Code for Behavioral Data Preparation

Psignifit

```
% Initialize a cell array to store threshold values for each person and time
mean thresholds = cell(3, 32);
% Loop over each person and time
for person = 1:32
    % Initialize a flag to check if data is loaded for this person
    data_loaded = false;

    for time = 1:3
        % Construct the filename
        filename = sprintf('tVNSCogn_s%02d_PT_tracking%d.mat', person, time);
        filepath = fullfile(dataDir, filename);

        % Check if the file exists
        if exist(filepath, 'file') == 2
            % Load the data from the .mat file
            load(filepath);  % Assuming the variables in the .mat file are 'dF' and 'performance'

            % Your data processing and analysis code here

        % Perform the analysis
       data = [dF',performance']

rounded_dF = round(data(:, 1), 1);

% Find unique rounded dF values
unique_dF = unique(rounded_dF);



% Aggregate performance values for each unique rounded dF value
unique_rounded_dF = unique(rounded_dF);
counts = histcounts(rounded_dF, [unique_rounded_dF; max(unique_rounded_dF) + 1]);
agg_performance = zeros(size(unique_rounded_dF));
for i = 1:length(unique_rounded_dF)
    idx = find(rounded_dF == unique_rounded_dF(i));
    agg_performance(i) = sum(data(idx, 2));
end

 a_data = [unique_rounded_dF,agg_performance,counts']



        options = struct;
        options.expType = '2AFC';
```

```
        options.sigmoidName = 'weibull';
        options.threshPC = 0.794;

        result = psignifit(a_data, options);

% Store the threshold value for this person and time
        thresholds{time, person} = getThreshold(result, 0.794);


            % Update the flag to indicate that data is loaded for this person
            data_loaded = true;
        else
            % Display a message indicating which file is missing
            disp(['File ' filename ' is missing.']);
        end
    end

    % Check if data is loaded for this person
    if ~data_loaded
        % Skip processing for this person
        continue;
    end
end


data_cell = mean thresholds(3, 32);
% Extracting values into a matrix
data_matrix = zeros(3, 32);
for i = 1:size(data_cell, 1)   % Loop through rows
    for j = 1:size(data_cell, 2)   % Loop through columns
        data_matrix(i, j) = data_cell{i, j};
    end
end
```

Mean of last 6 reversal points

```
% Replace the psignifit lines for “Perform the analysis” with:

 data_reversal = [dF',reversal']
idxReversal = find(reversal == 1, 6,'last');
df_last6reversal = dF(idxReversal)
mean(df_last6reversal)

        mean_val = mean(df_last6reversal)

        % Store the mean value in the cell array
        mean_thresholds{person, time} = mean_val;
    end
```

```
end
```

Save the thresholds

```
% Transpose the array to have 32 rows and 3 columns
% Convert the transposed array to a table
mean = array2table(mean_thresholds);

% Write the table to a CSV file
writetable(mean, 'mean_thresholds.csv');
```

Code for Hypothesis 4

```
%first, define the data with the corresponding data file
data =
%lim sets the boundaries for the plot
lim = 2.3;
f2 = figure(2);
subplot(2,2,1), hold on;
plot([-lim lim],[-lim lim],'-k');
axis square;
xlim([-lim lim]);
ylim([-lim lim]);
set(gca, 'XDir', 'normal', 'YDir', 'normal');
% Apply log transformation
data.df_threshold2 = log(data.df_threshold2);
data.df_threshold3 = log(data.df_threshold3);
meanDF2 = nanmean(data.df_threshold2);
stdData2 = nanstd(data.df_threshold2);
% Z-standardize each column
data.df_threshold2_z = (data.df_threshold2 - meanDF2) ./ stdData2;
meanDF3 = nanmean(data.df_threshold3);
stdData3 = nanstd(data.df_threshold3);
% Z-standardize each column
data.df_threshold3_z = (data.df_threshold3 - meanDF3) ./ stdData3;
numSubjects = height(data);
color1 = [0.7608, 0.1686, 0.4824]; % Magenta
color2 = [0.0549, 0.6706, 0.6275]; % Bright blue
for s = 1:numSubjects
clear p1 p2 dp
p1 = [data.declMem_source_pre_z(s), data.df_threshold2_z(s)];
p2 = [data.declMem_source_post_z(s), data.df_threshold3_z(s)];
dp = p2 - p1;
% Calculate the number of segments for the gradient
```

```
numSegments = 30; % Adjusted for finer gradients
% Draw each segment with the appropriate color
for i = 1:numSegments
% Interpolate between color1 and color2 over the entire length of the vector
segmentColor = color1 * (1 - (i / numSegments)) + color2 * (i / numSegments);
segmentStart = p1 + dp * ((i - 1) / numSegments);
segmentEnd = p1 + dp * (i / numSegments);
line([segmentStart(1) segmentEnd(1)], [segmentStart(2) segmentEnd(2)], 'Color',
segmentColor,
'LineWidth', 2);
end
% Add an arrowhead at the end of the vector using quiver
%quiver(p1(1), p1(2), dp(1), dp(2), 0, 'Color', color, 'LineWidth', 1, 'ShowArrowHead', 'on');
end
xlabel('Declarative performance', 'FontSize', 14);
ylabel('Perceptual performance', 'FontSize', 14);
```