Matlab Programming for Experimental Psychology

Assessment 1

1048577

14 January 2021

2,756 words

Matlab Assessment

The compatibility of the following functions has been tested on both a Retina-display Mac running OSX Catalina and Big Sur and a Windows 10 PC with various screen resolutions. These functions require Matlab R2020a or newer for optimal functioning, as well as the Curve Fitting Toolbox and the Image Processing Toolbox. Psychtoolbox-based functions are not compatible with external monitors connected to Retina-display computers or on Windows 8 operating system or newer. Psychtoolbox does not recognise the resolution of the external monitor as an extension of a Retina-display and formats the stimulus as a default one, while Windows 10 window desktop manager prevents Psychtoolbox from identifying the window space and may display the stimulus between two screens.[1] Psychtoolbox will have an error the first time that it is initialised on a Retina display. Please run a Psychtoolbox-inclusive function at least once prior to presenting the experiment to a participant.

Please use matlabProjectMaster.mlx to run the scripts provided below. The live script can run functions question by question. Please note that not all dependent functions are supplied, and that they need to be added to the working directory for them to function correctly. Dependent functions are noted in each question's instructions below.

Attached Files:

AltDrawDisc.m
alterGamma.m
AverageGammaPerceptionBarPlot.png
CloseandTidy.m
drawInstructions.m
eightTest.m
fitMeasuredGamma.m
gammaAnalysis.m
GammaDataPlot.png

[1] https://github.com/Psychtoolbox-3/Psychtoolbox-3/issues/276?fbclid=IwAR17EuVLnLZIlD1zVvVxfF5SXVwwyWpckmzpHdZGe15Zr6p4ZY1KfKLDSFE

GetKeyPress.m
matlabProjectMaster.mlx
OpenPTBWindow.m
Pattern1.png - Pattern8.png
Participant .csv data (e.g. “PerceptualGamma_123.csv”)
PerceptualGammaData.csv
plotGamma.m
plotLightOutput.m
PowerLawFit.m
Question2.png
RunGreyMatching.m
RunTrial.m
SetupInitialStimulus.m
UpdateStimulus.m

Question 1

Generate an array of values for 0 to 1 in steps of 0.05 and store the values in a variable called requestedSignal. Calculate the output intensities of a display with a gamma value of 2.2, using the equation $I = R^{\gamma}$ (where I is output intensity, R is requested signal and γ = 2.2) and store these values in a variable called lightOutput.

The following script generates an array of values, stores them, and calculates the intensities of a display with a gamma value of 2.2 for each value:

```
requestedSignal = [0:0.05:1]; %Create requestedSignal object with numbers from 0 to 1
in a 0.05 interval

lightOutput = requestedSignal .^ 2.2; %Run power law formula for every value in
requestedSignal
```

The `lightOutput` object contains each `requestedSignal` value to the power of 2.2. The object shows the output intensity when the gamma is 2.2.

Question 2

Plot a graph of light output against requested signal. The graph should reproduce the solid red curve in Figure 1. Think about (i) how to make sure the curve is smooth, and (ii) to *plot the curve without symbols. Save the code that generates the plot, and save the plot as a PNG file to submit with your assignment.*

The "plotLightOutput.m" function plots the gamma curve and outputs the plot as a .png file called "GammaDataPlot.png" in the working directory. The function accepts light output and requested signal data as input arguments, but only as matrices with either a single row or column to avoid potential errors with any other data types. The requested signal is placed on the x-axis while the light output is placed on the y-axis. The `lightOutput` data is then smoothed using the `smooth` function from the Curve Fitting Toolbox before being plotted, while `requestedSignal` should already have consistent intervals regardless of transformation (i.e. 1/9, 2/9…), have no noise, and be non-alterable due to their pre-specified nature. x and y axes limits are implemented since all values should remain between 0 and 1. In order to use this function for later questions, I have renamed the plot from this question to "Question2.png".

Question 3

Write a function to simulate the effect of displaying an image with different gammas. The function should accept two input arguments, the filename of the original image and the gamma value used for display. The original image is available on Canvas with the filename 'Bodleian_WTCollection.png'. You can use this as your test image and check that you are able to reproduce the images above for γ = 0.5, 1.0 or 2.0.

The "alterGamma.m" function accepts an image file and a desired gamma value as input arguments to simulate an environment with the given gamma value. The image file must be loaded by specifying the file location, or name if the image is in the current working directory (e.g. `alterGamma('testing.png', 1.0)`). The `imread` function transforms the image to a data array, and if the minimum and the maximum values of the data array exceeds

0 or 1 respectively, the `im2double` function rescales the output from integer data to a 0 to 1 scale. The power-law formula then transforms the image data to the specified gamma value. The `im2double` function is particularly useful for grey-scale intensities and removes the necessity to include another line of code to convert integer-based data to a 0 to 1 scale to fit the power-law formula. The function also allows for multiple image-type inputs, such as intensity, RGB, and binary. Finally, the updated data is displayed using the `imshow` function. imshow displays the matrix as an image and assumes that the data are pixel intensities.

Question 4

The file "GammaData.csv" contains light output measurements for a range of requested values for a particular display. Read in the data and plot the 'gamma' curve for this display.

The "plotLightOutput.m" function in question 2 can be used in the following manner:

```
GammaData = readmatrix('GammaData.csv');
plotLightOutput(GammaData(:, 1), GammaData(:, 2))
```

The dataset is read as a matrix as opposed to a table to omit any header text, without needing subsequent lines of code for processing. The `plotLightOutput` function then uses the first column as the `requestedSignal` argument and the second column as the `lightOutput` argument. The .png file of the plot is stored as "GammaDataPlot.png" in the working directory.

Question 5

To estimate the gamma value for this display, we need to fit a power-law curve to the data. The Matlab function fit is a powerful fitting function (available in the Curve Fitting

Toolbox, which you have access to through the university's student subscription–see the accompanying document on how to install additional toolboxes). The .m file "PowerLawExampleFit.m" provides an example of how to use this function to fit a power-law curve, using the "Nonlinear Least Squares" method with a specified starting value for the exponent. Use this example to write your own code to fit a power-law curve to the gamma measurements that you plotted in Q4. Since typical gamma values for displays are around 2, you'll want to update the starting value for the exponent to get the most reliable fitting behaviour. Save this in a .m file called "PowerLawFit".

The "PowerLawFit.m" function accepts x value data (requested signal) and y value data (recorded output) and plots the data points in comparison to a power-law curve generated using the `fit` function of the Curve Fitting Toolbox. This function utilises the nonlinear least-squares model and is an exponential type fit. Since the exponential model is convex/concave and will only provide one answer as the best fit, the inputted dataset will have one answer as the best fit as opposed to other nonlinear least-squares models with multiple possibilities. Thus, the most efficient starting point will be the one closest to "the correct answer". Since most screens will have a gamma of around 2, the starting exponent is set to 2 to make the algorithm more efficient. However, the function will run and find the correct fit speedily regardless of a starting point if omitted due to the limited nature of answers. The outputted figure will display the best-fitting gamma for the dataset, along with the curve and the data points.

Question 6

You will first need to generate a series of patterns made up of different proportions of white and black, ranging from 1/9 to 8/9 in steps of 1/9. The patterns used by Parraga et al. (2014) are shown in Figure 3. Each pattern is constructed from a 3-by-3 element, which is *then tiled to form a larger image. For our experiment, we'll make a larger image that is 600 by 600 pixels. We have provided a function called GenerateThreeByThreeElement which produces the appropriate 3-by-3 binary pattern for each proportion, specified with the input argument numWhite (which can take integer values from 1to 8). Utilising this function, write a new function or script that generates all eight test patterns(600-by-600 pixels each), and saves them as PNG files with filenames "Pattern1.png", "Pattern2.png", etc. You may find the built-in Matlab function repmat helpful.*

The "eightTest.m" function will create 8 patterns based on the white and black units created by the `GenerateThreeByThreeElement` function and no input argument is needed. For each pattern, the `for` loop will call a three-by-three element with an increasing proportion of white in each iteration. Once the element is created, it is repeated 200 times in each direction using the `repmat` function to create a 600 by 600-pixel pattern. The pattern is then outputted as a .png file, shown to the user in a figure window, and after a keystroke, will close and continue the next iteration of the loop. This process will allow the user to verify the patterns and continue without a build-up of multiple figure windows. The .png files will be saved as "PatternNumber.png", with "Number" representing the white proportion of the pattern in the white-black proportion (i.e. Pattern1.png for 1/9 white proportion) in the working directory.

Question 7

Have a look at the code in DummySetupInitialStimulus and DummyUpdateStimulus. They both also use a custom-written function called DrawDisc. You should be able to call these functions as follows:

[figWindowHandle, wPattern, hPattern] = DummySetupInitialStimulus(1, 0.2);

pause

DummyUpdateStimulus(figWindowHandle, wPattern, hPattern, 0.5)

Describe what happens and why? What's the benefit of having a DrawDisc function?

The script above in totality draws a grey disc with a specified grey-intensity of 0.2 over the halftone pattern with 1/9 ratio white to black, pauses and waits for user input, then draws a new disc over the same pattern with an intensity of 0.5. The first line of the script runs `DummySetupInitialStimulus` by accepting the desired white-ratio of the background pattern and the starting grey-level intensity of the disc. The function first checks for any out-of-bound values in the input arguments, then reads in one of the halftone patterns generated by `eightTest` that corresponds to the white-ratio specified in the input argument (i.e. Pattern1.png). It then calculates the height and width of the loaded pattern then opens a figure window and displays the halftone pattern. `DrawDisc` is then called with the generated height, width, and requested starting grey-level of the disc. The function then outputs the object of the figure window (`figWindowHandle`) and the width and height of the pattern square in pixels, which would be 600 by 600 (`wPattern`, `hPattern`). The pause function then waits for a keystroke from the user, then the `DummyUpdateStimulus` function is called by inputting the figure window object and the width and height of the background pattern and a newly specified grey-intensity of the disc, namely 0.5. The new disc is drawn by `DrawDisc`.

The script can be used to show a disc of a specified grey-intensity to a user to compare to the shade of the halftone pattern in the background. After the pause, the user can then compare the new grey-level intensity disc to see if the shade matches the background pattern. When this process is repeated throughout the whole grey-level intensity range, or at least until there is a perceived match, the function can identify the perceived grey-intensity of the halftone pattern. `DrawDisc` is particularly useful since it can draw a circle in the middle of the height and width of a given pattern regardless of size, as well as consistent proportion without having to repeatedly recreate the same code in different functions. The patch function within `DrawDisc` fills the entire polygon with one colour, ensuring that grey is produced rather than a white-black pattern. `viscircles` from the Image Processing Toolbox may seem as an easier alternative to draw circles within `DrawDisc` rather than a mathematical formula, however, one drawback is that the `viscircle` does not include a fill argument on its own and requires more lines of code. "AltDrawDisc.m" can be used instead if desired over the current `DrawDisc` function, and both can be used in the same manner. However, since `DrawDisc` only requires four lines of code, it is most likely more efficient.

Question 8

Using the Psychtoolbox functionality, write two analogous functions to present the experimental stimuli on a dedicated experiment screen (using the Screen function in Psychtoolbox). You should be able to demo the functions with the following lines of code (either by pasting both at the command prompt and then pressing "enter", or putting them in a script):

[windowPointer, wPattern, hPattern] = SetupInitialStimulus(1,0.2);

UpdateStimulus(windowPointer, wPattern, hPattern, 0.5)

The functions, "SetupInitialStimulus.m", "UpdateStimulus.m", and "CloseandTidy.m" can be used in the following manner:

```
[windowPointer, wPattern, hPattern] = SetupInitialStimulus(1, 0.2);
KbWait;
UpdateStimulus(windowPointer, wPattern, hPattern, 0.5)
KbWait;
CloseandTidy
```

`SetupInitialStimulus` receives the desired halftone pattern and grey-level as input arguments. The function checks for boundaries in input arguments and displays an error if the halftone range is not within 1-8 and grey-level is not within 0-1. It then opens a black

Psychtoolbox window if there are none presently open using the function "OpenPTBWindow.m". The specified halftone pattern is read (Pattern1.png), a hidden figure window is opened, and `AltDrawDisc` is called to draw the initially specified grey-level (0.2) over the halftone pattern. Since capturing the figure as an image requires output from and input into Matlab, the figure is then captured as a frame and converted as an image within Matlab and made into a texture to display on the Psychtoolbox window to use less processing power. This method ensures that the image is displayed true to its original dimensions and is shown as intended. To allow for integration with other functions, the function ends by flipping the drawn contents to the window and outputs the figure window object, width, and height of the halftone pattern. `KbWait` allows Psychtoolbox to wait for an input by the user to continue on to the next function while the window is still visible to the user, then `UpdateStimulus` takes the input arguments and checks whether the updated grey-level intensity is within range. If the intensity is out of range, the function returns to the Matlab command window and shows a warning and an option to either start over from the closest extreme grey-level within range or to quit the program. This is to account for key presses that may contribute to an accidental termination of an experiment and data loss if it is used within a loop and terminates from an error. A warning is used instead of the OS sound since users may not understand what they are doing wrong, may have accidentally been pressing the same button continuously, and would allow the script to continue after input. This error should not be a frequent issue. `UpdateStimulus` then uses the current Psychtoolbox window and draws a new disc as a texture with `AltDrawDisc` over the existing pattern. Finally, `KbWait` waits for the user input, then `CloseandTidy` closes all Psychtoolbox windows and terminates the program. Separating `CloseandTidy` allows for continuous display of stimuli in multiple contexts without closing the Psychtoolbox window.

Question 9

Write a function called GetKeyPress that collects a key press from the participant and returns the key that was pressed. The allowable keys should be: H, B, J, N and M.

The function, "GetKeyPress.m", requires no input arguments and collects a keypress from users and outputs it as the object, `character`. Accepted keys are H, B, J, N, M, and Q (to exit the grey-matching trial, if needed) and any other keypresses will issue an OS error noise until an acceptable keypress is entered. This function utilises "getkey.m" by Van der Geest (2019).[2] The `getkey` function will output the non-ASCII character of the first key pressed on the keyboard, meaning that the character itself is returned as an object instead of the ASCII code. `GetKeyPress` is flexible and can be used with Psychtoolbox or with the Matlab command window as long as it is paired with an instruction text using `Screen('DrawText')` or `disp()`.

Question 10

Write a function called RunTrial that runs a complete "method of adjustment" trial. This function should call the functions you have written in response to Q8 and Q9 above (i.e. SetupInitialStimulus, UpdateStimulus, GetKeyPress) or the equivalent dummy functions (i.e. DummySetupInitialStimulus, DummyUpdateStimulus).

The "RunTrial.m" function runs one grey-matching trial and returns the matched grey intensity level by calling `SetupInitialStimulus`, `UpdateStimulus`, `GetKeyPress`, and their dependent functions. The user must input the desired halftone pattern as an argument by entering the white ratio (e.g. 1 if 1/9). The function then randomly generates a grey-level between 1 and 255 for the disc, then draws the initial stimulus using

[2] This function can be downloaded from the Matlab File Exchange: https://www.mathworks.com/matlabcentral/fileexchange/7465-getkey

`SetUpInitialStimulus`, while "drawInstructions.m" provides on-screen instructions for completing the grey-matching procedure. `GetKeyPress` collects each keypress and a while loop evaluates the outputted character to see if it is H, a large brighter disc adjustment, B, a large darker adjustment, J, a small brighter adjustment, N, a small darker adjustment, M, a match, or Q, quit. The quit option is available since the larger experiment can take a long time, and participants may withdraw from it freely. A large adjustment represents 25/256 of available grey-levels by addition to avoid going between grey-levels, while a small adjustment represents 1/256. After each adjustment keypress, a pause is placed to clearly distinguish between the old grey-level and the new one, and `UpdateStimulus` is called to generate the same pattern with the newly defined grey-level. This process continues until M or Q is pressed. If M is pressed, a match message is shown on the screen with a short pause before continuing. When not using this function in a loop, please use the `CloseandTidy` function immediately after to close the Psychtoolbox window. If Q is pressed, the function will terminate and all Psychtoolbox windows will close. No data will be outputted to account for half-completed data sets.

Question 11 & 12

A single trial provides an estimate of the perceptual brightness match for one of the halftone patterns. Write a function or script to run a complete experiment that presents three repeats of each of the eight halftone patterns (i.e. 24 trials in total). The different halftone patterns should be presented in random order, but one repeat of the eight should be completed before starting on the second repeat, and each repeat should use a different random order. Save the brightness matches in a table with the following format, where grey-level indicates the grey-level of the match on that particular trial:

Pattern	Repeat1	Repeat2	Repeat3
1	*grey-level*	*grey-level*	*grey-level*
2	*grey-level*	*grey-level*	*grey-level*
3	*grey-level*	*grey-level*	*grey-level*
4	*grey-level*	*grey-level*	*grey-level*
5	*grey-level*	*grey-level*	*grey-level*
6	*grey-level*	*grey-level*	*grey-level*
7	*grey-level*	*grey-level*	*grey-level*
8	*grey-level*	*grey-level*	*grey-level*

Run your experiment to populate a results table and save the results in a .csv file, with a filename constructed as follows: 'PerceptualGamma_XXX.csv', where XXX is a three-letter participant identifier.

The function, "RunGreyMatching.m", accepts a participant ID entered as a string with single-quotation marks and conducts three repeat trials of grey-matching for every halftone pattern using `RunTrial`. A Psychtoolbox window is opened with `OpenPTBWindow`, then grey-matching trials using `RunTrial` will run with randomly organised patterns. The matched grey-level will be iteratively stored, and this process will repeat three times with a different random pattern for each iteration. A completion message is shown at the end of the experiment and the function waits for user recognition through keypress, collected data is labelled and outputted as "PerceptualGamma_participantID.csv" to the working directory, and the Psychtoolbox window is closed. For this question, "PerceptualGamma_123.csv" is outputted.

Question 13

The function GeneratePerceptualGammaData produces simulated perceptual gamma data for 20 participants. The function takes one input argument, which is used to seed the random number generator. Please use the numeric portion of your Bodleian Card number (above the barcode) as the input argument, so that everyone is working with a unique dataset. By explicitly seeding the random number generator, it is possible to recreate the *same datafiles by using the same seed. The gamma of the simulated display will be selected at random, and displayed as "Target Gamma".*

```
GeneratePerceptualGammaData(2916751)

%Generating results ...
%Target Gamma = 2.78
```

Question 14

Write a data plotting function or script to analyse the data from the 20 simulated participants. For each participant you should plot the raw data, plotting data from each of the three repeats on the same axes. Use the convention that the independent variable (the mean intensity of the halftone pattern) is on the x-axis, and the dependent variable (the grey-level of the brightness match) is on the y-axis. To create a multi-panel figure, you might want to use the Matlab function tiledlayout(available from 2019b) or subplot(available prior to 2019b, and continues to be available, but has less functionality for customising the layout).

The "plotGamma.m" function plots the 20 participants' data from `GeneratePerceptualGammaData` using `tiledlayout` without an input argument. The .csv files are read for each participant as identified in the above function. Three halftone patterns are plotted in the x-axis, the matched grey-level in the y-axis, and each repeat as a line for each participant. The x and y labels are globally placed and legends are hidden to save visual space. In addition, x and y limits are placed to account for the whole of grey-levels. The plots are displayed on a full figure screen.

Question 15

Write a data analysis function or script to combine the data from all participants. Foreach participant you should calculate the mean brightness match for each halftone *pattern, and then across participants, calculate the mean and standard deviation of these estimates. Plot the average data with errorbars representing +/-1 S.E..*

The “gammaAnalysis.m” function takes the identifiers from `GeneratePerceptualGammaData` for each participant and loads the .csv files without an input argument. The grey-level average is calculated across each pattern for each participant, then the grand mean is calculated from all the means for each pattern. Standard errors are calculated for each halftone pattern. The data is labelled and outputted as “PerceptualGammaData.csv” in the working directory. Participant mean data is in each column, the 21st column contains the grand means, and the 22nd column contains the standard errors.

Question 16

Finally, use your average data to estimate the gamma of the display that produced the simulated data. Note that the convention is that “gamma” is defined as the exponent in the power-law relationship between requested light level on the x-axis (i.e. the grey-level value) and the objectively measured light output on the y-axis (i.e. the average intensity of the halftone pattern in this case). How does your fitted value compare to the Target Gamma displayed in the GeneratePerceptualGammaData function?

The “fitMeasuredGamma.m” function takes a path or file name for a .csv file generated by `gammaAnalysis` as an input argument, reads the grand means, calculates the objective grey-levels for each halftone pattern (e.g. 1/9 = 0.111) then plots the data against a modelled gamma-curve using `PowerLawFit` from question 5 with the x-axis as the measured grey-levels and the y-axis as the objective levels. The fitted value is the same value as the one displayed in GeneratePerceptualGammaData. However, these values may not be exactly the same due to rounding errors in the datasets or the noise generated in the function above for the participants.

References

Van der Geest, J., 2019. getkey.m.

https://www.mathworks.com/matlabcentral/fileexchange/7465-getkey