**Data Analysis Tools Documentation**

The github repository at <https://github.com/JaneliaSciComp/G4_Display_Tools> has a suite of data analysis tools.

In the typical use case of these tools, there are two files you need to worry about: “DA\_plot\_settings.m” and “create\_data\_analysis\_tool.m”, both found at G4\_Display\_Tools/G4\_Data\_Analysis.

Table of Contents:

1. [New Format](#newFormat)
2. [Plot appearance settings](#appearanceSettings)
3. [Data analysis settings](#dataSettings)
4. [Running a typical analysis](#running)
5. [Adding new modules](#addModule)

**New Format:**

The files G4\_Process\_data\_flyingdetector (in DA\_Data\_Analysis/data processing) and G4\_Plot\_Data\_Combined (in DA\_Data\_Analysis/data plotting) have been changed slightly from previous versions. The processed .mat file produced is formatted differently, and this format is required to work with the data analysis tools described below.

For your convenience, there is a script called “convert\_processed\_file.m” in DA\_Data\_Analysis/support which will convert older processed files to this new format. Be sure to open this file and update the processed\_file\_name variable. If this does not match the filename of the processed file you want to convert (excluding the extension), it will not work.

You can run the convert\_processed\_file.m file and it will prompt you to browse to the experiment’s results folder. Alternatively you can pass this in by typing “convert\_processed\_file(folderpath)” in the matlab command line. You can simply pass in the path to your Results folder. The script will go through every fly folder and convert each processed file.

Alternatively, you may run the new processing file on old data to produce a new processed\_data file.

**Plot Appearance Settings:**

The first step is to ensure all settings and user-defined values are correct. The first file you should check is “DA\_plot\_settings”. It contains a significant number of settings related to the appearance of any plots generated when you run the data analysis as well as how they’re saved. Let’s do a quick overview of these settings:

The settings are split into seven different structs.

1. normalize\_settings.
   1. These are all settings related to the normalization of the data.
2. histogram\_plot\_settings.
   1. These are settings related to plotting basic histograms. These are not the same as the closed-loop histograms.
3. histogram\_annotation\_settings.
   1. These control how the histograms from 2 are annotated – font, line type, and many other things.
4. CL\_hist\_plot\_settings.
   1. These are the settings for closed-loop histograms
5. timeseries\_plot\_settings.
   1. Contains appearance settings for the timeseries plots
6. TC\_plot\_settings.
   1. Appearance settings for tuning curves
7. save\_settings.
   1. These settings affect how figures are saved to .pdf files. They do not contain the save path itself, we will cover this later.

Most of these settings will likely stay the same from one run to the next. I have tried to put all the settings that will change regularly in the actual tool file, which we’ll get to next. The DA\_plot\_settings may be more heavily adjusted if you want to produce nice looking figures with custom colors, line widths, etc.

**Data Analysis Settings:**

The second step is to update the user-defined settings in “create\_data\_analysis\_tool.m.” When you open this file, the format may look strange if you are not used to object-oriented programming. It will look something like this:

Classdef create\_data\_analysis\_tool < handle

Properties

[list of properties]

End

Methods

[A number of functions]

End

End

The user-defined settings are inside the first function under methods. If you click the small minus sign to the left of properties, the properties section will minimize and the user-defined settings will now be near the top of the page. You may also notice most variables are formatted as self.variable\_name rather than just variable\_name. This is just to indicate the variable belongs to this class. You do not have to worry about it.

These settings should be updated every time you run an analysis. Here’s an overview of the settings in this section:

1. Self.processed\_data\_file.
   1. This is a variable of type string. It should be the name of your processed data files. For example, if the experiments you are analyzing have processed files called G4\_Data\_Processed.mat, then this variable should be set equal to ‘G4\_Data\_Processed’. Please make sure all fly folders in this analysis have their processed files named the same way. Any that are named differently will not be included in the analysis.
2. Self.save\_settings.save\_path.
   1. This is a variable of type string indicating the path at which you want to save your figures.
3. Genotypes.
   1. This is an array of strings. Create one array item for each genotype (or group) being compared in this analysis.
4. Self.histogram\_annotation\_settings.date\_range.
   1. This is part of the histogram annotation settings from DA\_plot\_settings.m. But this string should be changed regularly. It is the range of dates which the data is from and will be part of the histogram labels. Please make sure this is up to date.
5. Self.OL\_conds, self.CL\_conds, self.TC\_conds.
   1. These variables allow you to customize how you’d like your trials arranged in your figures. If you want multiple figures, then make this variable a cell array, with each element being a regular array. For example:  
        
      self.OL\_conds{1} = [1 3; 5 7; 9 11; 13 15]  
      self.OL\_conds{2} = [17 19; 21 23; 25 27; 29 31]  
      self.OL\_conds{3} = [ 33; 35];  
        
      This would create three timeseries (open loop) figures. The first figure would be a 2x4 grid of plots laid out like so:  
        
      Trial 1 Trial 3  
        
      Trial 5 Trial 7  
        
      Trial 9 Trial 11  
        
      Trial 13 Trial 15  
        
      The second figure would look the same, but with different trials. The third figure would have two plots of trials 33 and 35, one below the other.  
        
      Histograms (CL\_conds) and tuning curves (TC\_conds) can be laid out the same way.  
        
      Please note that this setting is optional. You may simply leave the setting as self.OL\_conds = []; In this case, the program will create a default layout of the included plots.
6. self.OL\_conds\_durations.
   1. This array should be set up exactly the same as your self.OL\_conds array, except each element, instead of being the condition number you want in that place, should be the duration of that condition number. These durations will be used to set the x axis limits of each graph so trials of different length are not all forced on to one size axis. Please note that this array must be exactly the same size and shape as the OL\_conds array.
7. self.OL\_conds\_axis\_labels.
   1. This should be a cell array with an element for each figure (so it should have the same number of cell elements as OL\_conds). In each cell array element, there should be a 2 element vector with strings corresponding to [x-axis-label, y-axis-label]. These will be used to label the x and y axes in your figure. Note that all plots on a single figure should have the same axis labels.
8. The next setting has a little bit of code to it. This code generates condition names for one’s timeseries plots. Let’s go through it.
   1. Self.timeseries\_plot\_settings.cond\_name = cell(1,81). This should not change other than you should replace 81 with the number of conditions in your protocol.  
        
      patterns = []. This should be an array of strings with the names for each pattern that you’d like in your plot titles.  
        
      There may be other arrays you can iterate through to create plot names. These might include looms, sweeps, or wide-field rotations. They should all contain string elements. Below, you can then use for loops to iterate through these and assign a title to each element in your cell array. Please note you will need to alter the numbers inside each for loop to adjust for your protocol. We are working to make this process more abstracted.
9. After this, the last setting for you to update is the datatypes. Self.CL\_datatypes, self.OL\_datatypes, and self.TC\_datatypes should be set equal to cell arrays of string elements, each element being the datatypes you wish to plot. Possible datatypes are listed in the comments.

This is the end of the user-defined settings. You’re now ready to create your data analysis object. To do this, you will run the create\_data\_analysis\_tool file with a few inputs.

**Running a typical analysis:**

The first required input is exp\_folder. This is a cell array of all paths to fly folders being analyzed. For your convenience, there is a script which will create this for you called “get\_exp\_folder.m” located in G4\_Display\_Tools/G4\_Data\_Analysis/support. Simply open this file and update the results\_path variable to the path of your Results folder. You may also set “convert\_processed\_files” to 1 if the processed files in these folders need to be converted to the new processed file format.

The second input is trial\_options. This is a 1x3 array of 0’s and 1’s representing the presence of a pre-trial, inter-trial, and/or post-trial. Ie [1 1 1] means you have all three. [0 1 0] means you have an inter-trial, but no pre or post-trial.  
  
After this there are multiple optional inputs, or flags, which tell the program what analysis to do. The currently accepted flags are as follows:

* ‘-norm1’ – normalize the data over each fly
* ‘-norm2’ – normalize the data over groups
* ‘-hist’ – plot basic histograms
* ‘-TSplot’ – plot open loop timeseries data
* ‘-CLhist’ – plot closed loop histograms
* ‘-TCplot’ – plot tuning curves

Make sure not to leave out the apostrophes or the dash. Any subset of these can be passed in, in any order.

When you run create\_data\_analysis\_tool, you want to store it in a variable. So your command in the matlab command line might look like:

da = create\_data\_analysis\_tool(exp\_folder ,[1 1 1], ‘-norm1’, ‘-hist’, ‘-TSplot’);

This in itself will not run the analysis. What it does is creates an object, da, with all of your settings stored in it and the options for whatever flags you passed in turned on. You can use this object to double check if everything is correct if you would like. For example, you could now type “da.save\_settings” into the command line to review your save settings. If you forgot to pass in a flag, you could say da.TC\_plot\_option = 1 to retroactively tell it you want to make tuning curves as well. If you forgot to update the colors of your timeseries plot, you could say da.timeseries\_plot\_settings.rep\_colors = [0 0 0; 0 1 0; 0 0 1] and update them.

Once you know there are no adjustments to be made, simply type in the command da.run\_analysis, and this will start the analysis running. Assuming no adjustments after creating your data analysis tool, your commandline command will look something like this:

da = create\_data\_analysis\_tool(exp\_folder, trial\_options, ‘-hist’, ‘-TSplot’);  
da.run\_analysis

This will produce a number of graphs, automatically saving them at the save path you entered. Their filenames will contain all the genotypes being compared followed by a number.

**Adding new modules:**

Coming soon