**Table of Contents:**

1. Table of Contents……………………………………………………………………………………………………………..1
2. Introduction……………………………………………………………………………………………………………………..2
   1. Author’s note
   2. General set-up of functions
   3. TWDB
3. Classification methods (striosomality/MSN neuron type)……………………………………………….. 3
   1. Striosomality:
      1. ah\_find\_peakOrValley
      2. ah\_maximize\_consec\_suibset\_sum
      3. ah\_baseline\_bootstrapping
      4. ah\_striosomality2
   2. MSN Classification:
      1. sqr\_neuron\_type
4. Spike/burst analysis – gathering data………………………………………………………………………………. 5
   1. ah\_build\_spikes\_array
   2. Bursts:
      1. ah\_find\_trial\_bursts
      2. ah\_build\_bursts\_array
   3. ah\_get\_ses\_evt\_timings
5. TWDB functions……………………………………………………………………………………………………………… 7
   1. Creating TWDB
      1. twdb\_addsingleneuron
      2. twdb\_addneurons
      3. twdb\_addneuronsbulk
   2. Injecting data into TWDB
      1. ah\_ratID\_and\_distance\_intoTWDB
      2. ah\_injectDataIntoTWDB
      3. ah\_striosomality2\_grade\_twdb
      4. ah\_strioProjecting\_grade\_twdb
      5. sqr\_neuronType\_grade\_twdb
      6. sqr\_responder\_twdb
      7. dg\_readEvents
   3. Extracting data from TWDB
      1. twdb\_lookup
      2. ah\_extractDataFromTWDB
6. Binning functions…………………………………………………………………………………………………………….. 10
   1. ah\_fill\_spike\_plotting\_bins
   2. ah\_fill\_burst\_plotting\_bins
   3. ah\_fill\_burst\_spike\_plotting\_bins
7. Figure generation functions…………………………………………………………………………………………….. 11
   1. ah\_plot\_double\_aligned\_population\_analysis
   2. ah\_plot\_unsplit\_maze
   3. ah\_plot\_split\_maze
   4. ah\_barsWithErrors
8. Figure/script organization……………………………………………………………………………………………….. 12

**Introduction**

**Author’s Note:**

The goal of this readme is not to explain every detail on how each function in this small library works; rather, my goal here is to inform the reader how, on a macro scale, my functions work together to achieve nice results. I will also include some cautions in this readme of things to be certain of and other things to avoid, but in general, for specific information on how the functions work (in detail), please read the comments on the functions themselves. I spent a fair amount of time writing them and like to think that my efforts weren’t wasted. Without further ado, I will now start explaining my library. Good luck!

-Adam Hood

**General set-up of functions:**

The functions in ah\_lib are set up in two overarching stages: data collection, and data processing.

“Data collection” is a misnomer – within the functions of ah\_lib there is no actual data collection. What I name data collection is in fact a first step of processing, in which I convert the raw data (in this case, timestamps of spikes, events) into a form that is useable by the rest of the functions. The data is then stored in TWDB so that it may be accessed quickly and efficiently for the second stage. This stage is vastly more expensive and time-consuming than the data processing/analysis stage, but if done properly, need only be run a single time. Thereafter, all the processed data is already saved in a way that can be used for analysis from these functions.

Data collection functions will be presented in sections 2 – 4b.

The second stage consists of data processing and analysis. It can further be broken down into a pipeline: first, identify the neuron(s) to analyze. Next, extract the data for these neurons from TWDB. This can be done manually if necessary, but is much easier/more space efficient if it can be done automatically. Third, a type of analysis must be selected and the relevant function can be run to get “bins”, which are in turn fed into any of three plotting functions.

**TWDB:**

TWDB is a tool essential to this analysis. It is a Matlab struct array with an entry for each neuron. The fields of TWDB include pointers to the location of raw spike data, information on the neuron itself, and information injected by the functions in ah\_lib. The TWDB file, due to the amount of data stored within, is rather large (800 MB), so I do not recommend attempting to use this file directly on computers with smaller amounts of RAM (8 GB or more required at least). While complete proficiency with this tool is not required for understanding of ah\_lib, it is worth having some familiarity with it, and being able to perform lookups with it.

Credit to Tim Wall (Mit ’14) for the development of this tool and the methods associated with it.

**Classification Methods (striosomality, MSN neuron type)**

**Striosomality:**

In our striosomality methods, we attempt to find a template for a response to PL stimulation unique to striosomes (and in particular, not shared by matrix neurons). Further, given a response, we determine its significance relative to the baseline of the same neuron.

**2a-i: ah\_find\_peakOrValley**

Key to any analysis of a response aligned to stimulation, we need a peak/valley finding function. Our template of striosomality is a peak followed by an inhibition which is itself sometimes followed by a rebound. This function is used to identify all three phases.

This function takes as inputs an array of all the relevant spikes as well as the baseline firing rate. To acquire a list of all relevant spikes or compute baseline firing rate, see 3a (ah\_build\_spikes\_array). It also takes an argument identifying whether we are searching for a peak or a valley. It outputs the largest peak found, optimizing based on “area” – a discrete integral taken between any pair of spikes. In a peak, we maximize the positive value whereas in a valley, we maximize the negative value. Specifically, area is defined as **#{spikes} – E[#{baseline spikes}] = #{spikes} – baseline\_firing\_rate\*Δt** where **E** represents expected value, **Δt** represents time between two spikes, and **baseline\_firing\_rate** is the baseline firing rate. This algorithm proves efficient as for any **i<j<k**, if **Aij** represents the area between spikes **i** and **j**, then **Aik = Aij +** **Ajk**. Thus, we can optimize by computing the area between every consecutive pair of spikes and find the maximum consecutive subset sum. The function then returns the indices of the optimal peak/valley.

**2a-ii: ah\_maximize\_consec\_subset\_sum**

This is the principal algorithm employed the previous function. It takes in an array an input and outputs the start and end indices of the maximum consecutive subset sum of the array. Employs Kadane’s algorithm (<http://en.wikipedia.org/wiki/Maximum_subarray_problem#Kadane.27s_algorithm>)

**2a-iii: ah\_baseline\_bootstrapping**

The other cornerstone of any response analysis is a measure of significance. We need to ensure that a rise in activity is significantly stronger than any such rise in the baseline. This function makes an assumption of Poisson spike activity. Based on a series of randomly generated Poisson events, we search for a peak/valley under identical parameters as our search in the response time window. From this we compile a distribution, and based on this distribution, we can determine the significance of the response.

Inputs are the size of the baseline window to use (in seconds), the baseline firing rate, and the number of samples to compute. The output is an array with the area of the optimal peak/valley in each sample. In the event that no peak/valley was found in a given sample, an area of 0 will be entered into the distribution. Note that this function may cause external divide by 0 errors in a specific instance: if the baseline firing rate is sufficiently low and the window very small, then the distribution will be completely full of zeroes. In these instances, another method must be used.

**2a-iv ah\_striosomality2**

This function compiles all the ideas of the previous three into a template designed specifically for giving some grade on the “striosomality” of a given neuron. The template is “inhibition starting before 50ms, peak between stimulation and beginning of inhibition, rebound peak following inhibition”. Not all neurons follow this template; some only exhibit one or two of the phases; the function assigns them a grade nonetheless.

The function, as described in the template, splits the response into three phases. First it maximizes a peak in the 0-200ms window, ensuring that the inhibition starts at latest by 50 ms. It then searches for peaks both before the inhibition and after the inhibition. For each phase of response found, it uses baseline bootstrapping to compute a distribution. Since these distributions are independent, they may be combined easily, and a mean and standard deviation can be computed from the combined distribution. We thus arrive at a z-score grade for our response, by which we can later set a threshold as needed.

Inputs are the spikes of interest (aligned to stimulation) and a baseline firing rate. Both can be computed by **ah\_build\_spikes\_array (3a)**. Outputs are a “type” – a classification based on the phases that the response exhibit, the “grade” – the z-score computed previously, and some data from each phase – for the purpose of future data analysis/thresholding if needed. The types are:

0 – None of the three phases exhibited. Caution: by default, a neuron must be low firing to have this grade.

1 – Only the initial peak exists; no inhibition or rebound

2 – Only inhibition exists; no initial peak or rebound peak

3 – Inhibition and rebound exist, no initial peak

4 – Initial peak and inhibition exist, no rebound

5 – All three phases are exhibited in this response

The data keeps the start and end of each phase as well as the area and the number of spikes.

**MSN Classification**:

High-firing neurons (FSI) exhibit different firing patterns from other DMS neurons, thus it becomes desirable to classify these neurons to analyze the different responses.

**2b-i: sqr\_neuron\_type**

This function classifies neurons, giving one of five grades. Grade 1 neurons are used as FSI neurons and grades 3-5 can be used as MSN neurons. Occasionally grade 4 neurons will exhibit firing patterns of TAN neurons. This is accomplished by Gaussian clustering. It is essential that somewhere within the directory the matlab data file neuronTypeData.mat exists, as it is an input of the function. This data allows for the immediate classification of any DMS neuron; it was computed on a training set.

Credit to Qinru Shi (MIT ‘2015) for the development of this function.

**Spike/Burst Analysis, Gathering Data**

**Spike Analysis**

This the classical method of data analysis; nothing particularly fancy here.

**3a – ah\_build\_spikes\_array**

This function is probably the most essential piece of my library. It performs a relatively simple task that is applied in every facet of putting data into TWDB, for all that is a relatively simplistic function. The purpose of **ah\_build\_spikes\_array** is to build a cell array in Matlab with one entry for each trial of a given neuron and session. The entry contains a vertical array of the time stamps of every spike within a window of interest about the event to which we align. This can be aligned about a stimulation event or a door open event (or any other event for that matter). If we do not wish to keep separate spikes from different trials, they can be collapsed into a single array with the cell2mat command in Matlab (and sort, if necessary). This function is used to insert the spikes array itself into TWDB, as well as in **ah\_striosomality2** and in the computation of Striosome-projecting PL neurons.The spieks array is also used computing bursts.

This function takes the array of absolute timings of spikes and the array of events as inputs, as well as certain parameters such as an alignment event, and outputs the spikes array as well as, if needed, the baseline mean and standard deviation firing rate.

**Burst Analysis**

This type of analysis up to this point is not widely used; it was largely invented by myself and Alexander Friedman, at least in the context of its use in this project. A burst is defined as any set of consecutive spikes whose distance from each other is less than the inter-spike interval. (e.g. instantaneous firing rate greater than the baseline firing rate). The two functions presented here deal with finding bursts and building an analogous array to the spikes array for bursts.

**3b-i: ah\_find\_trial\_bursts**

This function is rather simple; it loops through all the spikes of a trial once and lumps together any group in which every consecutive spike is within a certain distance of the next spike. Inputs are an array of sorted spike times; the output is a list of the start time, end time and number of spikes in each burst as well as the indices of the start and end of each burst.

**3b-ii: ah\_build\_bursts\_array**

This function applies **3b-i (ah\_find\_trial\_bursts)** on a full neuron/session level. It compiles a cell array analogous to the spikes array for bursts; each trial entry lists all bursts found (Note: thresholding occurs at a later stage in the analysis). Each burst is listed by a start time, end time, and number of spikes found in the burst.

**Other Necessary Data**

**3c – ah\_get\_ses\_evt\_timings**

Probably the least glamorous function in this library, this function goes through the events file, identifies the events belonging to any given trial, and aligns them to the event of interest (usually click). Thus, for any given trial of any given sessions, assuming no equipment failures, we have a timeline of the events of each trial. To understand the exact format of the array computed, see the comments of the function; this array is crucial to the later analysis, but contains none of the actual data we use.

**TWDB Functions**

These are all the functions that interact directly with TWDB – first in setting up the TWDB file and later in extracting relevant data from TWDB. In talking about TWDB we do need to be careful to distinguish different databases: we have a training set, and its TWDB file, and we have a primary directory, and two TWDB fields associated with that. If not specified, the TWDB file in question is the primary file. Most TWDB injection functions are designed for one database or the other.

**Creation of TWDB**

The following functions build the initial TWDB file. Note that for these functions to work to their full potential, session directories MUST follow a specific naming scheme; extra information in the session directory name is acceptable, so long as the information these functions require exists:

1. Session ID – yyyy-mm-dd\_hh-mm-ss. Time stamp of the beginning of the session. Each session ID is distinct, hence it is crucial that it exists as a tag for each session.
2. Task designation and, if relevant, concentration of the mixture drink immediately following the task name – comb = cost benefit, tr = benefit benefit, eqr = cost cost, and negacomb = no conflict cost benefit. Without these identifiers, the TWDB functions will be unable to identify the type of task.
3. If a session is a laser session, “laser” must appear somewhere in the sessions directory name.
4. It is up to the discretion of the user whether or not to include “bad” sessions in the database e.g. sessions in which the rat is heavily biased, or the rat didn’t run, etc.

**4a-i twdb\_addsingleneuron**

Adds a single neuron to TWDB. The user should never need to call this function. NOTE: while the user will never need to call this function, it will probably need to be adjusted. Because they will naturally vary from rat to rat, especially across different labs/contexts, the regions that our tetrodes record from are hard-coded into this function. They WILL almost certainly needed to be adjusted by the user to match his or her own recordings.

**4a-ii twdb\_addneurons**

Adds a single session to TWDB. The user may require this if a given directory of sessions has already been added to TWDB but a new sessions exists in said directory and needs to be added to TWDB.

**4a-iii twdb\_addneuronsbulk**

In general, directories should be organized in some manner. In our case, we organized by rat; this function will add a full directory of session directories to an existing TWDB file or create a new TWDB file with that directory.

**Injecting Data Into TWDB**

The next series of functions insert data into TWDB – both for the sake of later selection AND later extracting data from TWDB. The difference between the types of data will be made more clear later, when I discuss lookups and extraction of data from TWDB.

**4b-i ah\_ratID\_and\_distance\_intoTWDB**

This function is necessary only for a training data set; if used by user, it WILL need to be adjusted. This function adds distances for every tetrode of every striosome of the training set, as well as the ID of the rat in question. The rat ID is also added by twdb\_addsingleneuron now. As the tetrode distances are hard-coded into TWDB, they WILL need to be adjusted.

**4b-ii ah\_injectDataIntoTWDB**

This function inserts the data into the primary TWDB file that is later used in analysis. For each neuron, it saves (1) a spikes array (aligned to click event in each trial), (2) a bursts array (same alignment), (3) an array of event timings in the session (same alignment), (4) some data on the baseline firing rate of the neuron. The only relevant data is the first entry of the data array which is the baseline firing rate of the neuron; all the other data types have been explained previously in section **3**. This also adds the index of each entry to TWDB if it hasn’t already been added. The index is *crucial* for the functions of ah\_lib to work together properly – when we search in TWDB, the index field is the one our search should output.

**4b-iii ah\_striosomality2\_grade\_twdb**

This function assigns striosomality type, grade, and data to each neuron of TWDB, as described by the function **2a-iv (ah\_striosomality2)**

**4b-iv ah\_strioProjecting\_grade\_twdb**

This function inputs data into TWDB on the response of every neuron to dms stimulation. It puts the number of spikes in the 1-10 ms window into TWDB as well as the ratio of the firing rate of this period compared to the baseline firing rate. We can then set criteria based on these values to determine whether or not a PL neuron is striosomal-projecting. Note that every neuron generally either has no response to dms stimulation or a very strong one, hence these thresholds are easily set (currently, grade ≥ 5, spikes ≥ 10).

**4b-v sqr\_neuronType\_grade\_twdb**

This function assigns every neuron in TWDB a neuron type based on the neuron classification function (**2b-i, sqr\_neuron\_type**). This can be used in neuron selection in **4c-i.**

**4b-vi sqr\_responder\_twdb**

This function assigned every neuron in the training set of TWDB a designation “Up”, “Down”, or “None” for the difference between the responses to electrical versus electrical AND optical stimulation (based on a t-test), as well as putting other data such as mean inter-spike interval and firing rate that are used in the neuron type classification. This data is generally used to make comparisons between MSN and FSI neurons.

**4b-vii dg\_readEvents**

This reads .nev extension event files; needed for the training data set. Is a directory including all necessary subfunctions.

**Extracting Data From TWDB**

Here begins the second stage of the analysis provided by these functions. Once all prior functions have been used in the creation of a TWDB file, the user can then begin some analysis. The next pair of functions allow the user to select neurons and then extract the data from TWDB for these specific neurons in a form useable by the binning functions (Section **5**) for the next stage of analysis.

The previous group of functions inserted two primary types of data into TWDB: (1) selection data and (2) analysis data. Selection data allows us to select the neurons we want (examples: Cost Benefit FSI neurons, Cost Cost striosomal-projecting PL neurons, etc.). Analysis data is the data that is then extracted from TWDB for the purpose of the next group of functions.

**4c-i twdb\_lookup**

As mentioned above, this is the method that allows us to select neurons from TWDB with the help of our selection data. **twdb\_lookup** supports any number of simultaneous lookups through TWDB; in particular, it support two specific types of lookups: key lookups and threshold lookups.

Key lookups ensure that all neurons found have a specific value for a specific field, usually to find neurons of specific tasks or regions of the brain (e.g. **‘key’, ‘taskType’, ‘CB’** to find cost benefit neurons, **‘key’, ‘tetrodeType’, ‘dms’** to find DMS neurons).

Threshold (grade) lookups allow us to ensure that certain parameters meet threshold criteria. This allows us to select classifications of neurons (neuron type, etc.), as well as thresholds on other parameters (striosomality grade, etc.). For instance: **‘grade’, ‘sqr\_neuron\_type’, 3, 5** to find MSN neurons; **‘grade’, ‘striosomality\_grade’, 2, NaN** to find striosome neurons with grade (z-score) at least 2.

Note that in almost every instance, the field that should be designated to be outputted will be the **index** field – a field of TWDB that contains the index of each entry (in string form).

Sometimes a lookup by itself will not be sufficient to select the neurons we want; we may need to loop through the neurons we have selected to add another layer of selection. We cannot select by data stored in arrays stored in TWDB, hence this must be done separately. Alternately, for preliminary analyses, neurons can also be selected by hand by looping through the neurons and looking at their individual traces.

**4c-ii ah\_extractDataFromTWDB**

Once we have determined the indices in TWDB of the neurons we wish to select, this function allows us to extract from TWDB the data necessary to proceed to the next level of analysis. This function provides four primary outputs used in analysis: a spikes array, containing the spikes of every trial of every neuron we selected, a bursts array, containing the bursts of every trial of every neuron we selected, a session-event timings array, containing the timings of all the events of every session, aligned to the click event of their trial, and an array with one row for every neuron containing data about the neuron; indices in both the session event timings array and the spike/bursts array, as well as information about the baseline firing rate. These serve as inputs for the binning functions of Section **5**.

**Binning Functions**

The next stage of the analysis are the binning functions. We have three binning functions; inputs are nearly identical in all cases, so I will discuss the functions here and list them later.

These functions take the raw data, after it has been collected, aligned, and extracted from TWDB, and converts it into a series of bins. These bins represent space within the maze, and are extended beyond the edges of the maze with the same spatial scaling. Many options are available as parameters of the function: we can select specific trials, or groupings of trials (for instance: chocolate choice trials only: see comments on selection\_criteria input). We can adjust parameters relevant to the plotting, such as number of bins and the location on a **0** to **1** scale of the click/lick events in our bins. For a more in depth description of the possibilities with the inputs to these functions, see the beginning comments.

**5a-i: ah\_fill\_spike\_plotting\_bins**

This function processes spikes into bins

**5a-ii: ah\_fill\_burst\_plotting\_bins**

This function processes bursts into bins

**5a-iii: ah\_fill\_burst\_spike\_plotting\_bins**

This function processes spikes that appear in bursts into bins

**Figure Generation Functions**

Once we have sorted our data into its bins, all the hard work is done; it’s a mere matter of aesthetics and semantics to draw figures. Once again, to fully have a sense of the parameters involved in plotting, please read the comments of the functions in question.

**6a-i: ah\_plot\_double\_aligned\_population\_analysis**

This function plots the trace of the selected trials of all the selected neurons with the selected type of analysis.

**6a-ii: ah\_plot\_unsplit\_maze**

This function plots the unsplit maze of one or three groups of bins. There is the option of separating choices on the arms of the maze, or simply using the same bins for both arms. Generally not optimal for separating two different groups of trials; use the following function for that.

**6a-iii: ah\_plot\_split\_maze**

This function plots the split maze of two groups of bins. Best for separating such things as chocolate and mixture trials.

**6a-iv: ah\_barsWithErrors**

This is the bar plotting function; can be applied well outside of the scope of these libraries and functions. Distinct from the previous three functions in that it does not take the bins of the previous functions as an input; data requires slightly more processing to be ready for this function

**Figure/Script Organization**

The figures and scripts are organized into a final directory of ah\_lib (figures and their scripts) and further organized into a group of directories. The directory titles general describe the type of figures contained in said directories: for instance, bar graph figures (and associated data and scripts) are contained in the directory “Bar\_graphs”; all mazes and traces of striosome populations are contained in the directory “Striosomes”, etc. The script generating any given figure is stored in the same directory as the figure itself, as is the data associated with said figure.

The scripts themselves are broken down into sections; the heading of each section describes the purpose of the code in that given section. The most important parts of these scripts for a user are the setup section, in which the directory paths may need to be adjusted, and the final section, which may or may not be necessary for the user. The final section closes all figures and updates the local Dropbox repository on the user’s computer. This will fail if there is no such local Dropbox repository, and may need to be adjusted accordingly. Otherwise, the scripts should need little adjustment unless the user wishes to play with the specs of the figures presented themselves.