Simulation

Table of Contents

Ground setting up 1
Generating data under differnet conditions 1
Calculate d-prime for each stimulus level 2
Plotting 3
Functions 7

Ground setting up

```
clear all; close all; clc
nTrials = 10000; %overall number of trials
%Signal types: 0 or 1; 0 for left, 1 for right

stimLevel = linspace(1, 7, 7); %stimulus bandwidth, here spatial frequency as
 example, 1-7 cpd, 7levels
nLevel = length(stimLevel); %overall number of levels (7)

%constNoise = true; %constant noise (1)
```

Generating data under differnet conditions

```
%overall 2 situations (with noise or without noise)
for idx=1:2
    switch idx
        case 1
            mode = 'YN';
            data.(mode) = simulate2AFC(nTrials, stimLevel, 1);
        case 2, mode = 'NN';
            data.(mode) = simulate2AFC(nTrials, stimLevel, 0);
    end
end

%%generate condition data files
for idc = 1:3 %3 conditions (valid, invalid, neutral)
    switch idc
        case 1
            cond = 'Valid'; %attentional cue = response cue
        case 2
            cond = 'Invalid';
        case 3
            cond = 'Neutral';
    end

    %data(i, :) = [level, attenCue(i), respCue(i), tarGabor(i), decisions(i)];
    for n = 1:nTrials
        for s = 1: nLevel
```

```
            level = stimLevel(s);
            if idc == 1
                conIdx = data.YN(:,2) == data.YN(:,3); %valid
            elseif idc == 2
                conIdx = data.YN(:,2) ~= 2 & data.YN(:,2) ~=
 data.YN(:,3); %invalid
            elseif idc == 3
                conIdx = data.YN(:,2) == 2; %neutral
            end

            nConidx = sum(conIdx == 1);
            conData.(cond) = nan(nConidx, 3); %[stim level, target gabor,
 response]
            conData.(cond)(:, 1) = data.YN(conIdx == 1, 1);
            conData.(cond)(:, 2) = data.YN(conIdx == 1, 4); %target gabor left
 or right
            conData.(cond)(:, 3) = data.YN(conIdx == 1, 5); %response left or
 right
        end
    end
end
```

Calculate d-prime for each stimulus level

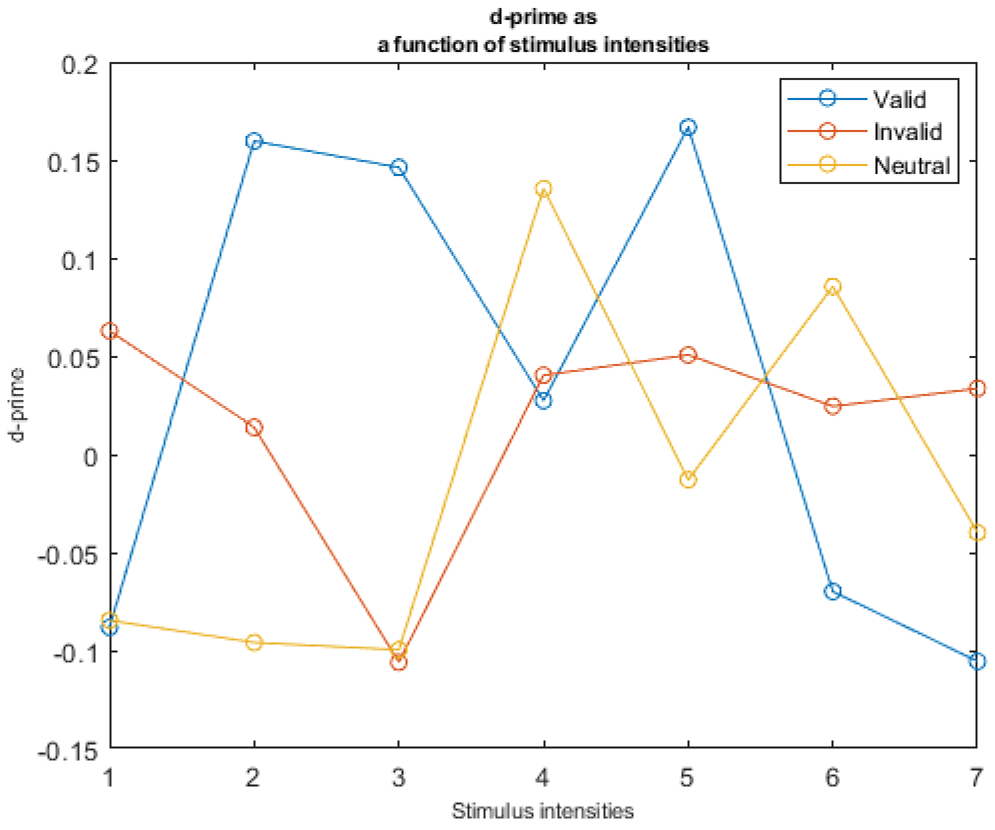
```
    dPrimes.valid = zeros(nLevel, 6);
    dPrimes.invalid = zeros(nLevel, 6);
    dPrimes.neutral = zeros(nLevel, 6);

    hit_idx_valid = conData.Valid(:,2) ==0 & conData.Valid(:,3) == 0;
    false_alarm_idx_valid = conData.Valid(:,2) == 1 & conData.Valid(:,3) == 0;

    hit_idx_invalid = conData.Invalid(:,2) ==0 & conData.Invalid(:,3) == 0;
    false_alarm_idx_invalid = conData.Invalid(:,2) == 1 & conData.Invalid(:,3)
 == 0;

     hit_idx_neutral = conData.Neutral(:,2) ==0 & conData.Neutral(:,3) == 0;
     false_alarm_idx_neutral = conData.Neutral(:,2) == 1 &
 conData.Neutral(:,3) == 0;

     %invalid condition
for k = 1:nLevel
    levelidx = stimLevel(k);
    dPrimes.invalid(k,1) = levelidx;
    dPrimes.invalid(k,2) = sum(conData.Invalid(:,1) == levelidx &
 hit_idx_invalid == 1); %hit
    dPrimes.invalid(k,3) = sum(conData.Invalid(:,1) == levelidx &
 false_alarm_idx_invalid == 1); %false alarm
    dPrimes.invalid(k,4) = norminv(sum(conData.Invalid(:,1) == levelidx
 & hit_idx_invalid == 1)/sum(conData.Invalid(:,1) == levelidx))-
norminv(sum(conData.Invalid(:,1) == levelidx & false_alarm_idx_invalid == 1)/
sum(conData.Invalid(:,1) == levelidx));
    dPrimes.invalid(k,5) = sum(conData.Invalid(:,1) == levelidx &
 conData.Invalid(:,2) == conData.Invalid(:,3)); %hit+correct reject
```

```
    dPrimes.invalid(k,6) = sum(conData.Invalid(:,1) == levelidx); %# of trials
 per stim level
end

%neutral condition
for k = 1:nLevel
    levelidx = stimLevel(k);
    dPrimes.neutral(k,1) = levelidx;
    dPrimes.neutral(k,2) = sum(conData.Neutral(:,1) == levelidx &
 hit_idx_neutral == 1); %hit
    dPrimes.neutral(k,3) = sum(conData.Neutral(:,1) == levelidx &
 false_alarm_idx_neutral == 1); %false alarm
    dPrimes.neutral(k,4) = norminv(sum(conData.Neutral(:,1) == levelidx
 & hit_idx_neutral == 1)/sum(conData.Neutral(:,1) == levelidx))-
norminv(sum(conData.Neutral(:,1) == levelidx & false_alarm_idx_neutral == 1)/
sum(conData.Neutral(:,1) == levelidx));
    dPrimes.neutral(k,5) = sum(conData.Neutral(:,1) == levelidx &
 conData.Neutral(:,2) == conData.Neutral(:,3)); %hit+correct reject
    dPrimes.neutral(k,6) = sum(conData.Neutral(:,1) == levelidx); %# of trials
 per stim level
end

%valid confition
for k = 1:nLevel
    levelidx = stimLevel(k);
    dPrimes.valid(k,1) = levelidx;
    dPrimes.valid(k,2) = sum(conData.Valid(:,1) == levelidx & hit_idx_valid ==
 1); %hit
    dPrimes.valid(k,3) = sum(conData.Valid(:,1) == levelidx &
 false_alarm_idx_valid == 1); %false alarm
    dPrimes.valid(k,4) = norminv(sum(conData.Valid(:,1) == levelidx
 & hit_idx_valid == 1)/sum(conData.Valid(:,1) == levelidx))-
norminv(sum(conData.Valid(:,1) == levelidx & false_alarm_idx_valid == 1)/
sum(conData.Valid(:,1) == levelidx));
    dPrimes.valid(k,5) = sum(conData.Valid(:,1) == levelidx &
 conData.Valid(:,2) == conData.Valid(:,3)); %hit+correct reject
    dPrimes.valid(k,6) = sum(conData.Valid(:,1) == levelidx); %# of trials per
 stim level
end
```

Plotting

```
    figure
    plot(stimLevel, dPrimes.valid(:,4), '-o'); hold on;
    plot(stimLevel, dPrimes.invalid(:,4), '-o'); hold on;
    plot(stimLevel, dPrimes.neutral(:,4), '-o'); hold on;


    xlabel('Stimulus intensities','FontSize', 8);
    ylabel('d-prime','FontSize', 8);
    title({'d-prime as', 'a function of stimulus intensities'}, 'FontSize',8);
    legend('Valid ', 'Invalid', 'Neutral','Location', 'best');
```

```
% disp(message);
%
% % Obtain bootstrap errors
% [SD, trash, trash, trash] = PAL_PFML_BootstrapParametric(...
%          StimLevels, OutOfNum, PFparams, PFparamsFree, Bse, PFfunc, ...
%          'searchOptions',options,'searchGrid', searchGrid);
%
% % Determine goodness-of-fit
% [Dev, pDev] = PAL_PFML_GoodnessOfFit(StimLevels, NumPos, OutOfNum, ...
%      PFparams, PFparamsFree, Bmc, PFfunc,'searchOptions',options, ...
%      'searchGrid', searchGrid);
% end
```



Functions

```
%Generate raw data file, assuming contant noise and no interval bia
function data = simulate2AFC(nTrials, stimLevel, constNoise)
nLevel = length(stimLevel); %overall levels
data = nan(nTrials, 5); % [stimLevel, attentional cue, response cue, left
 gabor, right gabor,response]
actual = rand(nTrials) > 0.5;
decisions = nan(nTrials, 1);
attenCue = randi([0,2], nTrials, 1); %0 si left, 1 is right, 2 is both
respCue = rand(nTrials,1) > 0.5;
tarGabor = rand(nTrials,1) > 0.5;
```

```
for i = 1:nTrials
    % Randomly select a stimulus level for each trial
    levelIndex = randi(nLevel);
    level = stimLevel(levelIndex);

    %Generate noise
    if constNoise == 1
        noise = 0.1;
    else
        noise = 0;
    end


    % Generate response
    evidence = 0;
    if actual(i) == 0 % left
        evidence = evidence + normrnd(level, 1) + noise;
    else % right
        evidence = evidence + normrnd(-level, 1) + noise;
    end

    % The collected evidence over the entire duration is summed up.
    % If the total evidence is positive, it's decided that left; otherwise,
 right
    decisions(i) = evidence > 0;

    % Store the level and response
    data(i, :) = [level, attenCue(i), respCue(i), tarGabor(i), decisions(i)];
end
end

% %Sorting data file
% function pfFile = sortpc(stimLevel, data)
% nLevel = length(stimLevel);
% pfFile = nan(10, 3);
% for l = 1: nLevel
%     level = stimLevel(l);
%     pfFile(l, 1) = level;
%     pfFile(l, 2) = sum(data(:,1) == level & data(:,3) == data(:,2));
 %Correct
%     pfFile(l, 3) = sum(data(:,1) == level); %Overall trial
% end
% end
```

Published with MATLAB® R2022b