCMC trace chains and generating preliminary state-trace plots for the individual participants and group average.

All of the relevant information (sample counts and posterior estimates) required for a range of model selection strategies is stored in the *sta* object. However, **StateTrace** also provides users with a number of summary functions that will not only extract this information but also allow for a customised presentation of the results. The **stSummary** function for example, can output tabular model selection results as Bayes factors or posterior model probabilities, and if probabilities are selected can use an exhaustive or trace-true model selection strategy. **stSummary** can also simply print the raw sample counts and prior probabilities if the user desires to explore other model selection strategies (e.g., sequential model selection strategy). Alternatively, **stProbplot** provides a graphical display of the posterior model probabilities and can also plot lines at the critical *p* values to aid visual assessment of these results. Similarly, for researchers who prefer a graphical approach over inference, the **stPlot** function can be used to create a customised state-trace plot; including drawing a line to represent the best trace or monotonic model to visually examine a model's fit to the data. The GUIs for these two plotting functions are particularly user-friendly as they do not close on execution. This allows the user to go back and forth between modifying parameter values and generating a plot as many times as is necessary.

The Bayesian analyses we describe here can be applied to other types of state-trace analysis, such as "dependent-variable state-trace analysis", which examines pairs of different types of dependent measurements (e.g., 2AFC judgements and confidence ratings) to determine whether they are mediated by a common latent variable. However, at present **StateTrace** is limited to binary measurements. It is also limited to designs in which all factors are repeated measurements, which enable analyses to be applied to each participant's data separately. Analyses of data averaged over participants are avoided as Prince, Brown and Heathcote (2011) showed they are potentially misleading. In future releases we intend to extend the **StateTrace** package to address responses with more than two alternatives, state and dimension factors with more than two levels, and designs with between-subjects factors.

5. References

Bamber, D. (1979). State-trace analysis: A method of testing simple theories of causation. *Journal of Mathematical Psychology, 19,* 137-181.

Bogartz, R.S. (1976). On the meaning of statistical interactions. *Journal of Experimental Child Psychology, 22,* 178-183.

Busemeyer, J.R., & Jones, L.E. (1983). Analysis of multiplicative combination rules when the causal variables are measured with error. *Psychological Bulletin, 93,* 549-562.

Dunn, J.C. (2003). The elusive dissociation. *Cortex, 39,* 177-197.

Dunn, J.C. (2004). Remember-know: A matter of confidence. *Psychological Review, 111,* 524-542.

Dunn, J.C. (2008). The dimensionality of the remember-know task: A state-trace analysis. *Psychological Review, 115,* 426-446.

Dunn, J.C., & Kirsner, K. (1988). Discovering functionally independent mental processes: The principle of reversed association. *Psychological Review, 95,* 91-101.

Gelfland, A.E., Smith, A.F.M., & Lee, R.-M. (1992). Bayesian analysis of constrained parameter and truncated data problems. *Journal of the American Statistical Association, 87,* 523-532.

Gilks, W.R., Richardson, S., & Spiegelhalter, D.R. (Eds.) (1996). *Markov Chain Monte Carlo in Practice*. Boca Raton, F.L.: Chapman & Hall/CRC.

Glanzer, M., & Cunitz, A.R. (1966). Two storage mechanisms in free recall. *Journal of Verbal Learning and Verbal Behaviour, 5,* 351-360.

Henson, R. (2006). Forward inference using functional neuroimaging: Dissociations versus associations. *Trends in Cognitive Sciences, 10,* 64-69.

Hoffman, T.J., & Laird, N.M. (2009). Fgui: A method for automatically creating graphical user interfaces for command-line R packages. *Journal of Statistical Software, 30,* 114. URL http://www.jstatsoft.org/v30/i02/.

Kass, R.E., & Raftery, A.E. (1995). Bayes factors. *Journal of American Statistical Association, 90,* 773-795.

Klugkist, I., Kato, B., & Hoijtink, H. (2005). Bayesian model selection using encompassing priors. *Statistica Neerlandica, 59,* 57-69.

Klugkist, I., Laudy, O., & Hoijtink, H. (2005). Inequality constrained analysis of variance: A Bayesian approach. *Psychological Methods, 10,* 477-493.

Loftus, G.R. (1978). On interpretation of interactions. *Memory & Cognition, 6,* 312-319.

Loftus, G.R. (1996). Psychology will be a much better science when we change the way we analyse data. *Current Directions in Psychological Science, 5,* 161-171.

Loftus, G.R. (2002). Analysis, interpretation, and visual presentation of experimental data. In H. Pashler (ED.), *Stevens' handbook of experimental psychology* (Vol. 4, pp. 339-390). New York: John Wiley and Sons.

Loftus, G.R., Oberg, M.A., & Dillon, A.M. (2004). Linear theory, dimensional theory, and the face-inversion effect. *Psychological Review, 111,* 835-863.

Maurer, D., LeGrand, R., & Mondloch, C.J. (2002). The many faces of configural processing. *Trend in Cognitive Sciences, 6*, 255-260.

Meehl, P.E. (1990). Why summaries of research on psychological theories are often uninterpretable. *Psychological Reports, 66,* 195-244.

Newell, B.R., & Dunn, J.C. (2008). Dimensions in data: Testing psychological models using state-trace analysis. *Trends in Cognitive Sciences, 12,* 285-290.

Plummer, M., Best, N., Cowles, K., & Vines, K. (2006). CODA: Convergence diagnosis and output analysis for MCMC. *R News, 6,* 7-11.

Poldrack, R.A. (2006). Can cognitive processes be inferred from neuroimaging data? *Trends in Cognitive Sciences, 10,* 64-69.

Prince, M., Brown, S., & Heathcote, A. (2011). The design and analysis of state-trace experiments. *Psychological Methods.*

Prince, M., Hawkins, G., Love, J., & Heathcote, A. (submitted). An R package for state-trace analysis. *Behavioural Research Methods.*

R Development Core Team (2007). R: A language an environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.r-project.org.

Raftery, A.E. (1995). Bayesian model selection in social research. *Sociological Methodology, 25,* 111-163.

Rakover, S.S. (2002). Featural vs. configurational information in faces: A conceptual and empirical analysis. *British Journal of Psychology, 93,* 1-30.

Rock, I. (1974). The perception of disoriented figures. *Scientific American, 230,* 78-85.

Shallice, T. (1988). *From Neuropsychology to Mental Structure.* Cambridge University Press.

Teuber, H-L. (1955). Physiological psychology. *Annual Review of Psychology, 6,* 267-296.

Valentine, T. (1988). Upside-down faces: A review of the effect of inversion upon face recognition. *Journal of British Psychology, 79,* 471-491.

Wand, M., & Ripley, B. (2009). KernSmooth: Functions for kernel smoothing for Wand and Jones (1995) “Kernel Smoothing”. R package. http://cran.r-project.org.

Yin, R.K. (1969). Looking at upside-down faces. *Journal of Experimental Psychology, 81,* 141-145.

6. Acknowledgements

Thanks to Cassandra Barrett, Amanda Brown, Christopher Brown, Sara Buller, Anna Charley, Nekia Dalley, Stuart Donaldson, Katie Foster, Jessica Gordon, Alice Grady, Tonelle Handley, Georgia Kelaher, Elaine Yheng Ching Loke, Katherine Moore, Soo Li Quah, Kate Smith, Evelyn Tan and Lucy West for assistance with stimulus preparation and running of participants for the example data.