Behavioral Analysis Suite v1.0 Documentation (Last Updated: 8/3/13)

Introduction

This guide will teach you how to use the Behavioral Analysis Suite, a software package allowing easy, fast analysis of behavioral data. It should work for any behavioral paradigm with a discreet trial structure. Its features include:

- Both online and offline analysis
- Trial sorting according to any parameter
- Sliding window plots of behavioral performance throughout a session
- Ability to combine datasets from multiple sessions and/or across multiple animals
- Easy to incorporate pop-out modules to further analyze subsets of data. Included modules:
    - Run speed analysis (exclusively for ViRMEn experiments)
    - Significance by bootstrapping
    - Position and velocity plots
    - Effect of preceding trials on trial choice
- Exportable processed data and figures

In order to use this package, your data will need to be saved in a specific format, and some, though not much, coding is required.

Getting Started - Data storage

This suite requires that the majority of data be stored in a 1xnTrials cell array of structures. In this format, each cell contains a structure with the relevant information from that trial. For example, one structure format might be to separate the subfields into four categories: info, maze, result, and time, each of which contains further subfields containing details. The exact makeup of this structure is up to the user, but several fields are required. These are listed below:

- result.correct (The result of the trial as a logical)
- info.mouse (the number of the mouse)
- info.conditions (a cell array of strings containing the name of each condition)
    - For example, {'Black Left','Black Right','White Left','WhiteRight'}
- info.itiCorrect (the delay period following a correct trial)
- info.itiMiss (the delay period following an incorrect trial)
- info.name (the name of the task)
- time.start (the time of the trial start as a Matlab datenum)
- time.stop (the time of the trial end as a Matlab datenum)

Other subfields to specify are entirely up to the user. To make this process simpler, use the function createSaveStruct.m included with the package in the User Processing folder. This function takes in a variety of arguments and outputs data as a structure. When modifying this function for your use, be aware that there should be two categories of variables: those which are used by every task and those which are only relevant to some tasks. Variables used in every task should be hardcoded by adding an input argument to the function BEFORE the 'varargin' argument. Simply input the field structure in the format 'data.x.y = variable' in the function to create a field for that variable.

Variables used in only some tasks should make use of the variable arguments option. In this situation, variables will be added to the end of the input arguments to the function in the format:

```
(…,'variable name as string',variable value)
```

For example, a variable named whiteDots would be added in as:

```
(…,'whiteDots',vr.whiteDots)
```

To use this, do not add in your variable as an input argument to the function. Instead, add a case to the switch/case statement at the end of the function. The case should be an uppercase version of the variable name string to be used as an argument. Within the case statement, add in a line in the format:

```
case 'WHITEDOTS'
      data.maze.whiteDots = varargin{i+1};
```

Create an entry like this for each variable which will only be saved for some experiments. This allows you to use the same function (createSaveStruct.m) to create a variety of different structures.

Data Storage Requirements

Unfortunately, saving the structures is not as simple a process as it seems. Because Matlab is pass by value (meaning that it writes a copy of the variable passed into a function), any behavioral software which runs a function on each iteration (such as ViRMEn) cannot have any particularly large variables passed to it without slowing it down substantially, especially as more and more trials are added. For example, having the cell array of structures stored in the structure passed to ViRMEn results in a marginal frame rate drop early on, but a large frame rate drop by the time 20 or 30 trials are reached (from 60 Hz to as low as 4 or 5 Hz).

To circumvent this issue, data must be saved on each iteration of the software. This eliminates the issue of having an ever growing dataset passed into the runtime function but adds another issue: Matlab cannot append values to already saved cell array (the built-in function ‘matfile’ does this, but unfortunately does not work for cell arrays or structures). Thus, as the cell array grows, the save time grows as well, increasing from ~10 ms for one trial to over a second for 100 trials.

So what’s the solution? Each trial has to be saved as its own variable into one file. Thus, instead of having one cell array saved in a file, there will be as many structures as trials saved in the same file and appended (using the save ‘-append’ option) at the completion of each trial. At the completion of the experiment, running the function catCells.m will then combine the structures into a cell array of structures.

catCells.m requires that each variable have the same start string and simply have the trial number appended to it. So, the first trial would be titled ‘data1’, the second, ‘data2’, and so on and so forth. The easiest way to do this is with an ‘eval’ statement looking something like this:

```
[dataStruct] = createSaveStruct(…); %create the data structure using
eval(['data',num2str(vr.numTrials),'=dataStruct;']); %create new variable
%save datastruct
save(vr.pathTempMatCell,['data',num2str(vr.numTrials)],'-append');
```

The code above creates a variable dataStruct which has the data from that trial, stores it into a new variable with the base string data and numTrials added to it (eg. data32), and then saves it into a specific file using the ‘-append’ flag.

With this method, each save operation only takes ~10 ms, so it marginally slows down the code. Then when the code terminates, running catCells.m will reformat the data into a single cell array:

```
[dataCell] = catCells(vr.pathTempMatCell,'data');
```

The first argument for catCells is the path of the stored data, and the second argument is the start string of the variables (the string before numTrials, so in this case, ‘data’). The output is a cell array in the format required: 1xnTrials with each cell containing a structure of information for that trial.

Current vs. Archived Animals

The software assumes data will be divided into two categories: current and archived (though the names of these are modifiable). This allows easy sorting of mice who are in use currently vs. old mice.

File Names

Files should be named after the following fashion: ‘AM008_120529’. In this format, the first two characters are the user’s initials, the next three the number of the mouse, and following the underscore the date of the experiment in the format ‘yymmdd’. The data array which is saved on each iteration of the behavioral task, it should be stored in a ‘.mat’ file with the information stored in an array titled ‘data’. The cell array data should have ‘_Cell’ appended to the filename (eg. ‘AM008_120529_Cell.mat’). Finally, if you have multiple files from a single day, they should have ‘_1’ appended to the end of the names (eg. ‘AM008_120529_1.mat’ or ‘AM008_120529_Cell_2.mat’).

Temporary Files Necessary for Online Processing

If you don’t intend to use the package for online processing or if you’re intending to run the analysis package on the same computer that runs the behavioral task, you can skip this section.

To use the analysis package on another computer, both computers must be on the same network (any of the computers on the HMS Private or plugged into the HMS network should fulfill this condition). First, on the computer running the behavioral task, create a shared folder titled ‘Temporary’ or something similar. The important part is that this folder must be accessible by other computers on the network.

During the behavioral task, data should save into this directory. Then, at the end of the task, the data can be copied out of this temporary directory and into the permanent directory. This should happen at the same time as the cell concatenation described above. Additionally, before the task begins, one variable titled ‘conds’ should be saved into the temporary directory which will contain the trial structures with the names of the conditions (this should be the same as the info.conditions field described above).

Getting Started – Installation

To install the software, simply copy the Behavioral Analysis Suite folder to your computer and add the entire folder to the Matlab path.

Getting Started - User Configuration File

This software package requires the user to complete a user configuration file. This file can be renamed and you can have multiple user configurations. Instructions are present in the userData file included with the package, but they are replicated below.

Section 1 – This is the path to stored data for offline processing. Within this folder should be two other folders representing your current and archived data. Ex. C:\Data\Ari\

Section 2 – These should be the initials of the user. Ex. AM

Section 3 – This should be the base filename for the temporary storage files used for online processing. Ex. tempStorage

Section 4 – This should be the path to the temporary storage directory accessible by the computer running the package. Ex. \\HARVEYRIG1\Temporary

Section 5 – These should be the two names of the current and archived folders on separate lines. Ex.
Current Mice
Archived Mice
It is important not to remove the end statement below this section.

Section 6 - These flags should be conditions which can be fed into an if statement to specify whether or not to tabulate specific variables as designated in the next section. The first flag listed will be considered flag '1', the second flag '2', and so on and so forth. Multiple flags should be listed on separate lines.
Ex. isfield(dataCell{1}.maze,'tower')
In this example, the conditional flag is that the field tower must exist in the first cell of the data.

Section 7 – Sliding Window Variables. This section should include any variables which should be included in the sliding window calculations. Each variable should have arguments separated by the delimiter, '|'. For each variable, the first argument should be the name of the variable, starting with 'winData.' The second argument should be whether or not a flag applies to the variable. If no flag applies, this argument should be 0. If a flag does apply, this argument should be the number of the flag. The remaining arguments (there is a limit of 10) should be cell specifiers in the format 'result.correct == 1'. These will be combined to generate the variable.

Examples:

```
winData.leftTurns|0|result.leftTurn==1
```

In this first example, the variable to be calculated is titled ‘leftTurns, no flag applies to it (because of the 0, which means that it will be calculated regardless of the task), and it is the number of trials in each window which fulfill the condition result.leftTurn == 1.

```
winData.nRewardsNoTower|1|result.correct==1|maze.tower == 0
```

In this example, the variable to be calculated is titled ‘nRewardsNoTower’, a flag does apply to it (flag 1, meaning it will only be tabulated when flag 1 is satisfied), and it is the number of trials in which both conditions (‘result.correct==1’ AND ‘maze.tower==0’) are satisfied.

Section 8 – Percentages for sliding window. This section should include the percentages which are going to be calculated and actually displayed in the sliding window. The first argument should be the display name of the variable in the GUI. The second argument should be the flag number as in section 7 (0 if no flag applied, otherwise the number of the flag).The third argument should be the name of the percentage variable, starting with 'winData.' The fourth argument should be the numerator, and the fifth the denominator. Arguments should be separated by the '|' delimiter, with each separate calculation on a new line. Note that variables in the numerator and denominator MUST be defined in section 7 above, UNLESS they are winData.nTrials, which represents the total number of trials in the window, or winData.nRewards, which represents the total number of rewards within the window. The standard percent correct (rewards/trials) calculation is built in.

Example:

```
Left Turns|0|winData.percLeft|winData.leftTurns|winData.nTrials
```

In this example, a percentage calculation is defined which will be displayed as ‘Left Turns’. No flags apply to it (because the second argument is 0). The name of the variable will be ‘winData.percLeft’ and it will be calculated as 100*(winData.leftTurns/winData.nTrials). For this to work, winData.leftTurns must be a variable which had been defined in section 7.

Getting Started – Understanding Conditions

The main advantage of storing data in the cell array of structures format is that it allows efficient sorting of trials based on any parameter stored in the structures. To this end, the package includes several functions which allow for fast trial sorting: findTrials and getTrials. Both of these make use of a special conditional entry format[1], described below.

Conditional Strings

To specify a single condition, simply input a string designating the subfields of dataCell, ignoring dataCell itself and the operator. For example, to extract all correct trials, input 'result.correct==1'.

One can also specify multiple values for a given condition. For example, the string 'result.correct==0,1' will be parsed to mean 'result.correct==0 || result.correct==1'. The operator in between the two values can be an AND or an OR and depends on the operator used to designate the condition. '==' makes use of the OR operator ('||'), while '~=' makes use of the AND operator ('&&'). So the string 'maze.condition~=1,2' would read as 'maze.condition ~= 1 && maze.condition ~= 2'. Note that only one value can be designated for the greater than and less than operators.

The user can also specify multiple operators for the same parameter separated by & or | (single). For example, 'maze.condition == 1 | > 2' will return 'maze.condition ==1 || maze.condition > 2'. Note that the same operator cannot be reused. This will return an error otherwise.

The user can also specify ranges as a single condition with less than. For example, '4 < maze.condition <= 6' will return 'maze.condition > 4 && maze.condition <= 6'. This can only be used less than and multiple values cannot be used.

To specify multiple conditions, input several strings as above separated by a semicolon. The conditions will then be combined with an && in between them. Note that a semicolon is only necessary between conditions, and not at the end of the last condition.

findTrials and getTrials

Both findTrials and getTrials take in two arguments: the dataCell, and a string specifying which trials should be selected. findTrials returns a 1xnTrials logical array specifying whether each trial meets the conditions set forth in the string while getTrials returns a cell array of structures containing only the trials in which the condition was satisfied. So, the following code,

```
sum(findTrials(dataCell,'result.correct==1'));
```

returns the number of correct trials. Alternatively, the code below,

```
getTrials(dataCell,'maze.condition == 1,2');
```

[1] Many thanks to Alex Trott in the Born lab for coming up with this strategy of storage and retrieval and generously sharing it.

returns a cell array containing only the trials in which the condition was 1 or 2.

getCellVals

Another function which you may find useful is getCellVals. This function takes in two arguments, the dataCell and a string specifying a given parameter (note that this is not a condition – it is merely the parameter itself). It then returns a 1xnTrials array containing all of the values of that parameter. So, for example, the following code,

```
getCellVals(dataCell,'maze.leftTrial');
```

returns all of the values of maze.leftTrial.

Getting Started – Process Data

While the software provides a framework, the information which will be displayed in the table is up to the user. It is designated in two files: processDataCell.m and convertToTableCell.m. processDataCell actually processes the data, while convertToTableCell takes the processed data and converts it into a table format.

processDataCell

This function takes in data (an array saved on every iteration of a given software) and dataCell (a cell array of length nTrials in which each cell is a structure containing all relevant data). In this section, the user can input any data which should be processed. **Note that only variables saved in the procData structure will be saved**. Anything else only exists within the function.

Importantly, for processing to apply to a subset of trials, the dataCell must be used either exclusively or to parse the data array. **Any operations performed on the whole of data will not take into account any trial filtering performed in the GUI, resulting in awry calculations.**

If you need the session time, it is stored in procData.sessionTime as the time of the session in minutes.

convertToTableCell

This function converts data processed by processDataCell.m and converts it to a table format. Each section corresponds to a section of the table whose visibility can be modified by the table view listbox in the gui. The special section provides a space for any special parameters as designated by the current maze.

Any variables created in processDataCell.m must be included here to be displayed properly. This is divided into four sections: General, Conditions, Timing, and Special. The visibility of these portions of the table can be easily modified using the listbox in the GUI. This prevents having too many variables in the table at any given time. The table structure contains four fields corresponding to each section

To include variables in the table, use the format below. There are two fields that must be filled for each variable: data and names. Each category (general, etc.) is a nonscalar structure array with as many fields as there are variables. Thus, the first portion of the assignment, which grows the structure is essential. Assign variables calculated in processDataCell.m and stored in the procData array using this format. The names field is a cell array of strings corresponding to the row names of each variable in the table. Order matters and must match the order in which variables are defined in the data section.

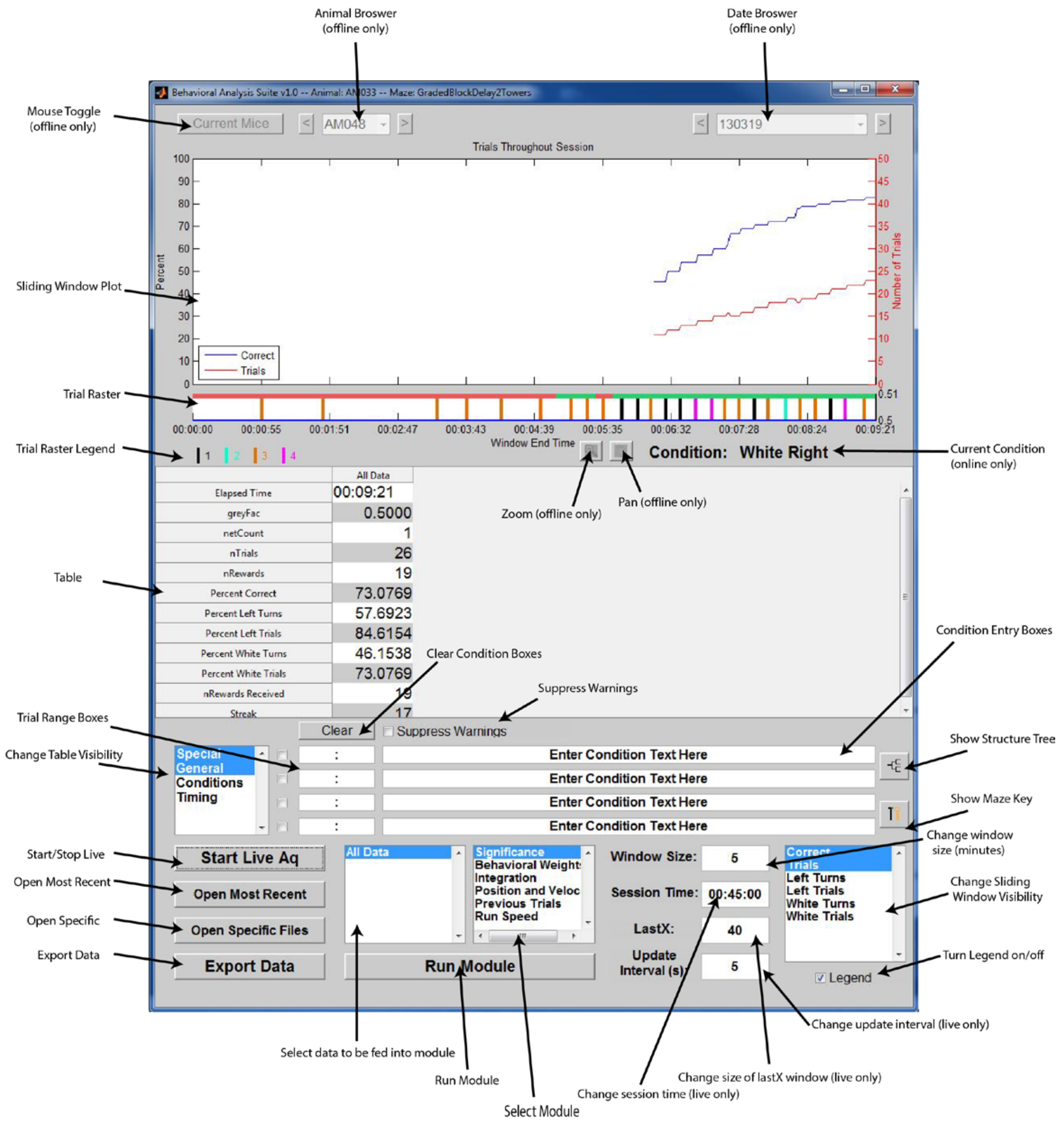
Below is an example of the definition of two variables: procData.nTrials and procData.nRewards to the table:

```
table.general.data(size(table.general.data,1)+1,1) = procData.nTrials;
table.general.data(size(table.general.data,1)+1,1) = procData.nRewards;
table.general.names = {'nTrials','nRewards’};
```

In the special section, the user will likely want to define specific conditions under which to display fields. For example, in a specific task, there might be a variable one wishes to display. If statements using 'isfield' are useful for this. See the below example:

```
if isfield(procData,'greyFac')
    table.special.data(size(table.special.data,1)+1,1) = procData.greyFac;
    table.special.data(size(table.special.data,1)+1,1) = procData.netCount;
    table.special.names = {'greyFac','netCount'};
end
```

Using the Software



	All Data
Elapsed Time	00:09:21
greyFac	0.5000
netCount	1
nTrials	26
nRewards	19
Percent Correct	73.0769
Percent Left Turns	57.6923
Percent Left Trials	84.6154
Percent White Turns	46.1538
Percent White Trials	73.0769
nRewards Received	19
Streak	17

Start the software

To start the software, type “startBAS;” into the Matlab command window.

If you find yourself using the same user configuration file every day, you can start the software silently by putting the name of the user configuration file as the first argument of startBAS. This will prevent the ‘Select User Configuration File’ dialog from appearing.

Ex. `startBAS('userData_AM_PC_ViRMEn.txt')`

Online Analysis

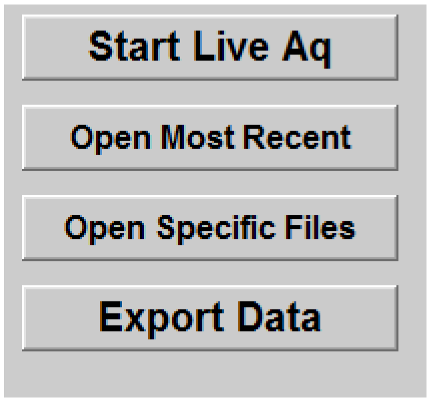
To use the online analysis mode, press “Start Live Aq.”

Open Most Recent

This button will open a dialog displaying the names of each mouse in the path. Select the mouse you want to analyze to open the most recent data for that mouse.

Open Specific Files

This button will open a dialog starting at your data path. From here, you can select files to analyze. You may select more than one file to analyze multiple files together.



Add Files

The add files button allows the user to add new files to the given dataset. Data will then be combined across the different files and analyzed together. Only available in offline mode.

Open New

Open New functions exactly like the open files button. It deletes any data currently displayed and allows the user to open an entirely new dataset.

Trial Range Entry

To limit analysis to a subset of trials, enter a trial range in this box. To limit analysis to trials 10 to 40, enter ’10:40’. You can also analyze the lastX trials of a session by entering ‘-X.’ For example, to analyze the last 30 trials of a session, enter ‘-30.’ Finally, you can easily exclude the first X trials of a session by entering ‘X+1’. For example, to exclude the first 10 trials of the session, enter ‘11’. The software will automatically read this as ’11:end’. You can turn on or off a trial range by checking or unchecking the box next to it. By default, the trial range entry box first narrows a trial range, and then processes conditions. For example, if the trial range is ‘1:10’, and the condition is ‘maze.leftTurn == 1’, it will return all of the trials between 1 and 10 in which the mouse turned left. If you want to only look at the lastX trials which match a given condition (ex. the last 5 trials in which the mouse turned left), regardless of a specific trial range, input ‘-X=’. Importantly, this can only be used if the condition entry box is not empty. Otherwise, it will throw an error.

Condition Entry

To show a subset of trials, enter a conditional string (described above in the “Getting Started – Understanding Conditions” section) and press enter. This will pop-up a new column in the table corresponding to that condition. You can turn on or off a condition by checking or unchecking the box next to it. This can also be used together with the trial range box. For example, one can analyze all the trials between trial 10 and 50 in which the mouse turned left by entering ’10:50’ in the trial range box and ‘result.leftTurn==1’ in the condition entry box.

Clear

This button will clear all trial range and condition entry boxes and uncheck them.

Suppress Warnings

When checked, this will suppress any warnings related to the trial range/condition entry boxes. Instead of an error, if an incorrect string is entered, the software just won’t display subset data. This should be unchecked until you’re familiar with the software.

Structure Tree

The structure tree button displays a figure showing the first three levels of the dataCell structure. This is mostly useful as a reference when entering conditions.

Maze Key

The maze key button displays a table of all the mazes for every mouse. Selecting a date will create another window which displays the other dates that same mouse ran the selected maze. This is mostly useful as a reference when opening multiple files.

Table Visibility

The visibility of different table sections can be easily modulated via the table visibility listbox. Simply select the categories that should be displayed. Multiple categories can be displayed at once.

Window Size

The sliding window size is specified by the window size text entry box in the GUI. This value corresponds to the size of the sliding window in minutes. To change the value, enter a new value and press enter.

Session Time

This is simply a check for the user. When the elapsed experiment time exceeds the time present in the session time box, the text entry box turns red, indicating that the session is over. This section is only available in online mode.

LastX

This text entry box allows the user to change how many of the most recent trials should be analyzed. This section is only available in online mode.

Update Interval

This text entry box allows the user to change the update interval (in seconds) of the data during online analysis.

Sliding Window Visibility

This listbox allows the user to change the visibility of curves on the sliding window plot. Multiple curves can be selected.

Sliding Window Plot

This plot shows a sliding window of the mouse's performance. Percent Correct and Trials Per Minute are set by default. To modify visibility of window variables, use the sliding window visibility listbox. To add other variables, read the user configuration file section.

Trial Raster

The trial raster displays each trial as a raster whose color indicates the condition number. The legend for the trial condition is located on the left below the raster. Additionally, the color above the raster tick indicates whether or not the mouse got the trial correct (green for correct, red for incorrect). To view the details of a given trial, simply click on the trial raster plot to display the structure and values corresponding to that trial.

Legend Check

This checkbox modulates the visibility of the sliding window legend.

Run Module

The run module button takes in information from the two listboxes above it to run a module. The listbox on the left designates which subset of data should be fed into the module. The listbox on the right allows the user to select the module to run.

Mouse Toggle

The mouse toggle allows the user to change which folder the animal browser looks in. The two options correspond to the current and archived folders designated in the user configuration file. This is only available in offline, single-file mode.

Animal Browser

The animal browser allows the user to quickly change between different animals in the same folder. Changing to an animal automatically brings up the most recent file. This is only available in offline, single-file mode.

Date Browser

The date browser allows the user to quickly change between different dates of the same animal. This is only available in offline, single-file mode.

Export Data

Export Data brings up another window allowing the user to export a subset of the data to the Matlab workspace, including the processed data, or to save the sliding window figure in a variety of formats, including .eps for Adobe Illustrator.

Advanced Features – Modules

New modules can be written and incorporated into the GUI without modifying the GUI code itself. To do this, write a module as a callback (must include the first two callback arguments) which takes in one or more of the following arguments: data (the per iteration array), dataCell (the cell array), exper (used in ViRMEn experiments), dataCellInd (a logical with all the data currently being analyzed indicating whether each trial has been included in the subset analysis. If no trial filtering occurs, it will be all 1s), or guiObjects (the handles of all of the original GUI objects). Next, create a text file with two parameters separated by the '|' delimiter. The first value should be the name of the module to be displayed in the GUI. The second parameter should be a string which calls the module function written above. Put all of the functions necessary for the module to run in a single folder titled "module_MODULENAME" and copy it into the "Modules" folder. For examples, look at the modules included in the software like significance.

Advanced Features – Exporting

You can export figures or data easily from the BAS. However, to export figures to pdf or eps formats, you need to install two pieces of software. The first is called Ghostscript (http://www.ghostscript.com). The second is only required for eps formats and can be found here: http://www.foolabs.com/xpdf. To install xpdf, copy the xpdf folder into C:\Program Files. Exporting should now work. Note that while some modules include the ability to export, not all do. For an example of how to export, look at the exportFig subfunction of integration_CALLBACK.m in the integration module.

Advanced Features – Silent Start

Normally, BAS requires the selection of a user file every time it's launched, but a silent start is also possible. To launch BAS silently, put a string containing the name of the user file, including the extension, as the first input to the startBAS. The full path is not necessary as long as the user file is located in the User Processing folder. Additional settings within the BAS can be modified using the following name-argument pairs:

SuppressWarnings -- set to true to prevent warnings from appearing. Default is false

WindowSize – number of minutes over which the sliding window is calculated. Default is 5.

SessionTime – string in format, 'HH:MM:SS', for the max session time. Default is '00:45:00'

LastX – number of lastX trials to look at. Must be a positive integer. Default is 40.

UpdateInterval – number of seconds between live updates. Default is 5.

For example,

```
startBAS('userData_AM_Rig_1_ViRMEn.txt','SuppressWarnings',true)
```

will launch the BAS using the user file ‘userData_AM_Rig_1_ViRMEn.txt’, and with suppress warnings checked.