STATSLAB: A MATLAB Toolbox for Performing the Percentile Bootstrap Test on Electroencephalographic Data

Traditional parametric methods for comparing groups and measuring associations are not robust to violations of assumptions (Wilcox, 2012). Statisticians have been telling us this for the last half century. Such non-robust methods include Student's t-test, ANOVA, and Pearson's r (Tukey, 1960; Huber, 1964; Hampel 1968; Wilcox, 1996a; Cressie & Whitford, 1986; Algina, Oshima, & Lin, 1994). Furthermore, there is no corrective technique (e.g., logarithmic or Box-Cox transformations) that is able to salvage these methods when faced with nonnormality and outliers (Docksum & Wong, 1983; Rasmussen, 1989; Thompson & Amman, 1990). Despite fifty years of evidence against the utility of traditional methods, and no evidence to the contrary, they are still routinely being used in psychology and other disciplines. As a result, avoidable Type I and type II errors continue to continue, and in many areas, psychology is failing as a cumulative science. Classic hypothesis tests and measures of association can be used but only in the supernatural event of normal distributions and homogeneity of variance (i.e., when no assumptions have been violated) (Wilcox, 2012). Fortunately, alternatives do exist which are virtually assumption free, result in increased power, narrower confidence intervals, and reduced type I error rates compared to classic methods (WIlcox, 2012)[1].

One such robust test that is particularly useful for analyzing EEG data is the percentile bootstrap, especially when used in conjunction with trimmed means (Wilcox,

[1] A common complaint is that popular software packages only include functions for traditional hypothesis tests and so researches have no choice but to use them. That is no longer the case as several programs are now available and are easy to use (e.g., Wilcox's robust statistics package for R (2010), ERP PCA toolkit (Dien, 2015), and LIMO EEG (Pernet et al., 2011).

2012; Desjardins & Segalowitz, 2014; Rousselet et al., 2008; Oruc et al., 2011). In the following sections I will give a tutorial on STATSLAB: a software bundle for MATLAB (Mathworks) that uses the percentile bootstrap test to analyze EEG data at the group and single-subject level. All commonly used categorical comparisons are possible with STATSLAB (i.e., single-factor, two-factor, within-subjects, between-subjects, and mixed designs). STATSLAB also includes tests for associations between continuous variables (i.e., robust correlations and locally-weighted scatterplot smoothing). You do not need to have extensive programming experience to use STATSLAB. You should however understand some general statistical concepts (e.g., bootstrapping, linear contrasts, confidence intervals, and factorial design terminology). These topics are covered to some extent in most introductory and graduate level statistics textbooks.

The STATSLAB Pipeline

There are four major stages in the STATSLAB pipeline: extraction, resampling, statistics, and figure plotting. Each of these steps can be executed by simply typing one line of code into the MATLAB command window. I will describe how to use the associated functions in detail and provide numerous examples of the various options that are possible along the way with STATSLAB. To begin it is assumed that you have segmented files produced by EEGLAB (Delorme & Makeig, 2004) with the extension .set. You should have one .set file for each subject in each condition. Initially STATSLAB relies on EEGLAB to some extent so you will need to have both toolboxes in you MATLAB path (file -> set path -> add with subfolders). The STATSLAB software bundle came with a demonstration dataset for you to practice with and explore. The relevant details of the study are as follows: We conducted a sleep deprivation (SD)

study where electrocortical responses were measured during alert (2 hours awake) and sleep deprived (20 hours awake) sessions. Participants performed a gender discrimination task while viewing faces that were either clear or distorted with Gaussian noise (the noise, or lack of noise, made for easy or hard gender discriminations). This was a 2x2 within-within (repeated measures) design. The four conditions were (1) awake, easy decision, (2) awake, hard decision, (3) sleepy, easy decision, and (4) sleepy, hard decision. Many of the following examples and screenshots will refer to this dataset. One more thing before beginning the tutorial, STATSLAB accumulates important information about your study, and your decisions along the way, in a MATLAB structure called STATS. In the demo folder, double click the file STATS_GFA_Analysis to load it into MATLAB. From there you can explore the contents of the STATS structure (by clicking on it) to get a sense of the information it contains. The STATS structure is the key to everything working properly, so take a look at it as you work through the examples. Let's begin looking at each stage in the pipeline.

1. Extraction - ExtractData.m

The function ExtractData.m extracts EEG data in various forms from your segmented files (e.g., channel data, GFA, data from independent components).

Usage:

```
[STATS]=ExtractData(condnames,condfiles,levels,design,savestring,varargin)
```

Inputs:

condnames - a cell array of condition labels. For example,

```
{'AE' 'AH' 'SE' 'SH'}
```

These labels correspond to the condition names I've decided to give in the SD study

(awake easy, awake hard, etc). You can name your conditions anything you wish. For example, a study with only three conditions might look like this:

```
{'Face' 'House' 'Butterfly'}
```

condfiles - a cell array of filenames for each subject and condition. Leave empty (ie., [ ]) and MATLAB will bring up an interface for you to load the appropriate subject condition files. After this is done, the file is saved with savestring appended to it (described below). For subsequent calls, just type the filename that was saved instead of using [ ] and you won't have to manually load each file again.

levels - the J and K levels of factor A and B respectively. For example,

```
[2 2]
```

indicates a 2x2 design such as in the sleep deprivation study. If you have only one factor, just enter the number of levels for that factor. For example,

```
3
```

would indicate a one-factor design with three levels.

design - a string indicating the design type. For example,

```
'ww'
```

indicates that this is a two-factor experiment that is fully within subjects as in the SD study. If you had done a mixed design (i.e., between-within), you would enter

```
'bw'
```

Keep in mind that if you are using a between-within design, the between subjects factor is always considered factor A as is traditionally done in factorial designs. So,

```
'wb'
```

is not a usable option and will result in an error. If you had a one-factor design that was fully between subjects, you would simply enter,

```
'b'
```

savestring - a string that you make up that will be appended to important output files, (most importantly the STATS structure) as you work through the STATSLAB stream. This should have some identifying information in it that will help you remember what type of analysis you ran. For example,

```
'GFA_analysis'
```

Thus, the STATS structure filename will be STATS_GFA_analysis.mat. That file will be loaded automatically in all subsequent stages in the pipeline.

varargin - this holds the optional inputs that tell STATSLAB what kind of information you wish to extract from the .set files. For example,

```
'scalpgfa'
```

extracts the full scalp data array for later GFA calculations. This particular option does not require any following input arguments. However for scalp channel data,

```
'scalpchan' , {'FCz','Cz'}
```

would extract the specified scalp channels and so two inputs are needed as you can see (one for the measure of interest and one giving the channel labels). If asking for more than one channel, the channel group will be averaged together during the resampling stages (ResampleData.m) giving you a channel cluster. This function does not handle doing stats on multiple channels independently in the same call. Keep in mind that the labels you choose to ask for must correspond to the labels in your EEGLAB channel structure. Furthermore, every subject must share these labels/have these channels. If one subject's Cz was removed during preprocessing, an error will occur because this subject does not have a Cz to extract. If you only wish to extract one channel you could enter,

```
'scalpchan' , {'FCz'}
```

There are two options that involve extracting independent components (ICs). These options also require two inputs, one is the IC measure of interest and the other is a text file which tells STATSLAB which ICs to extract for each subject. You will have to construct such a file, but it is straightforward. Click on the wwICfile.txt in your demo folder. You will see that there is one column of numbers. Each number refers to an IC that I want to extract for each subject. There are 18 subjects, so there are 18 rows with numbers arranged into one column. For example,

```
'icagfa', 'wwICfile.txt'
```

would reduce the data for each subject down to the IC listed in the text file and extract the resulting data for later GFA calculations. For a within-subjects design, there is only ever one column of numbers. However for a fully between subjects design there would be one column of numbers for every condition in your study. Click on bbICfile.txt in the demo folder to see an example. Here you will see four columns, one for each condition. It is important that the order of the columns corresponds to your condition labels. For example, if you have a 2x2 between subjects design, with condnames, {‘old’ ‘young’ ‘healthy’ ‘sick’}, then ‘old’ is condition 1, ‘young’ is condition 2, and so on. The columns in your IC text file should correspond to this arrangement as well. If you have an unequal number of participants in your between subjects design, just delimit empty rows with commas. For example, in bbICfile.txt you will see that condition 1 only has 3 subjects. A comma delimited space in row 4 column 1 is needed so STATSLAB knows which subjects are in which condition. For between-within (mixed) designs, the same construction applies, except that you only need a column for each level of factor A (the

between subjects factor). For example, if you have a 2x2 between-within subjects design, with condnames, {‘old_face’ ‘old_house‘ ‘young_face’ ‘young_house}, then the IC text file needs only 2 columns of numbers indicating which IC to choose. This is because there are 2 levels for factor A (the between subjects factor of age in this case). This difference in construction is simply due to the fact that if I choose IC #5 to extract for a subject in the ‘old_face’ condition, I would also want to extract that same component for that subject in the ‘old_house’ condition, as ‘face’ and ‘house’ are within subjects conditions and share identical ICA decompositions. Of course it would not make sense to compare different components for the same person across two conditions (there would surely be a significant difference, but a meaningless one).

Another IC extraction option is,

```
'icamax' , 'wwICfile.txt'
```

This would reduce the data for each subject down to the IC listed in the text file. Once the data has been projected back to the scalp, STATSLAB finds whichever site has the maximum weight for that IC, according to the ICA weight matrix, and extracts its data. For example, if you chose a medial-frontal IC for a particular subject, that component would be projected back to the scalp, and some site at or around FCz would probably have the maximum weight and its data would be extracted.

Finally, here are some examples of ExtractData.m as it would look in the MATLAB command line.

```
[STATS]=ExtractData({'AE' 'AH' 'SE' 'SH'}, [], [2
2],'ww','Occipital_Analysis','scalpchan',{'A23', 'A24' })
```

This would extract data from the listed scalp sites, and save the STATS structure with the filename STATS_Occipital_Analysis.mat. If I wanted IC data I could enter,

```
[STATS]=ExtractData({'AE' 'AH' 'SE' 'SH'}, 'condfiles_icamax_analysis.mat',
[2 2],'ww','icamax_analysis','icamax', 'wwICfile.txt')
```

This would load the file condfiles_icamax_analysis.mat instead of displaying a browser for you to choose the files (if you had manually loaded the files once before). The data would be reduced to the ICs in wwICfile.txt and data at the site with the max weight for those components would be extracted. The STATS structure would be saved as STATS_icamax_analysis.mat. After successfully running ExtractData.m, you will have once file for each subject that holds the extracted data. These files are in the native MATLAB format (.mat). Original filenames are preserved but with the word “extracted” appended to them.

2 . resampling - ResampleData.m

The function ResampleData.m resamples with replacement from the EEG trials in order to build sampling distributions of ERPs (or GFA waveforms if you selected that measure when you ran ExtractData.m). The process of continuously resampling with replacement from the data at hand referred to as bootstrapping. In essence, we are pulling ourselves up by our own bootstraps. From these empirical sampling distributions we can calculate statistics (CIs, *p* values etc). Note that the 20% trimmed ERP (mean) is taken across trials at each timepoint to give us a robust measure of central tendency. When you run ResampleData.m you will be prompted to load the output files from ExtractData.m (the ones with “extracted” appended to the filename). Once a file has been resampled it is saved with the original filename with the word “bootstrapped” appended to it. This function can take some time to run depending on the number of bootstrap samples you choose to take as well as the size and number of files. However,

your computer will not be rendered useless while it runs as not much RAM is required, just time. Grab a coffee, or run ResampleData.m over night. Let me explain the inputs to ResampleData.m.

Usage:

```
[STATS]=ResampleData(STATS,nboot,trialcap)
```

STATS - this is the STATS structure that was saved when you ran ExtractData.m. Simply type the filename as it appears in your directory. It will begin with “STATS” and have savestring appended to it as explained above. For example,

```
'STATS_GFA_Analysis.mat'
```

nboot - The number of bootstrap samples you wish to take. Usually this is a large number (e.g., 599, 1000, or 50000). The larger the number, the more time it will take to run. It is still unclear in the literature how many bootstrap samples is enough to build a good estimate of the population distribution. Most EEG studies are using 1000. When resampling the mean, as we are doing, the larger the number of resamples, the closer the bootstrapped distribution will be to normal. So for 1000 resamples simply enter,

```
1000
```

trialcap - This caps the number of total trials that are used in the resampling procedure. By default STATSLAB resamples from the total number of trials you have for a given condition. For example, if condition 1 has 200 trials to choose from, each resample will consist of a new 200 trials (because we sample with replacement). There are times however when you may want to sample from a smaller of trials in order to equate noise across conditions. For example, if condition 1 has 100 trials and condition 2 has 25 trials, condition 2 will have more noise. This can offset the waveforms in measures of GFA and result in artifactual significant differences, even in the baseline. This is bad.

The trial cap in this case should be set to 50 to equate noise. For scalp data or ICs that are projected to the scalp, this is not really a problem as noise oscillates around zero. I recommend setting a trial cap when using GFA measures, especially when there are large differences in the number of trials across conditions. To do this simply enter the number you wish to the cap to be, for example,

```
50
```

Here are some examples of ResampleData.m as it would look in the MATLAB command line.

```
[STATS]=ResampleData('STATS_GFA_Analysis.mat', 1000)
```

Or with the trial cap option,

```
[STATS]=ResampleData('STATS_GFA_Analysis.mat', 1000, 50)
```

3. calculating statistics - GroupStatistics.m & SubjectStatistics.m

Calculate statistics for any number of levels, up to two factors, for between subjects, within subjects, or mixed designs. At this point you can choose to run statistics at the group and/or single-subject level using GroupStatistics.m or SubjectStatistics.m, respectively. Both functions have nearly the same input arguments. As usual, after running one or both of these functions, the results are stored in the STATS structure. Let's begin with how to use GroupStatistics.m.

Usage:

```
[STATS]=GroupStatistics(STATS,alpha,nsamp,varargin)
```

STATS - this is the STATS structure filename as described above.

alpha - arbitrary significance cutoff value. This will be corrected using the Rom (1990) method for correcting for multiple comparisons.

nsamp - the number of resamples from the group you wish to take (e.g., 599, 1000, 50000, etc...). In other words, we need to resample with replacement from the subjects themselves in order to build a sampling distribution with which to calculate statistics. This is an estimate of the sampling distribution we would have if we were to go out into the world and take nsamp random samples. If you do not believe that resampling from the group is important, STATSLAB also calculates the group statistics that you would get without resampling from the group, which are much less conservative, but perhaps not as generalizable. Resampling takes time. Grab a coffee or run overnight if you have a large sample.

varargin - linear contrasts for Factor A, Factor B, and Factor AB (interaction). If you have a single factor experiment, then only enter contrasts for Factor A. Linear contrast coefficients take some getting used to if you are not familiar with them. Basically they tell STATSLAB which conditions to compare using 0's, 1's, and -1's. For example, in the SD demo study, I have a 2x2 within-within (repeated measures on both factors) design with four conditions: (1) Awake, Easy Decision, (2) Awake, Hard Decision, (3) Sleepy, Easy Decision, and (4) Sleepy, Hard Decision. With their labels, condition 1 is 'AE', condition 2 is 'AH', condition 3 is 'SE', and condition 4 is 'SH'. Let me show you what linear contrasts for Factor A (level of Sleepiness) look like. For example, if we want to compare condition 1, 'AE', with condition 3, 'SE', the linear contrast would be,

```
[1 0 -1 0]'
```

This means to subtract condition 3 from condition 1 ('AE' minus 'SE') and ignore conditions 2 and 4. Let's try the other Factor A comparison, 'AH' minus 'SH',

```
[0 1 0 -1]'
```

It is very likely that we will want to compare alert to sleepy for both levels of Factor B (easy and hard). No problem, just combine the last two examples,

```
[1 0 -1 0; 0 1 0 -1]'
```

You will notice the single quote at the end of these contrasts options. Do not forget it, it converts a row into a column, which is what STATSLAB requires. Don't worry about why that is.

At this point there is nothing left to compare for Factor A unless we wanted to pool conditions together. You can do this like so,

```
[1 1 -1 -1]'
```

Here the two levels of Factor B at each level of Factor A have been pooled together. Thus we would be comparing the sum of conditions 1 and 2, minus the sum of conditions 3 and 4. I'm not really sure why you would want to use pooled contrasts, but perhaps there is a place for that in someone's hypothesis.

Factor B contrasts are constructed in a similar way. To compare 'AE' to 'AH' and also 'SE' to 'SH', enter

```
[1 -1 0 0; 0 0 1 -1]'
```

Finally, because this is a factorial design, we can look at the interaction. Here is is the contrasts matrix for the interaction,

```
[1 -1 -1 1]'
```

I know this seems strange, but this configuration denotes (AE-AH) - (SE-SH), in other words, the difference of the difference, which is by definition an interaction. Altogether, GroupStatistics.m in the command line would look like this if we specified all non-pooled linear contrasts for each factor and the interaction, and used alpha = .05.

```
[STATS]=GroupStatistics('STATS_GFA_Analysis.mat',.05, [1 0 -1 0; 0 1 0 -1]',
[1 -1 0 0; 0 0 1 -1]', [1 -1 -1 1]')
```

Note that the contrast matrices go in a specific order. That is, Factor A contrasts, followed by Factor B contrasts, and then followed by the interaction contrast. You must specify all three. If you are not interested in looking at, for example, any factor B comparisons at all, you must still type contrasts for that factor. Just don't bother interpreting the results later on, if that is what you want. In most cases, you will want to look at at least one comparison for each factor and interaction. Let's look at more examples. What if we only have one factor, as in a 1x3 design? All possible (non-pooled) linear contrasts would look like this,

```
[1 -1 0; 1 0 -1; 0 1 -1]'
```

Thus, for your 1X3 design, there are 3 contrasts: (1) condition 1 minus condition 2, (2) condition 1 minus condition 3, and (3) condition 2 minus condition 3. In the MATLAB command line it would appear like so,

```
[STATS]=GroupStatistics('STATS_GFA_Analysis.mat',.05, [1 -1 0; 1 0 -1; 0 1
-1]')
```

Okay, now something trickier. Let us say we have a 2X4 design. Note that there are 8 conditions. Conditions 1:4 belong to the first level of factor A, and conditions 5:8 belong to the second level of factor A. Here are the contrasts matrices for all possible comparisons for factor A,

```
[1 0 0 0 -1 0 0 0; 0 1 0 0 0 -1 0 0; 0 0 1 0 0 0 -1 0; 0 0 0 1 0 0 0 -1]'
```

Notice that the semicolon separates each comparison that is being made across levels of factor A. For example the first contrast is comparing condition 1 with condition 5, the

second contrast is comparing condition 2 with condition 6, and so on. Here is an example of some factor B comparisons (I won't do them all),

```
[1 -1 0 0 0 0 0 0; 0 0 1 -1 0 0 0 0; 0 0 0 0 1 -1 0 0; 0 0 0 0 0 0 1 -1]'
```

Here the first contrast compares condition 1 to 2, the second compares condition 3 to 4, and the very last contrast compares condition 7 to 8. Now for this design here are all possible interaction contrasts,

```
[1 -1 0 0 -1 1 0 0; 0 1 -1 0 0 -1 1 0; 0 0 1 -1 0 0 -1 1]'
```

Notice that is this design there are three possible interactions that we could test. Remember that the interaction is simply the difference of the difference. For example the first interaction contrast compares (condition 1 - condition 2) - (condition 5 - condition 6).

I am not sure why I left the easiest example for last, but for the sake of completeness, here is what a two group comparison would look like (as in a *t*-test design),

```
[1 -1]'
```

Lastly, if you want to see what all possible contrasts look like for your 1-way design, you can type this into the command line and enter the number of conditions as the input argument,

```
[conA]=con1way(number_of_conditions)
```

Or if you have a factorial design,

```
[conA conB conAB]=con2way(number_of_FactorA_levels, number_of_FactorB_levels)
```

This will generate all possibilities for you to look at. I suggest using con1way or con2way to better understand how linear contrasts work based on different types of designs. These functions by default will show you pooled comparisons, which you may or may not actually want to test statistically. So learn how things work using these

functions and these examples, and enter your own contrasts based on your research question. If you have a design with many levels and perhaps 2 factors, you will see that the number of possible comparisons is immense and this can increase type I error. To control for multiple comparisons STATSLAB uses the Rom (1990) method recommended by Wilcox (2012). As usual, the more contrasts you make for a given factor, the harder it will be to reject the null hypothesis. Keep your designs small, it will make life easier.

Conducting single-subject statistics is done in a very similar way to group statistics. Here we would use the function SubjectStatistics.m,

Usage:

```
[STATS]=SubjectStatistics(STATS,alpha,varargin)
```

All input arguments for SubjectStatistics.m are identical to those in GroupStatistics.m (although there is no nsamp argument in SingleSubject.m). There is one important exception: If you have a between-within (mixed) design, it is by definition only possible to calculate single-subject statistics for your within subjects conditions. For example, say we have a 2x2 between-within design comparing older and younger folks to happy and sad faces. Keep in mind that it is imperative in this type of design to consider factor A as the between subjects factor. Condition 1 is 'Young_Happy', condition 2 is 'Young_Sad', condition 3 is 'Old_Happy', and condition 4 is 'Old_Sad'. Since age is our between-subjects variable, we may only compare calculate single-subject statistics across levels of factor B because that is our within-subjects variable (face type). In essence we would conduct two 1-way tests: condition 1 minus condition 2, and then condition 3 minus condition 4. No interaction is possible when doing single-subject

statistics for a between-within design. So in the case of a between-within design and single-subject statistics, STATSLAB requires some additional information in varargin beyond what is expected in GroupStatistics.m. Here is an example,

```
[STATS]=SubjectStatistics('STATS_BWAnalysis.mat',0.05,[1 -1]',
{'Young','Old'},{'Happy','Sad'})
```

Here you can see that varargin now holds an appropriate one-way contrast (which is used across all levels of factor A) as well as some new labels. These new labels are basically broad titles for each level, starting with factor A, then followed by factor B. STATSLAB now knows to do two separate single-subject analyses, one comparing 'Happy' and 'Sad' faces at the 'Young' level of factor A, and another analysis comparing 'Happy' and 'Sad' faces at the 'Old' level of factor A. If you had a 2x3 between-within analysis, and you wished to perform single-subject statistics, it would look like this,

```
[STATS]=SubjectStatistics('STATS_BWAnalysis.mat',0.05,[1 -1 0; 1 0 -1; 0 1
-1]',{'Young','Old'},{'Happy','Sad', 'Fearful'})
```

Notice that I have an appropriate contrast matrix (three comparisons being made which will be applied to each level of factor A) and the labels correctly denote 2 levels for factor A and 3 levels for factor B. As stated, these changes to the contrasts matrix and the additional labels only apply when doing single-subject statistics for a between-within design. After running SubjectStatistics.m and GroupStatistics.m all of the results are, as usual, stored in the same STATS structure you have been using all along.

4 . plotting results - GroupFigure.m & SubjectFigure.m

If you have made it this far you must be looking forward to visualizing the results. STATSLAB makes this easy. With GroupFigure.m and SubjectFigure.m, STATSLAB will generate figures for any of the contrasts you wish as well as add the appropriate titles

and legends automatically. The figure plotting functions in STATSLAB are intended to get you exploring the results ASAP. They are not intended to be infinitely flexible in terms of aesthetics (although some options are available). Note that in all MATLAB figures you can adjust colors, axes, labels, titles, etc, manually in the figure editor window (Edit -> Figure or Axes properties). Otherwise, if you are comfortable using the command line, the STATS structure holds all of the numbers and other information for you to build the figures however you wish. We'll begin with GroupFigure.m.

Usage:

```
[STATS]=GroupFigure(STATS,infodisplay,varargin)
```

STATS - The filename of the STATS structure.

infodisplay - a numerical flag (1 or 0). If set to 1, STATSLAB will show you your condition labels and contrast matrices. It is sometimes nice to have this information handy when looking at the figures. Especially as a check to see that the correct titles and legends are being displayed on the figures. Set to zero if you trust my programming (not recommended).

varargin - The optional inputs for GroupFigure.m are straightforward. For example,

```
'all'
```

This will generate all comparisons according to the specified linear contrasts. I suspect that most times, or least the first time plotting, you will want to use this option. If however you wish to only plot a subset of comparisons, you may enter this instead,

```
'FactorA', [1 2], 'FactorB', 2, 'FactorAB', 1
```

Let me translate. These options first indicate a factor (A, B, or the interaction), and then are followed by numbers indicating which contrast(s) you which to plot. In the previous

example I have chosen to plot the 1st and 2nd contrasts for factor A. For factor B, I only chose to plot the 2nd contrast. For the interaction, I have chosen to plot the 1st contrast. If you do not wish to see any figures at all for a given factor, simply leave it out like this,

```
'FactorA', [1 2], 'FactorAB', 1
```

So in this case no interaction plot would be generated. Altogether, GroupFigure.m looks like this in the command line,

```
[STATS]=GroupFigure('STATS_GFA_Analysis.mat',1,'FactorA', [1 2], 'FactorAB',
1, 'FactorB', [1 2]);
```

Notice that the order in which you enter the factors does not matter. For the SD study, the above example is equivalent to using the 'all' option (as all 5 contrasts were specified). If you wish to see what the group results would look like if you had not resampled from the subjects (only the trials), simply use the function GroupFigure_sample.m, which has identical inputs to GroupFigure.m. For example,

```
[STATS]=GroupFigure_sample('STATS_GFA_Analysis.mat', 'all');
```

The group figures produced by the last example can be found in the folder Demo/ GroupFigure.

Potting single-subject statistics presents a number of display challenges because of limited screen space and differences in scale across participants. Fortunately, STATSLAB includes a number of options for visualizing single-subject statistics. Each type of visualization has pros and cons as will be described.

Usage:

```
[STATS]=SubjectFigure(STATS,infodisplay,varargin)
```

The first two input arguments in SubjectFigure.m are the same as in GroupFigure.m. However, there are additional optional arguments that can be specified for varargin.

Three different visualizations of single-subject statistics can be generated with SubjectFigure.m: (1) traditional difference waves with CIs, (2) difference waves represented as colorbars, and (3) difference waves represented as colorbars with margin of error statistics (MOE). MOE is simply one arm of the CI and therefore reflects the precision of your estimate (e.g., mean difference = 6mV, but can vary by +/- 10Mv). Describing the MOE is really another way of using CI information (Cumming, 2012). To plot a traditional waveform with a CI use these options,

```
'wave', 'diffcol', [.8 0 0], 'CIcol' [0 .9 0] 'yaxis', [-4 4]
```

This option is nice because it mirrors the group figure visually. However, with a large number of subjects the waves will get too squished due to a lack of vertical screen space. `'diffcol'` and it's value of `[.8 0 0]`, specify the color of the difference wave as an RGB vector in the range of 0-1. The first number corresponds to red, the second to green, and the third to blue (hence RGB). Likewise, `'CIcol'` and its value of `[0 .9 0]`, specify the color of the CI. `'yaxis'`, and its value of `[-4 4]`, specify the range of the y axis. If any of these options are omitted, defaults will be used. Sometimes I like to generate the figures with default Y axis limits just to get a sense of what scale range would be appropriate to apply to all subjects, then simply run the function again specifying the Y axis range. To save on vertical space we can convert the difference waves to colorbars using these options,

```
'diff', 'caxis', [-4 4]
```

The C axis specifies the color range, which for these types of plots is identical as the Y axis range in the previous example. If you do not specify a C axis range, every subject will be plotted on their own scale. However, unless you have a very homogenous group

in terms of amplitude range, this will make subjects very hard to compare to one another because they are on different scales. As with the 'wave' option, you may want to leave out the C axis option at first to see what kind of variability exists, and then choose a fitting C axis range and plot again. These colorbar plots are nice, however, they do not show the CIs, and it's a sad day in statistics land when we omit CIs (Cumming, 2012). The solution is to display MOE information to give a sense of the width of the CI and thus the precision and accuracy of our estimate. This is accomplished like so,

```
'CI_MOE', 'caxis', [-4 4], 'zaxis', [0 3]
```

The Z axis option is the MOE range to plot. Leave it out and subjects will be scaled to themselves as usual. Keep the first value of the Z axis as 0. Again, the C axis range specifies the color range, as in the 'diff' option above. Note that in all single-subject figures, black lines indicate when the CI does not include zero (significant differences). Here are some examples of what SubjectFigure.m looks like in the command line with various visualization options. The last example produces the figures found in the folder Demo/SubjectFigure.

```
[STATS]=SubjectFigure('STATS_GFA_Analysis.mat',1,'FactorB', [1 2], 'wave',
'diffcol', [0 0 0], 'CIcol', [0 0 .9], 'yaxis', [-4 4]);

[STATS]=SubjectFigure('STATS_GFA_Analysis.mat',1, 'FactorB', [1 2], 'diff');

[STATS]=SubjectFigure('STATS_GFA_Analysis.mat',1, 'FactorB', [1 2], 'diff',
'caxis', [-4 4]);
```

```
[STATS]=SubjectFigure('STATS_GFA_Analysis.mat',1, 'all', 'CI_MOE', 'caxis',
[-4 4], 'zaxis', [0 1.5]);
```

Measures of Association

In addition to categorical comparisons, STATSLAB can perform robust measures of association between subject waveforms and other continuous variables (e.g., RTs, accuracy, personality measures). STATSLAB tests for significant correlations with winsorized and bootstrapped Pearson's r, at all EEG time points (Wilcox, 2012). To visualize linear and nonlinear associations at any latency, STATSLAB has easy-to-use scatterplot functions. For linear relationships, CIs and prediction bands around the slope may be plotted. For nonlinear relationships, bootstrapped locally-weighted scatterplot smoothers (LOWESS), which are visually weighted by kernel density, may also be plotted. First, let's look at how to correlate ERPs from any condition or contrast, with an external continuous variable, at all EEG time points using WinBootCor.m.

robust correlations - WinBootCor.m

One can use bootstrapping to build an empirical sampling distribution of any statistic. WinBootCor resamples from the X and Y values (keeping appropriate values paired) in order to build the sampling distribution of Pearson's r. Using this sampling distribution we can calculate *p* values and CIs. In addition, outliers are controlled for by winsorizing the values in the both X and Y variables separately for every bootstrap sample (Wilcox, 2012).

WinBootCor requires a .mat file as your Y data (e.g., columns of behavioral measures to be correlated with EEG waveforms). The dimensions of this file are as follows: One column of data for every EEG correlate you wish to test, with as many

rows as you have subjects. For example, if you have 20 subjects in your dataset and you want to correlate RTs, accuracy scores, and scores on an anxiety scale, with some ERPs, your .mat file would be 20 rows X 3 columns. You may have such a file on hand from a spreadsheet or in a text file. Simply copy and paste the data (just the numbers) into a MATLAB variable and save it (right click and you will see familiar options).

Usage:

```
[STATS]=WinBootCor(STATS,infodisplay,nboot,tr,Ylabel,varargin)
```

STATS - The filename of the STATS structure.

Infodisplay - A numerical flag (0 or 1). Set to 1 if you would like to see your contrasts, condition names, Xlabels, and Ylabels.

nboot - Number of bootstrap resamples

tr - percentage to winsorize from the X and Y data. Keep as decimal place (e.g., .2 for 20%, .5 for 50% etc). Wilcox recommends .2.

Ylable - A string, or cell array of strings, indicating your Y variable(s) (correlates) that will each be correlated, one by one, with your EEG waveforms (X variable). For example, `{'RTs', 'accuracy', 'perfectionism_scores'}`

The order here must correspond to the columns in your Y data file described above. That is, RTs are in the first column, accuracy scores are in the 2nd column, etc.

varargin - String and numerical pair indicating the EEG data you would like to use in the correlation. For example,

```
'Condition', 3
```

would correlate data from condition 3 with each Y variable. Using condition waveforms in the correlation does not require any more commands and can be used for any

design. Note that later on when plotting scatterplots, the corresponding condition label will automatically be added to the X axis and the Y label will be added to the Y axis.

To use difference waves as the X variable there are always three varargin input arguments, and some subtle differences depending on the design. Furthermore, in order to use difference waves for correlations, you must have previously calculated single-subject statistics (which produces the difference waves). For ‘ww’ designs the first argument in varargin can be 'FactorA', or 'FactorB', or 'FactorAB'. Follow that with a number indicating the contrast (difference wave) you wish to use in the correlation. Then simply make up a label to identify/name your contrasts. Note that this label is automatically added to your scatterplots in subsequent steps. For example,

```
'FactorAB', 1, 'Interaction_comparison'
```

would use the first interaction waveform as the X variable in the correlation and name this comparison, 'Interaction_comparison’. For 'bw' designs, you must indicate ‘FactorA1’, FactorA2, etc, as the strings, as contrasts were taken separately for each level of factor A, as previously noted. For example,

```
'FactorA2', 3, 'my_main_comparison'
```

would use the 3rd difference wave (contrast) that was calculated at the 2nd level of FactorA. And this comparison would be named, ‘my_main_comparison’. For a ‘w’ (one-way repeated measures) design, you could type this,

```
'FactorA', 2, 'my_main_comparison'
```

There is only one factor in a ‘w’ design, and so we always call this FactorA. There is no other possible option. The 2nd contrast would be taken and used in the correlation, and this test would be given the name ‘my_main_comparison’. For a ‘b’ (one-way between subjects measures you can not use difference waves in the correlation, only condition

waveforms. This is because single-subject statistics cannot be calculated in this type of design. Here are some examples of WinCorBoot.m in the MATLAB command line.

```
[STATS]=WinBootCor('STATS_someanalysis.mat',1,1000,.2,'RT','Condition',3)
```

In this example, the Y data file (.mat) should only have one column of data for the RTs. Thus, RTs will be correlated with condition 3's waveform. In the SD study, condition three has the label 'SE'.

```
[STATS]=WinBootCor('STATS_someanalysis.mat',1,1000,.2,{'RTs',
'accuracy'},'FactorAB',1, 'interaction')
```

Here there are 2 Y variables entered, so the Y data file must have 2 columns. In this case the first interaction contrast is being correlated with the both Y variables, and this test is called "interaction".

```
[STATS]=WinBootCor('STATS_someanalysis.mat',1,1000,.2,'RTs','FactorA2',
2,'Easy_vs_hard_SLEEPY')
```

As you can see, this set of commands is for a 'bw' design as the 'FactorA2' argument points out. Altogether, we would be correlating RTs with the 2nd contrast at the 2nd level of factor A.

To plot winsorized and bootsrapped Pearson's r, CIs, and *p* values, simply modify the following command to include the condition labels/contrast labels and the Y labels you used when you ran the correlation. For example, if condition 1 has the label 'AE', change "Condition_label" to AE. If your Y label was 'RTs'. Change "Y_label" to RTs.

```
figure; plot(STATS.xtimes,STATS.corr_results.Condition_label.Y_label.rw)
```

This will give you r values, hence rw (winsorized Pearson's r).

```
figure; plot(STATS.xtimes,STATS.corr_results.Condition_label.Y_label.p)
```

This will give you p values.

```
figure; plot(STATS.xtimes,STATS.corr_results.Condition_label.Y_label.CI(1,:))
hold on
plot(STATS.xtimes,STATS.corr_results. Condition_label.Y_label.CI(2,:))
```

These will give you the CI around the r value. If contrasts were used in the correlation, instead of entering you condition label as is shown above, simply replace that with the label you gave to your contrast. For example, if I labelled my contrast “interaction”, I would enter the following commands to plot r values, *p* values, and CIs,

```
figure; plot(STATS.xtimes,STATS.corr_results.interaction.RTs.rw)
figure; plot(STATS.xtimes,STATS.corr_results.interaction.RTs.p)
figure; plot(STATS.xtimes,STATS.corr_results.interaction.RTs.CI(1,:))
hold on
plot(STATS.xtimes,STATS.corr_results.interaction.RTs.CI(2,:))
```

visualizing linear and nonlinear relationships - SlopeCI.m & LowessCI.m

Pearson’s r is impossible to interpret without seeing a scatterplot of the data. SlopeCI.m produces a scatterplot at any latency you wish and overlays a parametric slope with its CI and prediction band. LowessCI.m uses LOWESS (Cleveland, 1979) to trace a path through the data which does not need to be linear or constrained by any mathematical function. Basically, a window slides across the scatterplot and the data contained in each window is fit with a line. The resulting trace can highlight complex relationships in the data that a typical mathematical function (linear, curvilinear etc) may miss. LowessCI.m uses bootstrapping to produce many LOWESS curves. The resulting sampling distribution of LOWESS curves is used to calculate the kernel density (i.e., transforming a discrete distribution into a continuous one) within vertical bins that divide the width of the scatterplot. Each bin is colored based on its kernel density, giving a nice

visualization of the likelihood of a LOWESS curve passing through any given point in the scatterplot. The folder Demo/Scatterplots has some example figures. Let's look at how to use SlopeCI.m and LowessCI.m.

Usage:

```
[STATS]=SlopeCI(STATS,infodisplay,Xlabel,Ylabel,msplot,CI_color,colorlimit)
```

STATS - The filename of the STATS structure.

Infodisplay - A numerical flag (0 or 1). Set to 1 if you would like to see your contrasts, condition names, Xlabels, and Ylabels.

Xlabel - String that indicates the X variable (EEG data). Must be the same Xlabel used when running WinBootCor.m. For example if condition 1 for the SD study was used in a correlation when you ran WinBootCor.m you could enter,

```
'AE'
```

Or if a contrast was used, use the label that you gave it when you ran WinBootCor.m. For example,

```
'interaction'
```

Ylabel - A string indicating your Y variable(s) (correlates) Must be the same Ylabel used when running WinBootCor.m. For example,

```
'RTs'
```

msplot - a number indicating the millisecond latency at which to make the scatterplot

CI_color - the color of the CI and prediction band. This is in the form of an RGB triplet. For a gray CI enter,

```
[.5 .5. 5]
```

colorlimit - the color that the CI will fade to as it increases in width. I like when the CI fades into the background of the figure, when the relationship is most uncertain (i.e., when the CI is widest). So, for most cases you can make it fade to white by entering,

```
[1 1 1]
```

SlopeCI.m would look like this in the command line,

```
[STATS]=SlopeCI('STATS_Analysis.mat',1,'AE','RTs',170,[.4 .4 .4],[1 1 1])
```

This command would produce a scatterplot at 170ms, with the amplitudes from the “AE” condition on the X axis, and the RTs on the Y axis. The CI would be grayish, and would fade to white as the CI became wider.

The inputs to LowessCI.m are similar to SlopeCI.m, with a few differences that have to do with the bootstrapping and kernel density estimates.

Usage:

```
[STATS]=LowessCI(STATS,infodisplay,Xlabel,Ylabel,msplot,nboot,span,nbins)
```

STATS - The filename of the STATS structure.

Infodisplay - A numerical flag (0 or 1). Set to 1 if you would like to see your contrasts, condition names, Xlabels, and Ylabels.

Xlabel - String that indicates the X variable (EEG data). Must be the same Xlabel used when running WinBootCor.m. For example if condition 1 for the SD study was used in a correlation when you ran WinBootCor.m you could enter,

```
'AE'
```

Or if a contrast was used, use the label that you gave it when you ran WinBootCor.m. For example,

```
'interaction'
```

Ylabel - A string indicating your Y variable(s) (correlates) Must be the same Ylabel used when running WinBootCor.m. For example,

```
'RTs'
```

msplot - a number indicating the millisecond latency at which to make the scatterplot

nboot - number of bootstrap LOWESS samples

span - number in range 0 to 1, indicates size of sliding window (percent of data to use)

nbins - number of vertical bins in which to calculate kernel density.

Altogether, LowessCI would look like this in the command line,

```
[STATS]=LowessCI('STATS_Analysis.mat',1,'AE','RTs',170,1000,.4,1000)
```

Keep in mind that when using LowessCI.m the larger the number of bootstrap samples (nboot) and number of bins (nbins) the longer it will take to produce the figure. Play around with the different input arguments to see how they affect the visualization.

References

Algina J, Oshima TC, Lin W-Y, (1994). Type I error rates for Welch's test and James's second- order test under nonnormality and inequality of variance when there are two groups. Journal of Educational and Behavioral Statistics .19:275–291.

Cleveland, W. S. (1979). Robust Locally Weighted Regression and Smoothing Scatterplots. Journal of the American Statistical Association, 74(368), 829–836. doi:10.2307/2286407

Cressie NAC, Whitford HJ, (1986). How to use the two sample t-test. Biometrical Journal.28:131–148.

Delorme, A., & Makeig, S. (2004). EEGLAB: an open source toolbox for analysis of single-trial EEG dynamics including independent component analysis. Journal of Neuroscience Methods, 134(1), 9–21. doi:10.1016/j.jneumeth.2003.10.009

Desjardins, J. A., & Segalowitz, S. J. (2013). Deconstructing the early visual electrocortical responses to face and house stimuli, 13, 1–18. doi: 10.1167/13.5.22.doi

Doksum KA, Wong C-W, (1983). Statistical tests based on transformed data. Journal of the American Statistical Association .78:411–417.

Hampel FR, Ronchetti EM, Rousseeuw PJ, Stahel WA, (1986). Robust statistics. New York, NY: Wiley.

Hsiang, S. M. (2013). Visually-Weighted Regression, 1–10.

Huber PJ, (1964). Robust estimation of location parameters. Annals of Mathematical Statistics. 35:73–101.

Oruç, I., Krigolson, O., Dalrymple, K., Nagamatsu, L. S., Handy, T. C., & Barton, J. J. S. (2011). Bootstrap analysis of the single subject with event related potentials. Cognitive Neuropsychology, 28(5), 322–337.doi10.1080/02643294.2011.648176

Pernet, C. R., Chauveau, N., Gaspar, C., & Rousselet, G. a. (2011). LIMO EEG: a toolbox for hierarchical LInear MOdeling of ElectroEncephaloGraphic data. Computational Intelligence and Neuroscience, 2011, 831409. doi10.1155/2011/831409

Rasmussen JL, (1989). Data transformation, type I error rate and power. British Journal of Mathematical and Statistical Psychology .42:203–211.

Rom DM, (1990). A sequentially rejective test procedure based on a modified Bonferroni inequality. Biometrika .77:663–666.

Rousselet, G. A., Gaspar, C. M., Kacper, P., Pernet, C. R., & Tangermann, M. (2011). Modeling single-trial ERP reveals modulation of bottom-up face visual processing by top- down task constraints (in some subjects). Frontiers in Psychology, 2, 1–19. doi:10.3389/fpsyg.2011.00137

Tukey JW, (1960). A survey of sampling from contaminated normal distributions. In: Olkin I, Ghurye S, Hoeffding W, Madow W, Mann H, eds. Contributions to probability and statistics. Stanford, CA: Stanford University Press: 448–503.

Thompson GL, Amman LP, (1990). Efficiencies of interblock rank statistics for repeated measures designs. Journal of the American Association .85:519–528.

Wilcox R. R, (1996a). Statistics for the social sciences. San Diego,CA: AcademicPress.

Wilcox, R. R. (2005). Introduction to Robust Estimation and Hypothesis Testing. Academic Press. Retrieved from https://play.google.com/store/books/details