

This document contains notes regarding running the analysis code that was used for the following paper:

Keshvari S, Van den Berg R, Ma WJ (2013)

No evidence for an item limit in change detection

PLoS Computational Biology 9(2): e1002927. DOI: [10.1371/journal.pcbi.1002927](https://doi.org/10.1371/journal.pcbi.1002927)

Before you start

Make sure to have the circular statistics toolbox by Phillip Behrens installed before running code. Note that the term "fake" when referring to data in these functions refers to fake data generated by the models in order to test whether the analysis returns the correct predictions for Bayesian model comparison and ML parameter estimation. On the other hand, `predictions_ort.mat` and `predictions_color.mat` are used in generating tables of model predictions without having to loop over all subject trials. Note that the order of the model numbers tested in the manuscript is [5 4 17 16 15], for [IP SA SR EP VP], respectively. Model number 7 is used for the guessing-rate analysis (Section in the paper titled "Apparent guessing as an epiphenomenon").

Typical procedure for analysis of orientation-change and color-change data

Make sure that your path includes all of the subfolders for the analysis.

The file `analysis_example.m` shows an example of a full analysis. It first generates model predictions, then fits the models to the subject data, and finally creates plots of these fits.

% Generate model predictions for each model and set size. This could all be done completely in parallel per set size and model. Note that model 15 (VP) takes quite a bit of memory and time to run for each set size.

```
num_samples = 1000;
temp_subj_num = 1;
for model_num = [5 4 17 16 15]
    for set_size_idx = 1:4
        % orientation-change
        is_color = 3;
        run_model_a_on_subj_b_with_c_samples_new(model_num, temp_subj_num,
num_samples, is_color, set_size_idx);
        % color-change
        is_color = 4;
        run_model_a_on_subj_b_with_c_samples_new(model_num, temp_subj_num,
num_samples, is_color, set_size_idx)
    end
end
```

% Compute the log-likelihoods and maximum-likelihood fits to each subject. Note that for model 15 (VP), this takes up about 8GB of RAM per subject. There are 10 subjects for orientation-change data, 7 for color-change.

```

run_num = [];
for model_num = [5 4 17 16 15]
    for subj_num = 1:10
        is_color = 0;
        get_subj_LL_cluster_par(model_num,is_color,run_num,subj_num);
    end
    for subj_num = 1:7
        is_color = 1;
        get_subj_LL_cluster_par(model_num,is_color,run_num ,subj_num);
    end
end
end
% Generate the psychometric curves for orientation and color
load subjCell
load subjColorCell
num_delta_bins = 11;
model_nums = [5 4 17 16 15];
is_color = 0;
[p_C_hat_mat HR FA] =
compute_psych_curves_multiple(subjCell,num_delta_bins,model_nums,is_color)
is_color = 1;
figure
[p_C_hat_mat HR FA] =
compute_psych_curves_multiple(subjColorCell,num_delta_bins,model_nums,is_color)

% Compute and plot BMC results
drawBars_new()

```

Analysis Functions:

run_model_a_on_subj_b_with_c_samples_new(model_num,subj_num,samples,is_color,set_size_idx)

- Given the model number defined in parameter_ranges_new, the subject number from subjCell, the color index (see the "switch" at the beginning of the function), and the set size to test (for orientation-change experiment, use 1, 2, 3, or 4 for set sizes 2, 4, 6, or 8, respectively. For color-change experiment, use 1, 2, 3, or 4 for set sizes 1, 2, 4, or 8, respectively), produce the probability of responding "change" (P_C_HAT) for each trial. To create a table of predictions for all possible trials (this is the most efficient way to do the analysis), set "is_color" to 3 or 4 (orientation or color data, respectively) and set subj_num to 1. Output is saved in "P_C_HAT" folder.

EXAMPLE: run_model_a_on_subj_b_with_c_samples_new(15,1,1000,3,1);

will run VP model on a all possible orientation-change trials with set size 2, and save the output.

run_model_x(Data,[various parameters])

- where "x" is the ID of the model to be tested. Used by

"run_model_a_on_subj_b_with_c_samples_new" to generate model predictions.

qinterp1Lin(spacing,Y,xi)

- Quick linear interpolation function for use with uniformly spaced X. "spacing" is a scalar that is the reciprocal of the spacing between two elements of X, e.g. $1/(X(2) - X(1))$. Y is the function value (column vector) and xi are the values to be looked up (column vector).

qinterp1(spacing,Y,xi)

- Quick lookup function for use with uniformly spaced X. "spacing" is a scalar that is the reciprocal of the spacing between two elements of X, e.g. $1/(X(2) - X(1))$. Y is the function value (column vector) and xi are the values to be looked up (column vector).

myBessel(X,spacing,lookupY)

- Quick lookup for the modified bessel function of the first kind, $\nu == 0$. "X" are the values to be looked up, "spacing" is the reciprocal of the spacing in the linearly spaced function arguments (as in qinterp1 and qinterp1Lin), and "lookupY" are the values of the modified bessel function at the function arguments (similarly to "Y" in qinterp1 and qinterp1Lin).

load_ML_fits(model_num,is_color)

- Load data generated for each subject given their ML fits from a particular model "model_num" and experiment type "is_color"

get_subj_LL_cluster_par(model_num,is_color,run_num,subj_num)

- Get the log-likelihood and ML fits for a given model "model_num" and experiment type "is_color" (0 for orientation-change, 1 for color-change) for subject "subj_num" and save them. This accepts a run number "run_num", which is useful when multiple sets of fake data are generated (e.g. for the AGR analysis) and you only want to generate log-likelihoods and ML fits for one set of fake data. If not using different runs, set run_num to an empty matrix [].

[A B C D] = get_max_params(model_num,is_color)

- Get the ML parameters determined by the ML fits for a given model "model_num" and experiment type "is_color" (0 for orientation-change, 1 for color-change) for all subjects. The format of the output will depend on the particular model's parameters.

[A B C D] = get_max_params_cluster(model_num,is_color,run_num)

- Get the ML parameters determined by the ML fits for a given model "model_num" and experiment type "is_color" (0 for orientation-change, 1 for color-change) for all subjects. This accepts a run number "run_num", which is useful when multiple sets of fake data are generated (e.g. for the AGR analysis) and you only want to load one set of fake data. If not using different runs, set run_num to an empty matrix [].

[p_C_hat_mat HR FA] =

compute_psych_curves_multiple(Subj_Data_Cell,num_delta_bins,model_nums,is_color)

- Generate the psychometric curves for all subjects, given the data cell "Subj_Data_Cell", number of bins over delta "num_delta" (see the generative model). "HR" and "FA" are the hit and false-alarm rates, and "p_C_hat_mat" is a matrix of the proportion reports 'change' as a function of set size and delta. "model_nums" is a vector containing the models to test, and "is_color" is 0 for orientation-change data and 1 for color-change data;

[LL_model] = compute_BMC(model_vec,is_color)

- Compute the posterior of each model for each subject (note that "LL_model" is a misnomer, this function actually computes the posterior) for use in Bayesian model comparison (BMC). "model_vec" is a vector with the model numbers to compute, "is_color" determines the experiment type. Note that this can take up a large amount of memory when testing models that use independent parameters per set size.

AGR_analysis(skip_analysis)

- Generate fake data for each model using ML parameters from the actual data, and fit the apparent guessing rate (AGR) model to this fake data. "skip_analysis" determines whether to skip the fake data generation and just plot the results. Note that using this function requires commenting and un-commenting code within it.

[bars_mean,bars_std,bars_color_mean,bars_color_std] = draw_bars_new()

Compute BMC for both experiments and plot the results. Outputs are the mean and standard error for the BMC values over subjects for each model.

[c] = cols(x)

Return the number of rows in x.

s = row_sum(x)

Sum for each row in x.