4.7.3 Testing your experiment

Authors: Yoni Amir, Inbal Gur-Arie & Adi Sarig, August 2020

After the experiment is programmed and before running it on a large amount of subjects, we should test if the experiment is doing what we think it does.

Below is a general manual of tests we should perform on every experiment before running it. Each test has an explanation and some tests have references to matlab generic codes that could be used as templates for your version, adapted to test your own experiment.

The tests you should run by yourself before running the experiment on subjects (including pilot):

1.    Go over the methods file and extract the key features of the experiment (to be tested below): (i) how many conditions should you have, and what are the levels; (ii) how many trials should be included in each condition; (iii) what are the events in each trial; (iv) what is their duration; (v) what is the expected size of the stimuli; (vi) what responses are being collected; (vii) is there a time-window for response collection; (vii) are the trials divided to blocks, and are there breaks?; (iix) any other factor that is essential for your experiment. Be as thorough and as detailed as possible.

Even prior to the formal tests described below, **try the experiment**; does it look about right? This is the first step. There is no point in running a battery of tests before the experiment is final and seems ok to you.

 2.    Test the instructions are presented where they should:

When running the experiment make sure every part has its own instructions in place, and that none is missing (that is, all parts have instructions).

3.    Make sure the instructions are properly put:

Proofread your instructions- after (or during) you run your full experiment, make sure there aren’t any typos, misspell, etc. both in the code itself and when you run it. Also, note that we use a gender-neutral language in our experiments, adopting the plural form.

4.   Make sure the participant’s answers are correctly recorded:

This test is done against the output of your experiment, whether it is saved on matlab, excel, or any other program:

a.    Document the responses you give and compare them to the experiment’s output. Best would be to follow a predefined protocol: for instance, if subjects respond using the keyboard arrows, you should press 10 times to the right, 10 times to the left etc. You should know the exact order of responses you gave for you to be able to check it afterwards in the output. Next, after completing the experiment, you should open the output and make sure the order of reactions you took is as the output says.

b.    If your task has correct/incorrect answers and this is coded (in that case, make sure your code saves BOTH the accuracy and the actual response, so you could recalculate the accuracy and make sure the coding is accurate), run the same procedure described above, but now manipulating accuracy. That is, you should answer correctly for 10 trials, then give wrong answers for 10 steps and so on (again, make sure you remember the pattern of answers you gave). Afterwards, check the documentation in the output- that it saves your answers correctly.

c.     Assuming you are interested in RTs, it wouldn’t hurt to also play with that a bit: For instance, in the 1st five trials, answer slowly, and in trials 6-10, fast. Check the responses in the logfile, just as a crude test that responses are logged correctly with respect to time.

 5.     Make sure the output contains everything it should

This part describes what should generally be saved in your output, but you should also go back to the method file to see if there are any additional types of information that need to be saved:

a.    Subject number

b.    Part of the experiment (e.g. calibration, main task, objective post-test), when relevant.

c.     Block number

d.    Trial Number

e.    Trial type (there should be a separate column for each condition that defines the trial)

f.      All events that happened in the trial (fixation, mask, target, prime etc.), including their time stamps. You can also code their duration (which is derived from their time stamps), but make sure you also save the actual onsets.

g.    All responses.

h.    Accuracy (correct/incorrect) of responses, if relevant (optional, this could also be calculated later on based on the responses).

i.      RTs for every response.

6.    Make sure you know what each column in the output stands for

Document in a different file what each column means for the sake of future generation (indicative column names are highly recommended).

7.    Run the full experiment on yourself and analyze the results; based on the tests described in the next section. Also, you can make sure the results generally make sense (though note that due to individual differences, this won’t always happen):

a.    If one condition is supposed to be harder than the other, the RTs should show this pattern (longer RTs for the harder condition).

b.    If one condition is supposed to be harder than the other, the accuracy should show this pattern (less accurate in the harder condition).

 If this is an EEG/eye tracking experiment, you should also test the triggers and make sure they correspond to your logfile, so that the exact number of triggers are sent, their content is accurate, and their timing is accurate (test #4 in the next section). Also, you must save a file in which you explain the meaning of the triggers and how they code the different events and conditions.

The tests you should run on your data to validate all conditions and timings are as expected:

This section describes tests you run with designated codes on your data. Notice the codes are written as general template, so you will need to adjust them to your experiment (they are located at “G:\My Drive\MudrikLab020818\Common Resources\Running an experiment\Code\Testing your experiment”)

1.    Make sure the duration of each stimulus is accurate:

A.    As saved in the logfile:

The script “**check\_experiment\_timings**” asks you to choose one participant data (matlab file), specify the names of the timing columns in your output file (e.g., fixation time stamp vector, target time stamp vector etc…) and define “stimuli expected durations” (for each stimulus). Then, the script  runs the following funcitons: **check\_fixation\_duration ,check\_prime\_duration, check\_target\_duration, check\_mask\_duration.** These get the time stamps vectors of the different stimuli presented in a typical masking experiment (see the function documentation in the code itself for further explanation), and the expected duration of each stimulus. The functions validate each stimulus was indeed presented at the expected duration, and that the standard deviation is not above 2 msec (0.002 seconds).

Below is an example of the script’s input in the command window:

Input the name of your data table (as variable not string!): Data

Input the name of the fixation onset time-stamp vector(as string): 'fixation\_onset'

Input the name of the prime onset time stamp vector(as string): 'prime\_onset'

Input the name of the mask onset time stamp vector(as string): 'mask\_onset'

Input the name of the target onset time stamp vector(as string): 'target\_onset'

Input the name of the target offset time stamp vector(as string): 'mask\_onset'

Input the expected fixation duration in msec (int): 500

Input the expected prime duration in msec (int): 50

Input the expected mask duration in msec (int): 50

Input the expected target duration in msec (int): 50

After running, the script will print the results in the command window. Here is an example of one subject results from the demo subjects:

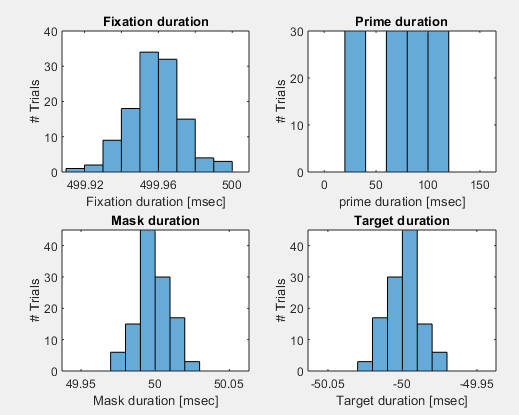
The mean fixation duration is 499.9584 msec and it is good, the std is 0.01687 msec and it is good

The mean Prime duration is 74.9884 msec and the duration of prime is not as expected, the std is 30.1679 msec and the std of prime time is too high

The mean mask duration is 50.0002 msec and it is good, the std is 0.021028 msec and it is good

The mean target duration is -50.0002 msec and the duration of target is not as expected, the std is 0.021028 msec and it is good

The script also generates plots of the data, so you can visually inspect the mean and std of the different stimuli durations:



B.   Run a photodiode test:

The log files report the timings as saved by the running platform (e.g., matlab), but there could still be discrepancies between the log files and what was actually presented on screen. Thus, the ultimate test of your timings is done with a photodiode that responds to flashes of light on the screen. In our lab, this is tested using an arduino device. Uri should add test

2.    Make sure the number of trials is as expected.

The function **verify\_trial\_num\_all\_subs** expects the following inputs:(i) the expected number of trials; (ii) the number of trials of each condition; (c) the number of characters to save from the file's name as the subject's name (for example if the file name you chose is “subject\_12\_my\_cool\_experiment.mat” and you just want the name ‘subject\_12’ the number of characters is 10). The function counts the number of trials each participant had, and validates it is the right number. It further counts the amount of trials each participant had in the different conditions, and validates it is the right number.

If you call only this function (not using the “**check\_trials\_num\_and\_randomization”** script, see below) the input is as follows:

verify\_trial\_num\_all\_subs(120,15,9)

After running the function, matlab will ask you to choose the directory where subjects’ data is saved (for example “subjects for demo”)

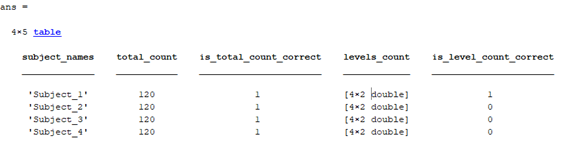
Then, you will have to provide some inputs in the command window. Below is an example of the function input in the command window:

Input condition Name as string, if no more conditions enter 0: 'ISI'

Input condition Name as string, if no more conditions enter 0: 'Congruent'

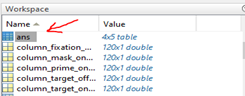
Input condition Name as string, if no more conditions enter 0: 0

The output of this function will look like this:

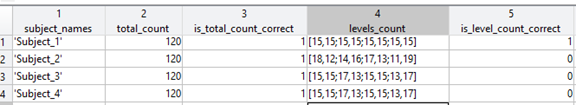


In the columns “is\_total\_count\_correct” and “is\_level\_count\_correct”, “1” denotes that the subject has the expected number of trials and “0” denotes that he/she hasn’t. And so, in the table above, though subjects have the correct number of trials across the entire experiment, this is not the case for the specific conditions for subjects 2-4.

The column “total\_count” returns the number of trials over the entire experiment. In order to see the number of trials counted in each condition click on the “ans” variable:



And check the “level\_count” column:



3.  Check randomization of trial order:  this function makes sure each subject has a different order of trials. The function again accepts the number of characters to save from the file's name as the subject's name. The function compares the division of conditions to trials between subjects to make sure each subject has a different order. Note that the code only inspects condition assignment; if your experiment also includes multiple stimuli (e.g., if you are using our ObScene stimulus bank), you should add a section that tests that the items are also differently ordered between subjects).

 If you call only this function (not from the “**check\_trials\_num\_and\_randomization”** script, see below) the input is as follows:

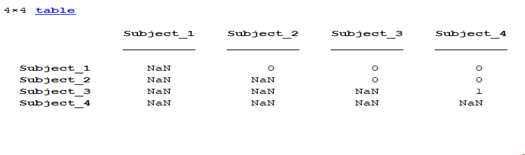
check\_randomization(10)

After running the function, matlab will ask you to choose the directory with your subjects’ data (for example “subjects for demo”)

And then you will have to provide some inputs in the command window. Below is an example of the function input in the command window:

Input randomized condition Name as string: 'ISI'

 The output of this function is:



The value “1” represents the same order of trials for the two subjects represented by the row and the column.

In order to make things easier ,we’ve created a script that calls the functions mentioned above in sections 2 and 3, named “**check\_trials\_num\_and\_randomization**”. First, the script will ask you to choose the directory where subjects’ data is saved (for example “subjects for demo”).

Then, in the command window, you will be asked to provide the following information: the expected number of trials in the entire experiment, the expected number of trials in each condition, the names of the columns in the output file where the different conditions are coded, the name of the column of the randomized condition. Here is an example of a demo run:

Input the total amount of trials in your experiment (int): 120

Input the amount of trials expected in each condition (int): 15

how many characters to save from the name of the file as the subject name (int): 9

Input randomized condition name as string: 'ISI'

Input condition Name as string, if no more conditions enter 0: 'ISI'

Input condition Name as string, if no more conditions enter 0: 'Congruent'

Input condition Name as string, if no more conditions enter 0: 0

 The outputs are the same as if you run each function separately.

4. If EEG or eye tracking- check the triggers are the same in matlab as in EEG / eye tracking.

**Check list of all tests:**

* The instructions are presented in the right place
* The instructions are properly put with no typos
* The participant’s answers are correctly recorded
* The output contains all predefined columns (see a list above)
* The name of the columns is clear and indicative
* A separate file explains the meaning of each column
* Responses are properly recorded
* Reaction times are properly recorded
* Accuracy is properly recorded
* If EEG/Eye tracking: a separate file explains the meaning of each trigger
* Presentation times and duration are accurate (prime / masking/ stimuli, SOA are as expected): log files test
* Presentation times and duration are accurate (prime / masking/ stimuli, SOA are as expected): photodiode test
* The number of trials is as expected.
* Each condition contains the right amount of trials.
* Trial order differs between subjects
* If EEG/Eye tracking: check the triggers are accurate

4.7.4 Raw data

Remember you should always keep the raw data (the data you get after a subject completed the experiment, without any further processing) safe! There should be a designated folder named “raw data” in every experimental folder, where all the raw data is saved.

Make sure to back up the raw data after each running day (or even better- after every subject). Usually, we run the experiment locally; therefore, the data is not backed up automatically to the drive and we must remember to copy the data to the drive after every run.

**Chapter 5: Common Resources**

5.3 How to check that your experiment shows stimuli with good temporal accuracy

Author: Uri Korisky ([Uri.korisky@gmail.com](mailto:Uri.korisky@gmail.com))

When you present an experiment (especially using PsychoToolbox), correct timing of stimuli presentation is always an issue. The main bottleneck is the refresh rate of the screen, but other factors might play a role as well.

5.3.1 Programming rules

• In your PTB MATLAB code, include the following lines when initializing PsychToolbox. This can be anywhere before the presentation of the first stimulus:

Screen('Preference', 'SkipSyncTests', 0);

This line ensures that PTB carries out its synchronization tests. Without it, PTB reports about stimulus presentation timing might (and will probably) not be valid.

This function takes some time to run, ~60 seconds max, so if you’re debugging for any purpose other than checking stimuli presentation timings, you can pass “1” as the third parameter.

Screen('Preference', 'VisualDebugLevel', 4);

This line sets the graphics debugging of PTB to the highest level. It may throw way more errors and warnings than usual, and give you an idea of how problematic your code is. Again, if you don’t debug for timing at the moment, you can leave this at “1” or any other level that suits you.

• As a rule of thumb, load all your images and create all the needed textures **before** entering a loop for presenting the stimuli. You don’t need loading and processing times messing with your display accuracy.

5.3.2 Setting MATLAB to high priority

If you’re working on a desktop that is not completely dedicated to your experiment, you can never know what runs in the background, or which application might decide suddenly that it wants to update\synchronize\backup while your experiment is running. To avoid other programs from slowing down your experiment, start MATLAB with the (almost) highest priority so that windows will know not to disturb it:

1. Close all open instances of MATLAB.

2. From the start menu, type “cmd” in the search box. A DOS window will appear.

3. Type the following line and press ENTER:

start /HIGH <path\_to\_matlab>matlab.exe

where <path\_to\_matlab> is replaced with the path on your computer where the file “matlab.exe” for the version of MATLAB you want to run resides. For example in my computer the line for running MATLAB version 2013b would look like this:

start /HIGH C:/Progra~1/MATLAB/R2013b/bin/matlab.exe

Notice that in DOS environment all folder names must be 8 characters or shorter, so you might have to change “Program Files” to “Progra~1” etc.

Disconnect DropBox, to the very least! You can disconnect the computer from the net, physically, as well.

5.3.3 Checking real presentation times with an oscilloscope connected to a light-sensitive diode

To get an objective measure of the timing in our experiment, we will use a diode placed on the screen, and measure its response along time. The diode senses luminance levels and transfers them into an electric current, which is then recorded by the oscilloscope. The data from the oscilloscope can later be downloaded to a flash drive (USB disk-on-key) and opened in MATLAB or excel and further analyzed.

IMPORTANT!

As the diode only records luminance levels, it doesn’t care what your stimulus looks like. It is best for the debugging that you replace your actual stimuli with high-contrast, black-and-white stimuli, preferably simple black or white rectangles. In my opinion, you should leave the experiment in its original form in terms of loading the images and creating textures from them, and only replace what is displayed with a rectangle.

5.4 Having MATLAB accept Hebrew characters

Author: Uri Korisky ([Uri.korisky@gmail.com](mailto:Uri.korisky@gmail.com))

If you try to write Hebrew characters in your MATLAB code, you’ll find out that re-opening it turned all your characters to question marks. This is because of both Windows and MATLAB.

First, [change Windows’ configuration](https://support.microsoft.com/en-us/windows?ui=en-US&rs=en-001&ad=US#1TC=windows-7) as to languages used by programs other than Windows:

1. Open Region and Language by clicking the Start button , clicking Control Panel, clicking Clock, Language, and Region, and then clicking Region and Language.
2. Click the Administrative tab, and then, under Language for non-Unicode programs, click Change system locale.  If you're prompted for an administrator password or confirmation, type the password or provide confirmation.
3. Select the language, and then click OK.

To restart your computer, click Restart now.

If the problem persists, change MATLAB’s encoding preferences. This example shows an encoding specific to Hebrew, but you may also try 'windows-1255' if that doesn't work.

slCharacterEncoding('ISO\_8859-8')

After these steps, there shouldn’t be any problem importing\exporting Hebrew to Excel files. If you want to write into text files, use fprintf() rather than fwrite().

Another thing that can't hurt: under preferences->Fonts, choose both code and text fonts to be a simple font which contains Hebrew. In preferences->Fonts->Custom, make sure your command window, command history and Editor all use the same font.

How to know if your problem stems from MATLAB:

* When you write Hebrew text in a file, close it and reopen, you get question marks instead of text.
* Variables containing Hebrew text appear in the workspace window as question mark icons
* PTB crashes when you try to display a Hebrew text string using "DrawFormattedText".

PsychToolBox and Hebrew:

Include the following lines in your code:

Screen('Preference', 'TextEncodingLocale', 'Hebrew\_israel.1255');

Screen('Preference', 'TextRenderer', 1);

Yoav Roll: On some computers TextRenderer 1 doesn’t work, so 0 should be used:

Screen('Preference', 'TextRenderer', 0);

These lines set the encoding of characters by PTB to be Hebrew, and the text renderer to be 1.

How to know if your problem stems from PTB:

* The text is displayed, but the encoding looks wrong: either the whole text is Latin vowels,  x's, or arrow icons.

5.5 Illustrator

Author: Itay Yaron ([mufc.itay@gmail.com](mailto:mufc.itay@gmail.com)), October 2020

In the lab, Adobe Illustrator is most commonly used to facilitate editing of high quality figures and images. The lab purchased a license that enables two concurrent sessions of Illustrator, so the first thing needed in order to work with Illustrator is to obtain the license information (username and password) from the lab manager.

Then, do the following steps:

1. Enter<https://creativecloud.adobe.com/apps/all/desktop?action=install&source=apps&productId=illustrator>.
2. Sign in with the username and password of the license.
3. Download illustrator to your local machine (if you are not redirected to the download page, click on the link in step 1).
4. Launch the installer and sign in with the username and password of the license.
5. Open the installed program :

a.      If two users from the lab are currently using the license, you would have to ask one of them to sign out (you can use the Lab's WhatsApp group for that purpose), or alternatively force them to sign out (for example in cases where one of these users launched the program a couple of weeks ago).

1. When you finished working with the program make sure to sign out, to enable other users to sign in easily.

**References**

Candice Morey Lab Handbook <https://ccmorey.github.io/labHandbook/>

[Wilson, G. *et al. PLoS Comput. Biol.* 13, e1005510 (2017).](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1005510)