Computational Cognitive Neuroscience: Assignment 2
Computational modelling of behavioural data

Xiao Heng (s2032451)

February 22, 2024

(a) Exploring the data

By computation, the mean of STAI-Y2 scores is **45.42**, with corresponding standard deviation **15.92** and median **42.50**. Following the threshold of $STAI \leq 43$, the number of participants considered as healthy controls is exactly summed up to 25, and most of them are the recruited as the calm population (last 25 indices); however, the partition is not completely perfect, as the **22nd** participant is counted due to a low level of STAI-Y2 score (39); meanwhile, the **27th** participant is not counted, with a rather high level of STAI-Y2 score (45). Generally, the participants fit in the correct group based on the cutoff rule.

Then, by compute the number of times each participant choose option A, we find the average of this number across all subjects is **38.4** (or **24%**, in percentage). Under a random respond strategy, the expected number of aversive sounds experienced by participants would be **106**.

(b) Simulations

The participants' behaviours are modelled with a series of different reinforcement learning models. In the first model, value of the chosen stimulus i is updated on trial t after observing outcome o (which will be 0 for no sound and 1 for sound) as follows:

$$V_i^{(t+1)} = V_i^{(t)} + \alpha \times (o^{(t)} - V_i^{(t)})$$

And the probability of choosing stimulus A as opposed to stimulus B on trial t is modelled using a softmax function:

$$p(action\ A|V^{(t)}, \beta) = \frac{e^{-\beta \times V_A^{(t)}}}{e^{-\beta \times V_A^{(t)}} + e^{-\beta \times V_B^{(t)}}}$$

Under this model, we did $n = 10000$ times of simulations, with 160 trials (during which the probability of each stimulus to lead to the aversive noise were changed every 40 trials according to the following values: 60/40, 80/20, 60/40, 65/35) as well as parameter settings of $\alpha = 0.3$, $\beta = 8$ and $V_0 = 0.5$. The average evolution of values $V(A)$, $V(B)$ and their difference on average $V(A) - V(B)$ are shown in **figure 1**.

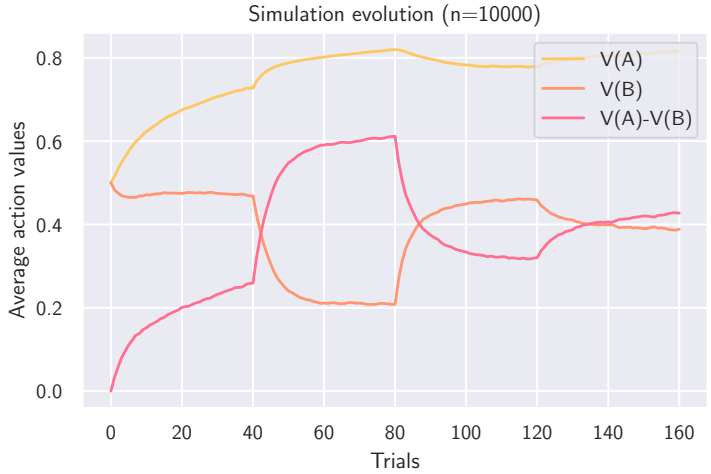

Figure 1: The average evolution of values $V(A)$, $V(B)$ and corresponding difference $V(A) - V(B)$.

Roughly speaking, the update rule is a learning process based on historical experience. From **figure 1**, we could notice that the trend of curve changes every 40 trials, due to the hidden mechanism of aversive noise probability. For the first 40

trials, $V(A)$ increases (to the direction of 1) and $V(B)$ decreases little (to the direction of 0), because stimulus A leads a higher possibility of aversive noise (60%); and such opposite move also causes the increase of $V(A) - V(B)$. Then, in the following 40 trials, the $V(A)$ leads to aversive noise in a higher possibility than before (80%). Since $V(A)$ is laeger than $V(B)$ now, in most cases, participants will choose action B (based on the softmax choice mechanism), obtaining positive feedbacks in high possibility, decreasing the $V(B)$ rapidly. Another reason for the slow increasing speed of $V(A)$ is that, its value has already reaches round 0.8, which is closed to 1, so that the update difference $o^{(t)} - V_i^{(t)}$ now is small, leading slight change in updating. Then in the following two stages, the case is similar, and the change of value as well as moving speed (slope) are both mainly affected by the relative probability of leading the aversive noise between action A and B (e.g., in the 3rd stage, the probability gap decreases into the initial level, so the difference of $V(A) - V(B)$ also decrease, but still higher than the level at trial 40, the end of stage 1, due to the historical experience during stage 2. In other word, the effect of historical experience remains.).

(c) Exploring parameter settings

With a sensible parameter range $0 < \alpha < 1$ and $0 < \beta < 10$, we simulate the 160 trials with $n = 1000$ times for each parameter combination. Under a method similar to grid search method, we set the "grid" as 5%, so in each range, we test 19 different values of parameter (e.g., for α, we test 0.05 to 0.95). The result is shown in **figure 2**.

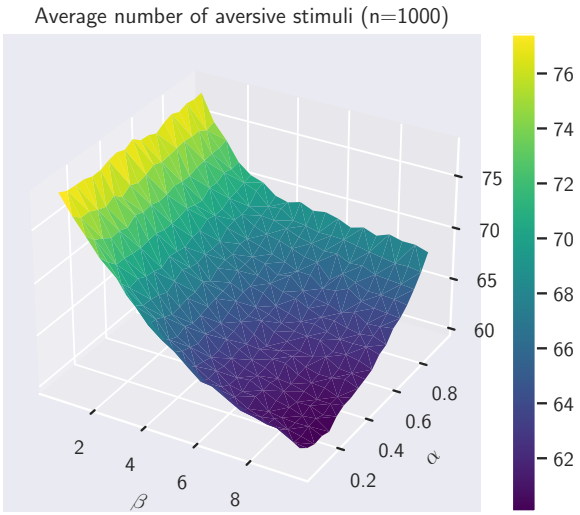

Figure 2: Average number of aversive stimuli under different parameter settings ($0 < \alpha < 1$ and $0 < \beta < 10$)

The surface of graph is really smooth, and we can easily notice that the "low α" and "high β" is the best combination, leading global minimum (lowest average outcome of aversive stimuli) within this parameter range. The intrinsic reason is considered that a low α (as is commonly regarded as "learning rate" in reinforcement learning) study from the experience lower (in each trial, it leads to a smaller change in $V(\cdot)$), but more steady. Since this is a probabilistic generative mechanism, noise exists, and a cautious learning strategy might be preferable.

At the same time, a larger β (personally, I would relate it with "ϵ-greedy strategy" in reinforcement learning, as under a given action value, the β or ϵ would affect the probability of choosing the "greedy" way, or best valued action) in the softmax function will lead a more deterministic choice of action with lower value (more precisely, its effect is based on the difference of two actions' values). Since in our assumption or initial setting that the responding mechanism is rather stable (although it shifts every 40 trials, action B always leads a lower probability of aversive sound respond), so a more deterministic or "greedy" choice based on updated value would perform better.

(d) Likelihood function

To estimate the best fitted parameters based on given choices and outcomes, negative log likelihood (NLL) method is applied:

$$NLL = -\sum_{c \in Choices} \log p(c|V, \theta)$$

Under the same parameter settings of $\alpha = 0.3$, $\beta = 8$ and $V_0 = 0.5$ as before, we implement the NLL and calculate the 1st and 10th participants' NLL values as **50.19** and **58.60** respectively.

(e) Model fitting

Based on the NLL in last section, to find the best fitted parameters, we adopt the **scipy.optimize.minimize()** function (**Nelder-Mead** method) to minimize the NLL function. By calculation, the mean and variance of learning rate and inverse temperature are shown in **table 1**:

	mean	var
α	0.42	0.01
β	5.28	3.04

Table 1: The fitted α and β among all participants.

The specific fitting results of each participant are shown in **figure 3**. Note that, to normalize the values of β into the same range of values of α, we multiply a coefficient of 0.1 on it.

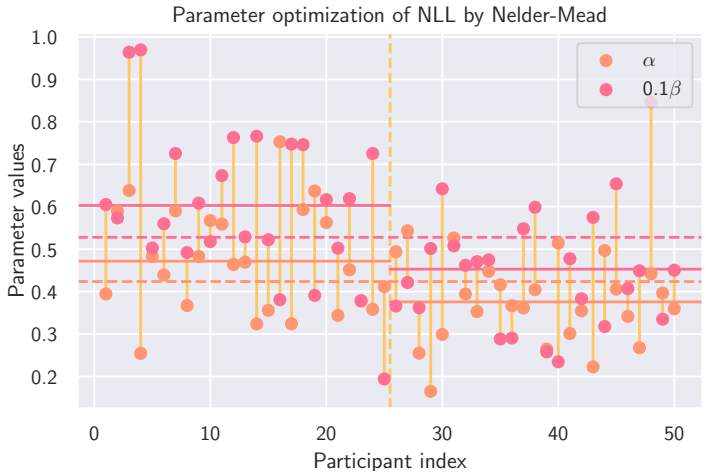

Figure 3: The fitting results. The vertical dashed line in the middle of figure is a partition of two groups (anxious and calm groups). The horizontal dashed lines are the mean among all participants. The horizontal solid lines are the mean of either anxious or calm group, respectively.

From the mean of sub-groups (horizontal solid lines), we can easily notice that the behaviour of anxious group are fitted with both higher α and β, indicating a potential higher learning rate (study faster on experience) and a higher inverse temperature (make choice more deterministically). Since all the parameters are within a sensible range ($0 < \alpha < 1$ and $0 < \beta < 10$), it could be regarded as expected results.

As for the Pearson's correlation coefficient between α and β, the value across all participants is 0.07, while -0.13 for group 1 and -0.1 for group 2. In this case, the Pearson's correlation coefficient is always at a low level, indicating that both parameters (α and β) can explain defferent parts of mechanism in the model, and can not easily replace the other.

(f) Group comparison

To test whether the estimated parameter values in the last section are significantly different across groups, we calculate the mean as well as t-statistic and p-value of α and β between the two groups, shown in **table 2**. The corresponding degrees of freedom could be obtained from $50 - 2$, which is **48**, as the sample size is 50 and parameter number is 2 in the t-test.

	mean(1)	mean(2)	t-stat	p-value
α	0.47	0.38	3.03	0.00
β	6.03	4.53	3.30	0.00

Table 2: The t-statistic and p-value of α and β between two different groups (anxious and calm).

Though there is indeed different in the value of parameters between two groups, to prove the statistically difference, the p-values of t-test, which are both 0, tell us that the null hypotheses (the two mean are statistically the same, with the same expectation) is rejected, so now we have the confidence to say that the estimated mean of parameters between two groups are statistically different, satisfying the early assumptions that, trait anxiety could be related to changes in avoidance learning (i.e. an altered learning rate) or some change in decision making (inverse temperature).

(g) Parameter recovery

To check the reliability and identifiability of our parameter estimates, first, we sample 50 sets of parameters. In the multivariate normal distributions with 0 covariance, we need to specify mean and variance. Intuitively, we set the two different mean of parameters in two different groups which are already obtained previously as the mean now we want to simulate (in this case, we still seperate the 50 simulated participants into 2 groups, with different mean). Then for the variance, we follow the instruction, as 0.01 and 0.5 for α and β respectively. Notice that, to ensure the sampled parameters are not extremely large or small (it indeed has a low probability to happen during the sampling from arbitrary distribution), based on prior knowledge from previous sections, every sampled value will be checked. The threshold is set as 95% confidence interval in normal distribution ($mean \pm 1.96 \times std$). If a sampled value violates this "constraint", then it will be re-sampled, until the threshold is satisfied. This ensure us to always obtain sensible values. The sampling result is shown in **figure 4**, and the parameter sampling (in the left sub-plot, light purple and dark purple dots present the nonsensical values of α and β, which will be re-sampled) as well as correlation between simulated and fitted parameters in the last (5th) iteration are shown in **table 3**.

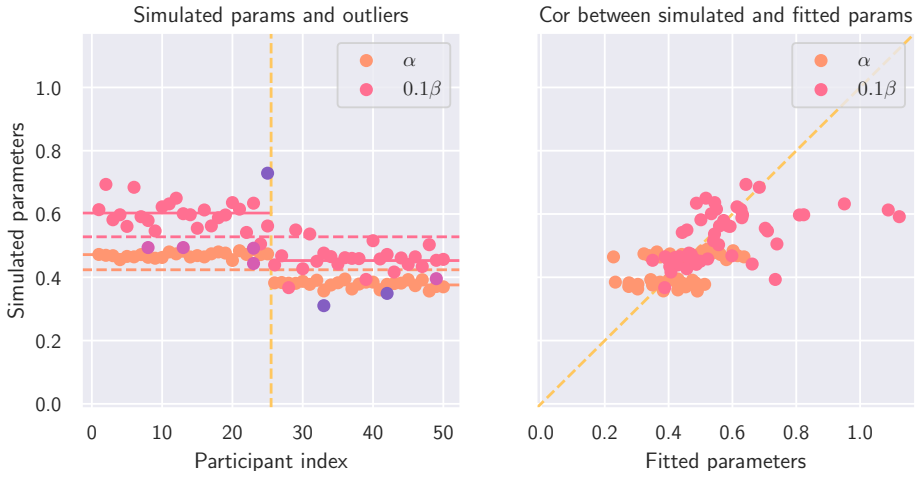

Figure 4: Pearson's correlation between simulated and fitted parameters under different number of trials and number of simulated data sets.

	(1)	(2)	(3)	(4)	(5)
$cor(\alpha)$	0.25	0.38	0.40	0.57	0.43
$cor(\beta)$	0.70	0.68	0.65	0.73	0.50

Table 3: The Pearson's correlation from 5 times of parameter recovery repeats.

From the results, we notice that in most cases, the correlation between simulated and fitted β is higher than the case of α, indicating that the decision process may be easier to implement than the learning process. As for further exploration of the effect from number of trials and the number of simulated data sets, which influence the performance of the parameter recovery, see **figure 6**. We test different combination of number of participants and number of trials; as for the number of participants, we choose all even number that include half of anxious participants and half of calm participants, ensuring the data balance; in terms of the trials, we go from early stage to the late stage, from the first trial. Regardless the case when the number of participants is small (result would be unstable), the trend is that, longer trials lead to better recovery performance (higher correlation), and with the growth of participants number, the performance converges to a more stable status.

(h) Alternative model

As for the 2nd model, a new parameter A is introduced:

$$V_i^{(t+1)} = A \times V_i^{(t)} + \alpha \times (o^{(t)} - V_i^{(t)})$$

Since A is the coefficient of historical experience, from the view of reinforcement learning, without implementing and only read from the formula, it could be viewed as γ, the discount factor, while in the biological explanation, it could be regarded as a rate of forgetting. Too early experience will be ignored. By saying this, I personally assume that its value should satisfy $0 < A < 1$.

Now, after implementing the simulation and NLL function, we fit the new model with initial parameter setting $\alpha = 0.4$, $\beta = 5$, $A = 0.5$, as the result is shown in the left subplot of **figure 5**.

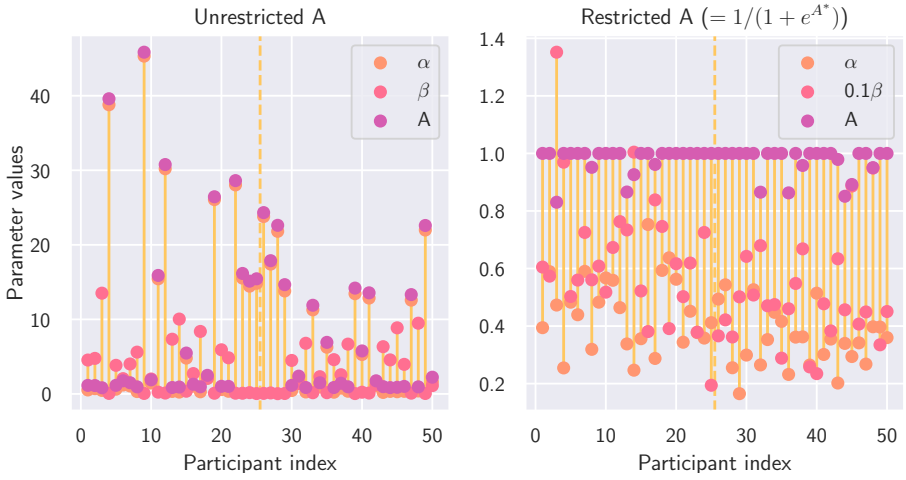

Figure 5: 2nd model fitting.

From the fitting result, it is really nonsensical that both A and α grows so large (sometimes even overflow during exponential computation, and one way to avoid these issues is by constraining parameter values to always give finite choice probabilities and log-likelihoods at the boundaries (Wilson & Collins, 2019)). Following the previous measure, we try to calculate the Pearson's correlation coefficient between α and A. The result is **0.29**, indicating a weak correlation between these two parameters. However, in common hypotheses about model parameters, they should be independent or uncorrelated, so this is an indication of parameters' trading off against each other, and reparameterizing may work (not implemented in this assignment). Then, consider this as a potential local minimum case, we try some other initial parameters, but still obtain the similar result. Furthermore, from the view of mathematics, we try to reconstruct the update function to find the intrinsic relationship between α and A:

$$V_i^{(t+1)} = A(V_i^{(t)} + \frac{\alpha}{A} \times (o^{(t)} - V_i^{(t)}))$$

In this case, by regarding the $\frac{\alpha}{A}$ term as original α, the 2nd model is just the same as the original 1st model, with an additional multiplier A. This indicates that A would not break the proportional relationship between $V(A)$ and $V(B)$, but their absolute value will be affected, leading the change in softmax function ($\frac{e^{-\beta \times V_A^{(t)}}}{e^{-\beta \times V_A^{(t)}} + e^{-\beta \times V_B^{(t)}}} = \frac{1}{1+e^{-\beta \times (V_B^{(t)} - V_A^{(t)})}}$). Now we know that such nonsensical case is just a numerical transformation, leading to slightly better NLL in optimization, but meaningless in interpretation. To avoid this, we need to constrain the value to satisfy the range of $0 < A < 1$. One way to do this is to define a new parameter A^* and let $A = \frac{1}{1+e^{A^*}}$. Now A^* is an unconstrained variable with arbitrary values, while A is automaticly restricted between 0 and 1. Re-fit the new "restricted" model, the result is shown in the right side subplot of **figure 5**.

Now for most cases, the parameter A is restricted to 1 (indicating no effect from A), while for some cases, the A indeed show an effect of "forgetting". Note that there is a little different from native "forget" of historical experience. Here, in order to follow the convention in reinforcement learning, we choose to set a boundary of 0 to 1 on the A, so that it will definitely decrease the $V(\cdot)$ or do nothing (when $A = 1$), which could be regarded as "optimistic attitude" based on previous experience, leading to a lower $V(\cdot)$. However, in another consideration, we could also set a symmetrical boundary, like $0 < A < 2$, so that the A could also be slightly larger than 1, leading to a pessimistic impact based on experience, leading to an "overestimated" $V(\cdot)$. These two boundary settings are similar, so we just implement the first one, and only give a brief discussion about the second one, due to the limited pages.

(k) Discussion and extra model

Before further model analysis in the following sections, considering the completeness, we first introduce the 3rd model, so that the following sections could include the comparison among all the 3 types of models, and the question "based on the model recovery, which model do you think generated the data used in this assignment" would also be discussed in the final section. Similarly with the previous models' structure, but here two new learning rates are introduced, one for neutral outcomes ($o^t = 0$) α^+, and one for punishments α^-:

$$V_i^{(t+1)} = V_i^{(t)} + ((1 - o^t)\alpha^+ + o^t\alpha^-)(o^{(t)} - V_i^{(t)})$$

With initial parameter setting $\alpha^+ = 0.25$, $\alpha^- = 0.35$ and $\beta = 6$, we fit the 3rd model as before, shown in **figure 7**, obtaining the corresponding statistics in terms of 2 groups, as in **table 4**.

The corresponding degrees of freedom is **47**. Note that, the t-stat or p-value indicates the difference of two fitted parameters between 2 groups (anxious and calm). So apparently, there is no statistically difference between the α^+ in each

	mean(1)	mean(2)	t-stat	p-value
α^+	0.29	0.23	0.98	0.33
α^-	0.58	0.47	2.84	0.01
β	14.8	8.18	1.72	0.09

Table 4: The t-statistic and p-value of α^+, α^- and β between two different groups (anxious and calm) in 3rd model.

group, meaning that the two groups show similar pattern when deal with neutral outcomes. However, as for the parameter α^-, the p-value tells us the difference exists at 1% significant level, indicating the different behaviour mechanism between high anxious participants and rather calm participants when they suffer a aversive sound or "punishment". Then for β, we can only say that it is statistically different between the two groups at a 10% level, which is not very significant based on the convention in statistics, or we can regard it as a slight difference in terms of decision process between the two groups. Overall, the α^-, which is about the feedback or respond mechanism in learning punishment or negative experience, is the main difference that leads to the non-homogeneousness between two groups. So in the anxiety therapy, maybe we should try to avoid and decrease the anxious person' negative attitude to affairs.

(i) Model comparison

To evaluate the goodness of fit among 3 different models, we introduce the AIC and BIC here:

$$AIC = 2 \times NLL + 2 \times p$$
$$BIC = 2 \times NLL + p \times \log(n)$$

In which the NLL is the negative log likelihood, p is the number of parameters, and n is the number of trials. In detail, for each participant, we will compute AIC and BIC scores for each model based on associated NLL (see **figure 8**). Then, by summing up the participant scores for each model, we could get a single score for each model, evaluating the overall performance across all the participants. After such computation, the results for 3 models are shown in **table 5**.

	Model 1	Model 2	Model 3
AIC	6070.43	6123.36	**6043.09**
BIC	**6377.95**	6584.64	6504.37

Table 5: The AIC and BIC of 3 different models.

Under AIC, the 3rd model is the best; while under BIC, the 1st model is the best. If only one model should be chosen, personally, I would consider the 1st model, with simpler model structure. Although it does not win in the AIC comparison, the AIC of 1st model is really close to the AIC of the 3rd model. In this case, 1st model might be the best one.

(j) Model recovery and confusion matrix

To check the reliability of model comparison under model recovery simulations, in this section, first, we simulate 500 "participants" from each model, separated into group 1 of 250 high anxious ones and group 2 of 250 calm ones (by this separation, during the simulation, we will input different mean and standard deviation to different groups, based on the fitted results of the 50 true samples from previous sections).As for the nonsensical value check, it is the same as before that we set the sensible range as 95% confidence interval in normal distribution ($mean \pm 1.96 \times std$). Since there are 3 models, we will totally simulate 1500 serials, each contains 160 trials. Then, for each trial, we fit it with 3 different models, and compare the AIC and BIC respectively, finding the best one and count. Finnaly, we could obtain the confusion matrix in percentage, as **table 6**.

Sim/Fit	1	2	3
1	**0.66**	0.10	0.24
2	**0.51**	0.20	0.29
3	0.43	0.12	**0.45**

AIC

	1	2	3
1	**0.92**	0.01	0.06
2	**0.64**	0.16	0.20
3	**0.74**	0.04	0.22

BIC

Table 6: Confusion matrix about the performance of 3 different models under AIC and BIC.

Note that the row of confusion matrix is the "correct" model which is used in the simulation, while the column model is the one to fit the simulated data. Under AIC, the maximum of each row is located on the diagonal, though some of the value is not close enough to 1. This indicates that in most cases, the data simulated by certain model will be best fitted also

by this model. While in the BIC case, it shows us that no matter the data is simulated by which model, there is always a higher probability that the 1st model will reach the best performance among all 3 models.

Finally, back to the left question in (k), that based on the confusion matrix under BIC, I consider that the 1st model may be the one that generate the data used in this assignment (notice the fact that even in the simulation part, we still use the estimated parameter mean and variance from the original given data, and this ensure a similar pattern between the given data and the simulated data. Futhermore, the proportion relationship of two groups (1:1) is also maintained during the data generative process).

Appendix

Supplemental figures

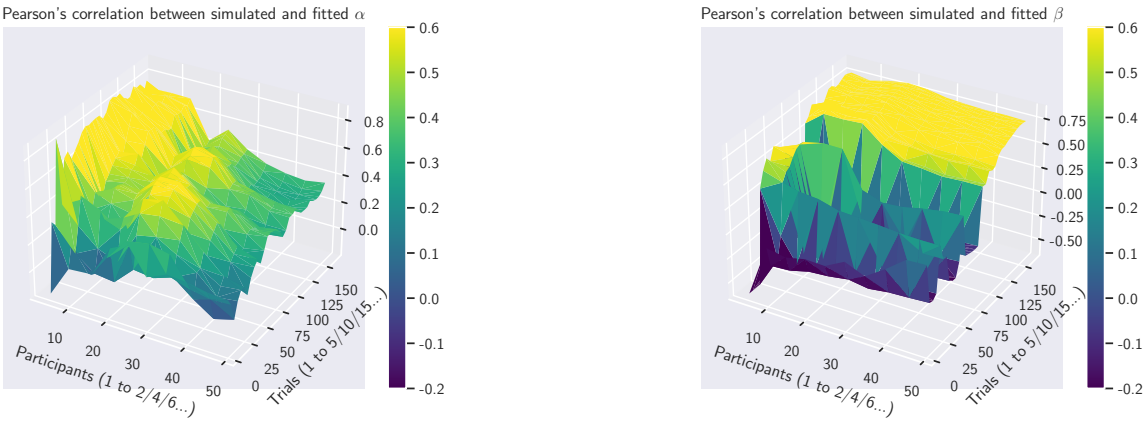

Figure 6: Pearson's correlation between simulated α/β and fitted α/β under different num of trials and simulated data sets.

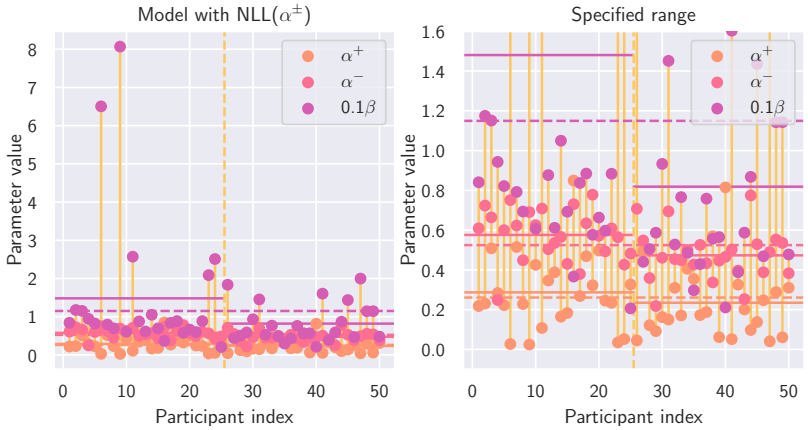

Figure 7: 3rd model fitting.

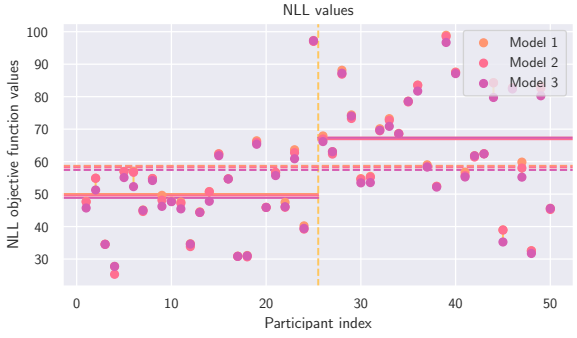

Figure 8: NLL for 3 different models in each data set.

Reference

Wilson, R. C., & Collins, A. G. (2019). Ten simple rules for the computational modeling of behavioral data. Elife, 8, e49547.

Python code: section (a)

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import copy
from scipy.optimize import minimize
from scipy import stats
import random

sns.set()
mpl.use('pgf') # for latex

# (a): Exploring the data (6)

# read .csv
score = pd.read_csv("stai_scores.csv", header = None)[0]
choices = pd.read_csv("inst_choices.csv", header = None)
outcomes = pd.read_csv("inst_outcomes.csv", header = None)

# statistics
score_mean = np.mean(score)
score_std = np.std(score)
score_median = np.median(score)

# cutoff count
cutoff_count = sum(score <= 43)
cutoff_index = [i for i in range(len(score)) if score[i] <= 43]
cutoff_count_unfit = sum(score[25:50] > 43)

# choices & outcomes count
choices_count = []
for i in range(50):
    choices_count.append(sum(choices.loc[i] == 1))
choices_count_sub1 = choices_count[0] / 160
average_choices_percentage = np.mean(choices_count) / 160
random_aversive_sound = 160 * 0.25 * (0.6 + 0.8 + 0.6 + 0.65)

# output
print("Mean:", score_mean)
print("Standard deviation:", round(score_std, 2))
print("Median:", score_median)
print("Count of healthy controls (STAI<=43):", cutoff_count)
print("Indices of the control subjects:", cutoff_index)
print("Count of unfitted healthy controls:", cutoff_count_unfit)
print("Count of choosing A:", choices_count)
print("Average percentage of choosing A (subject 1):", round(choices_count_sub1, 2))
print("Average count of choosing A:", round(average_choices_percentage, 2))
print("Random strategy expected count of aversive sound:", round(random_aversive_sound))
```

Python code: section (b)

```
# (b): Simulations (7)

# update strategy 1
def update_1(V, action, outcome, alpha):
```

```
    V[action-1] = V[action-1] + float(alpha) * (outcome - V[action-1])
    return V

# softmax decision
def softmax_decision(V, beta):
    p_action_A = 1 / (1 + np.exp(-beta * (V[1] - V[0])))
    return p_action_A

# simulation
def simulation(V0, n, update, decision, alpha, beta, sample = False):
    p_outcome_A1 = [0.6, 0.8, 0.6, 0.65]
    V_AB = np.zeros([161, 2])
    aver_stim = 0
    if sample == True:
        outcomes_sample = np.zeros(160)
        choices_sample = np.zeros(160)
    for s in range(n):
        V = copy.deepcopy(V0)
        V_AB[0,:] = V_AB[0,:] + np.array(V)
        for k in range(4):
            for i in range(40):
                # action
                if decision(V, beta) >= np.random.rand(1):
                    action = 1
                    p_outcome_1 = p_outcome_A1[k]
                else:
                    action = 2
                    p_outcome_1 = 1 - p_outcome_A1[k]
                # outcome
                if p_outcome_1 >= np.random.rand(1):
                    outcome = 1
                    aver_stim = aver_stim + 1
                else:
                    outcome = 0
                # update
                V = update(V, action, outcome, alpha)
                V_AB[k*40+i+1, :] = V_AB[k*40+i+1, :] + np.array(V)
                # record
                if sample == True:
                    choices_sample[k*40+i] = int(action)
                    outcomes_sample[k*40+i] = int(outcome)
    V_AB = V_AB / n
    aver_stim = aver_stim / n
    if sample == True:
        return choices_sample, outcomes_sample
    return V_AB, aver_stim

# config
V0 = [0.5, 0.5]
n = 10000
update = update_1
decision = softmax_decision
alpha = 0.3
beta = 8

# implement
V_AB_1, aver_stim_1 = simulation(V0, n, update, decision, alpha, beta)

# plot
fig, ax =  plt.subplots(1, 1, sharex = True, figsize = (6,3.7))
label = ['V(A)', 'V(B)', 'V(A)-V(B)']
color = ['#FFC75F', '#FF9671', '#FF6F91']
ax.plot(np.arange(0, 161, 1), V_AB_1[:, 0], label = label[0], color = color[0])
```

```
ax.plot(np.arange(0, 161, 1), V_AB_1[:, 1], label = label[1], color = color[1])
ax.plot(np.arange(0, 161, 1), V_AB_1[:, 0] - V_AB_1[:,1], label = label[2], \
        color = color[2])
ax.set_title('Simulation evolution (n=10000)')
ax.set_xlabel('Trials')
ax.set_ylabel('Average action values')
ax.legend(loc = 'upper right')
plt.show()

# check
print("Average number of aversive sounds:", aver_stim_1)

# image export
# plt.savefig('1.pgf', format='pgf')
```

Python code: section (c)

```
# (c): Exploring parameter settings (7)

# config
V0 = [0.5, 0.5]
n = 2
update = update_1
decision = softmax_decision
alpha = 0
beta0 = 0
intv = 0.05
sepn = int(1/intv) - 1

# implement
aver_stim = np.zeros([sepn,sepn])
for i in range(sepn):
    alpha = alpha + intv
    beta = beta0
    for j in range(sepn):
        beta = beta + 10 * intv
        _, aver_stim_ab = simulation(V0, n, update, decision, alpha, beta)
        aver_stim[i,j] = aver_stim_ab

# plot
fig = plt.figure(figsize = (6,4.7))
ax = fig.gca(projection = '3d')
aa = np.arange(intv, 1, intv)
bb = np.arange(10*intv, 10, 10*intv)
beta_axis, alpha_axis = np.meshgrid(bb, aa)
alpha_axis.resize(sepn*sepn)
beta_axis.resize(sepn*sepn)
aver_stim.resize(sepn*sepn)
plot_c = ax.plot_trisurf(beta_axis, alpha_axis, aver_stim, cmap = plt.cm.viridis, \
                         linewidth = 0)
fig.colorbar(plot_c)
ax.set_title('Average number of aversive stimuli (n=1000)')
ax.set_xlabel(r'$\beta$')
ax.set_ylabel(r'$\alpha$')
plt.show()

# image export
# plt.savefig('2.pgf', format='pgf')
```

Python code: section (d)

```
# (d): Likelihood function (6)

# negative log likelihood
def nll(theta, choices, outcomes, V0):
    length_num = len(choices)
    V = copy.deepcopy(V0)
    nll = np.zeros(length_num)
    for i in range(length_num):
        c = int(choices[i])
        c_alt = 1 + int(c==1)
        nll[i] = 1 / (1 + np.exp(-theta[1] * (V[c_alt-1] - V[c-1])))
        V[c-1] = V[c-1] + theta[0] * (int(outcomes[i]) - V[c-1])
    nll = -np.sum(np.log(nll))
    return nll

# config
V0 = [0.5, 0.5]
theta = [0.3, 8]

# implement
nll_1 = nll(theta, choices.loc[0], outcomes.loc[0], V0)
nll_2 = nll(theta, choices.loc[1], outcomes.loc[1], V0)
nll_10 = nll(theta, choices.loc[9], outcomes.loc[9], V0)

# output
print("NLL for 1st participant is:", round(nll_1,2))
print("NLL for 2nd participant is:", round(nll_2,2))
print("NLL for 10th participant is:", round(nll_10,2))
```

Python code: section (e)

```
# (e): Model fitting (9)

# optimization
V0 = [0.5, 0.5]
theta_opt = np.zeros([50,2])
nll_a_obj = np.zeros(50)
for i in range(50):
    nll_result = minimize(nll, [0.3, 8], args = (choices.loc[i], outcomes.loc[i], \
                                                 V0), method = "Nelder-Mead")
    theta_opt[i, :] = nll_result['x']
    nll_a_obj[i] = nll_result['fun']

# statistics
theta_mean = np.mean(theta_opt, axis = 0)
theta_var = np.var(theta_opt, axis = 0)
theta_cor = np.corrcoef(theta_opt[:, 0], theta_opt[:, 1])[0, 1]
theta_cor_1 = np.corrcoef(theta_opt[0:25, 0], theta_opt[0:25, 1])[0, 1]
theta_cor_2 = np.corrcoef(theta_opt[25:50, 0], theta_opt[25:50, 1])[0, 1]

# plot
fig, ax =  plt.subplots(1, 1, sharex = True, figsize = (6,3.7))
label = [r'$\alpha$', r'0.1$\beta$']
color = ['#FF9671', '#FF6F91']
# color = ['#FF6F91', '#D65DB1']
for i in range(50):
    plt.vlines(x = i+1, ymin = theta_opt[i, 0], ymax = 0.1*theta_opt[i, 1], \
               color = '#FFC75F', zorder = 1)
plt.axvline(x = 25.5, ymin = 0, ymax = 1, linestyle = "dashed", \
            color = '#FFC75F', zorder = 2)
plt.axhline(y = np.mean(theta_opt[:, 0]), xmin = 0, xmax = 1, \
            linestyle = "dashed", color = color[0], zorder = 2)
```

```
plt.axhline(y = 0.1*np.mean(theta_opt[:, 1]), xmin = 0, xmax = 1, \
            linestyle = "dashed", color = color[1], zorder = 2)
plt.axhline(y = np.mean(theta_opt[0:25, 0]), xmin = 0, xmax = 0.5, \
            color = color[0], zorder = 2)
plt.axhline(y = 0.1*np.mean(theta_opt[0:25, 1]), xmin = 0, xmax = 0.5, \
            color = color[1], zorder = 2)
plt.axhline(y = np.mean(theta_opt[25:50, 0]), xmin = 0.5, xmax = 1, \
            color = color[0], zorder = 2)
plt.axhline(y = 0.1*np.mean(theta_opt[25:50, 1]), xmin = 0.5, xmax = 1, \
            color = color[1], zorder = 2)
ax.scatter(np.arange(1, 51, 1), theta_opt[:, 0], label = label[0], \
           color = color[0], zorder = 3)
ax.scatter(np.arange(1, 51, 1), 0.1*theta_opt[:, 1], label = label[1], \
           color = color[1], zorder = 3)
ax.set_title('Parameter optimization of NLL by Nelder-Mead')
ax.set_xlabel('Participant index')
ax.set_ylabel('Parameter value')
ax.legend(loc = 'upper right')
plt.show()

# output
print("Mean of fitted parameter values [alpha, beta]:", \
      [round(i,2) for i in theta_mean])
print("Variance of fitted parameter values [alpha, beta]:", \
      [round(i,2) for i in theta_var])
print("Pearson's correlation coefficient (total):", round(theta_cor,2))
print("Pearson's correlation coefficient (group 1):", round(theta_cor_1,2))
print("Pearson's correlation coefficient (group 2):", round(theta_cor_2,2))

# image export
# plt.savefig('3.pgf', format='pgf')
```

Python code: section (f)

```
# (f): Group comparison (4)

# t-test
alpha_mean_1 = theta_opt[0:25, 0]
alpha_mean_2 = theta_opt[25:50, 0]
beta_mean_1 = theta_opt[0:25, 1]
beta_mean_2 = theta_opt[25:50, 1]
t_alpha = stats.ttest_ind(alpha_mean_1, alpha_mean_2)
t_beta = stats.ttest_ind(beta_mean_1, beta_mean_2)

# output
print("The mean of alpha in 2 groups:", round(np.mean(alpha_mean_1), 2), \
      round(np.mean(alpha_mean_2), 2))
print("The t-statistic of alpha:", round(t_alpha[0], 2))
print("The p-value of alpha:", round(t_alpha[1], 2))
print("The mean of beta in 2 groups:", round(np.mean(beta_mean_1), 2), \
      round(np.mean(beta_mean_2), 2))
print("The t-statistic of beta:", round(t_beta[0], 2))
print("The p-value of beta:", round(t_beta[1], 2))
print("Degrees of freedom:", 48) # (n1+n2-2)
```

Python code: section (g)

```
# (g): Parameter recovery (9)

# config
mu_alpha = [np.mean(alpha_mean_1), np.mean(alpha_mean_2)]
mu_beta = [np.mean(beta_mean_1), np.mean(beta_mean_2)]
```

```
sigma_alpha = 0.01
sigma_beta = 0.5
V0 = [0.5, 0.5]
n = 1
update = update_1
decision = softmax_decision
repeat_num = 5
participants_num = np.arange(4, 54, 5)
trials_num = np.arange(5, 165, 5)

# recovery
theta_cor_samples = np.zeros([repeat_num, 2])
choices_sample = np.zeros([50, 160])
outcomes_sample = np.zeros([50, 160])
theta_samples_fit = np.zeros([50, 2])
for r in range(repeat_num):
    # sample parameters
    theta_sample = np.zeros([50, 2])
    theta_sample_record = np.zeros([50, 2, 20])
    for k in range(2):
        for i in range(25):
            # ensure alpha and beta in mean+-1.96*std
            j_a = 0
            j_b = 0
            while theta_sample[k*25+i, 0] <= mu_alpha[k] - 1.96 * sigma_alpha \
            or theta_sample[k*25+i, 0] >= mu_alpha[k] + 1.96 * sigma_alpha:
                theta_sample[k*25+i, 0] = np.random.normal(mu_alpha[k], sigma_alpha)
                theta_sample_record[k*25+i, 0, j_a] = theta_sample[k*25+i, 0]
                j_a = j_a + 1
            while theta_sample[k*25+i, 1] <= mu_beta[k] - 1.96 * sigma_beta \
            or theta_sample[k*25+i, 1] >= mu_beta[k] + 1.96 * sigma_beta:
                theta_sample[k*25+i, 1] = np.random.normal(mu_beta[k], sigma_beta)
                theta_sample_record[k*25+i, 1, j_b] = theta_sample[k*25+i, 1]
                j_b = j_b + 1
    # simulation and fit by optimizing NLL and statistics
    for i in range(50):
        choices_sample[i, :], outcomes_sample[i, :] = \
        simulation(V0, n, update, decision, theta_sample[i, 0], \
                   theta_sample[i, 1], sample = True)
        theta_samples_fit[i, :] = \
        minimize(nll, [0.3, 8], args = (choices_sample[i, :], \
                                        outcomes_sample[i, :], V0), \
                 method = "Nelder-Mead")['x']
    theta_cor_samples[r, 0] = np.corrcoef(theta_samples_fit[:, 0], \
                                          theta_sample[:, 0])[0, 1]
    theta_cor_samples[r, 1] = np.corrcoef(theta_samples_fit[:, 1], \
                                          theta_sample[:, 1])[0, 1]

# explore
choices_sample = np.zeros([50, 160])
outcomes_sample = np.zeros([50, 160])
theta_cor_sample = np.zeros([len(participants_num), len(trials_num), 2])
theta_sample_fit = np.zeros([50, len(trials_num), 2])
theta_sample_copy = np.zeros([50, 2])
for i in range(25):
    theta_sample_copy[2*i, :] = theta_sample[i, :]
    theta_sample_copy[2*i+1, :] = theta_sample[25+i, :]
for i in range(50):
    if (i in participants_num) is True:
        i_index = participants_num.tolist().index(i)
    choices_sample[i, :], outcomes_sample[i, :] = \
    simulation(V0, n, update, decision, theta_sample_copy[i, 0], \
               theta_sample_copy[i, 1], sample = True)
```

```
    for j in range(len(trials_num)):
        theta_sample_fit[i, j, :] = \
        minimize(nll, [0.3, 8], args = (choices_sample[i, 0:trials_num[j]], \
                                        outcomes_sample[i, 0:trials_num[j]], V0), \
                 method = "Nelder-Mead")['x']
        if (i in participants_num) is True:
            theta_cor_sample[i_index, j, 0] = \
            np.corrcoef(theta_sample_fit[0:(i+1), j, 0], \
                        theta_sample_copy[0:(i+1), 0])[0, 1]
            theta_cor_sample[i_index, j, 1] = \
            np.corrcoef(theta_sample_fit[0:(i+1), j, 1], \
                        theta_sample_copy[0:(i+1), 1])[0, 1]

# plot
# fig, ax =  plt.subplots(1, 1, sharex = True, figsize = (6,3.7))
# label = [r'$Cor(\alpha)$', r'$Cor(\beta)$']
# color = ['#FF9671', '#FF6F91']
# for i in range(5):
#     plt.vlines(x = i+1, ymin = theta_cor_sample[i, -1, -1, 0], \
# ymax = theta_cor_sample[i, -1, -1, 1], color = '#FFC75F', zorder = 1)
# plt.axhline(y = np.mean(theta_cor_sample[:, -1, -1, 0]), xmin = 0, xmax = 6, \
# linestyle = "dashed", color = color[0], zorder = 2)
# plt.axhline(y = np.mean(theta_cor_sample[:, -1, -1, 1]), xmin = 0, xmax = 6, \
# linestyle = "dashed", color = color[1], zorder = 2)
# ax.scatter(np.arange(1, repeat_num+1, 1), theta_cor_sample[:, -1, -1, 0], \
# label = label[0], color = color[0], zorder = 3)
# ax.scatter(np.arange(1, repeat_num+1, 1), theta_cor_sample[:, -1, -1, 1], \
# label = label[1], color = color[1], zorder = 3)
# ax.set_title('Pearson's correlation between simulated and re-fitted parameters')
# ax.set_xlabel('Trials')
# ax.set_ylabel('Pearson's correlation')
# ax.legend(loc = 'upper right')
# plt.show()

# plot
fig, ax =  plt.subplots(1, 2, sharey = True, figsize = (8,3.7))
label = [r'$\alpha$', r'$0.1\beta$']
color = ['#FF9671', '#FF6F91']
#for i in range(5):
#    ax[0].vlines(x = i+1, ymin = theta_cor_sample[i, -1, -1, 0], ymax = \
# theta_cor_sample[i, -1, -1, 1], color = '#FFC75F', zorder = 1)
for i in range(2):
    ax[0].axhline(y = mu_alpha[i], xmin = 0.5*i, xmax = 0.5*(i+1), \
                  color = color[0], zorder = 2)
    ax[0].axhline(y = 0.1*mu_beta[i], xmin = 0.5*i, xmax = 0.5*(i+1), \
                  color = color[1], zorder = 2)
ax[0].axhline(y = np.mean(mu_alpha[:]), xmin = 0, xmax = 1, \
              linestyle = "dashed", color = color[0], zorder = 2)
ax[0].axhline(y = 0.1*np.mean(mu_beta[:]), xmin = 0, xmax = 1, \
              linestyle = "dashed", color = color[1], zorder = 2)
ax[0].scatter(np.arange(1, 51, 1), theta_sample[:, 0], label = label[0], \
              color = color[0], zorder = 3)
ax[0].scatter(np.arange(1, 51, 1), 0.1 * theta_sample[:, 1], label = label[1], \
              color = color[1], zorder = 3)
ax[0].axvline(x = 25.5, ymin = 0, ymax = 1, linestyle = "dashed", \
              color = '#FFC75F', zorder = 2)
for i in range (25):
    for k in range(2):
        for j in range(20):
            if (theta_sample_record[25*k+i, 0, j] <= mu_alpha[k] - 1.96 * \
                sigma_alpha or theta_sample_record[25*k+i, 0, j] >= mu_alpha[k] + \
                1.96 * sigma_alpha) and theta_sample_record[25*k+i, 0, j] != 0:
                ax[0].scatter(25*k+i+1, theta_sample_record[25*k+i, 0, j], \
```

```
                                color = '#D65DB1', zorder = 3)
            if (theta_sample_record[25*k+i, 1, j] <= mu_beta[k] - 1.96 * sigma_beta \
                or theta_sample_record[25*k+i, 1, j] >= mu_beta[k] + 1.96 * sigma_beta) \
            and theta_sample_record[25*k+i, 1, j] != 0:
                ax[0].scatter(25*k+i+1, 0.1*theta_sample_record[25*k+i, 1, j], \
                                color = '#845EC2', zorder = 3)
ax[0].set_title('Simulated params and outliers')
ax[0].set_xlabel('Participant index')
ax[0].set_ylabel('Simulated parameters')
ax[0].legend(loc = 'upper right')
ax[1].scatter(theta_samples_fit[:, 0], theta_sample[:, 0], label = label[0], \
              color = color[0], zorder = 3)
ax[1].scatter(0.1*theta_samples_fit[:, 1], 0.1*theta_sample[:, 1], label = label[1], \
              color = color[1], zorder = 3)
alpha_min = np.min([np.min(theta_sample[:, 0]), np.min(theta_samples_fit[:, 0])])
beta_min = 0.1 * np.min([np.min(theta_sample[:, 1]), np.min(theta_samples_fit[:, 1])])
alpha_max = np.max([np.max(theta_sample[:, 0]), np.max(theta_samples_fit[:, 0])])
beta_max = 0.1 * np.max([np.max(theta_sample[:, 1]), np.max(theta_samples_fit[:, 1])])
lim_min = np.min([alpha_min, beta_min])
lim_max = np.max([alpha_max, beta_max])
lim_min = np.min([lim_min, 0.1*np.min(theta_sample_record[theta_sample_record>0])]) - 0.05
lim_max = np.max([lim_max, 0.1*np.max(theta_sample_record[theta_sample_record>0])]) + 0.05
x_grid = np.arange(lim_min, lim_max, 0.01)
ax[1].set_ylim(lim_min, lim_max)
ax[1].set_xlim(lim_min, lim_max)
ax[1].plot(x_grid, x_grid, linestyle = "dashed", color = '#FFC75F', zorder = 2)
ax[1].set_title('Cor between simulated and fitted params')
# ax[1].set_ylabel('Simulated parameters')
ax[1].set_xlabel('Fitted parameters')
ax[1].legend(loc = 'upper right')
plt.show()

# image export
# plt.savefig('f4.pgf', format='pgf')

# 3d plots
trials_axis, participants_axis = np.meshgrid(trials_num, participants_num)
participants_axis.resize(len(participants_num)*len(trials_num))
trials_axis.resize(len(participants_num)*len(trials_num))
theta_cor_sample_mean = theta_cor_sample
theta_cor_sample_mean_alpha = \
theta_cor_sample_mean[:, :, 0].reshape(len(participants_num)*len(trials_num))
theta_cor_sample_mean_beta = \
theta_cor_sample_mean[:, :, 1].reshape(len(participants_num)*len(trials_num))

# 3d plot 1
fig = plt.figure(figsize = (6,4.7))
ax = fig.gca(projection = '3d')
plot_g_1 = ax.plot_trisurf(participants_axis, trials_axis, theta_cor_sample_mean_alpha, \
                           cmap = plt.cm.viridis, linewidth = 0, vmin=-0.2, vmax=0.6)
fig.colorbar(plot_g_1)
ax.set_title(r'Pearson's correlation between simulated and fitted $\alpha$')
ax.set_ylabel('Trials (1 to 5/10/15...)')
ax.set_xlabel('Participants (1 to 2/4/6...)')
plt.show()

# image export
# plt.savefig('s1.pgf', format='pgf')

# 3d plot 2
fig = plt.figure(figsize = (6,4.7))
ax = fig.gca(projection = '3d')
plot_g_2 = ax.plot_trisurf(participants_axis, trials_axis, theta_cor_sample_mean_beta, \
```

```
                          cmap = plt.cm.viridis, linewidth = 0, vmin=-0.2, vmax=0.6)
fig.colorbar(plot_g_2)
ax.set_title(r'Pearson's correlation between simulated and fitted $\beta$')
ax.set_ylabel('Trials (1 to 5/10/15...)')
ax.set_xlabel('Participants (1 to 2/4/6...)')
plt.show()

# image export
# plt.savefig('s2.pgf', format='pgf')

# output
print("Pearson's correlation of alpha:", [round(i,2) for i in theta_cor_samples[:, 0]])
print("Pearson's correlation of beta:", [round(i,2) for i in theta_cor_samples[:, 1]])
```

Python code: section (h)

```
# (h): Alternative model (9)

# update strategy 2
def update_2(V, action, outcome, alpha):
    V[action-1] = alpha[0] * V[action-1] + alpha[1] * (outcome - V[action-1])
    return V

# simulation() remains the same as before by setting the config:
# update = update_2
# alpha = [alpha, A]

# negative log likelihood
def nll_A(theta, choices, outcomes, V0):
    # theta = [alpha, beta, A]
    length_num = len(choices)
    V = copy.deepcopy(V0)
    nll_A = np.zeros(length_num)
    for i in range(length_num):
        c = int(choices[i])
        c_alt = 1 + int(c==1)
        nll_A[i] = 1 / (1 + np.exp(-theta[1] * (V[c_alt-1] - V[c-1])))
        V[c-1] = theta[2] * V[c-1] + theta[0] * (int(outcomes[i]) - V[c-1])
    nll_A = -np.sum(np.log(nll_A))
    return nll_A

# optimization
V0 = [0.5, 0.5]
theta_opt_A = np.zeros([50,3])
nll_A_obj = np.zeros(50)
for i in range(50):
    nll_A_result = minimize(nll_A, [0.4, 5, 0.5], \
                            args = (choices.loc[i], outcomes.loc[i], V0), \
                            method = "Nelder-Mead")
    theta_opt_A[i, :] = nll_A_result['x']
    nll_A_obj[i] = nll_A_result['fun']

# negative log likelihood
def nll_A1(theta, choices, outcomes, V0):
    # theta = [alpha, beta, np.log(np.divide(1, A) - 1)]
    length_num = len(choices)
    V = copy.deepcopy(V0)
    nll_A1 = np.zeros(length_num)
    for i in range(length_num):
        c = int(choices[i])
        c_alt = 1 + int(c==1)
        nll_A1[i] = 1 / (1 + np.exp(-theta[1] * (V[c_alt-1] - V[c-1])))
```

```
        V[c-1] = V[c-1] / (1 + np.exp(theta[2])) + theta[0] * \
        (int(outcomes[i]) - V[c-1])
    nll_A1 = -np.sum(np.log(nll_A1))
    return nll_A1

# optimization
V0 = [0.5, 0.5]
theta_opt_A1 = np.zeros([50,3])
for i in range(50):
    theta_opt_A1[i, :] = minimize(nll_A1, [0.4, 5, np.log(np.divide(1, 0.5) - 1)], \
                                  args = (choices.loc[i], outcomes.loc[i], V0), \
                                  method = "Nelder-Mead")['x']
theta_opt_A1[:, 2] = 1 / (1 + np.exp(theta_opt_A1[:, 2]))

# statistics
theta_mean_A = np.mean(theta_opt_A, axis = 0)
theta_var_A = np.var(theta_opt_A, axis = 0)
theta_mean_A1 = np.mean(theta_opt_A1, axis = 0)
theta_var_A1 = np.var(theta_opt_A1, axis = 0)
theta_cor_A1 = np.corrcoef(theta_opt_A1[:, 0], theta_opt_A1[:, 2])[0, 1]

# plot
theta_opt_A1[:, 1] = 0.1 * theta_opt_A1[:, 1] # warning
fig, ax =  plt.subplots(1, 2, figsize = (8,3.7))
label_A = [r'$\alpha$', r'$\beta$', 'A']
label_A1 = [r'$\alpha$', r'0.1$\beta$', 'A']
color = ['#FF9671', '#FF6F91', '#D65DB1']
# color = ['#FF6F91', '#D65DB1']
for i in range(50):
    ax[0].vlines(x = i+1, ymin = np.min(theta_opt_A[i, :]), ymax = \
                 np.max(theta_opt_A[i, :]), color = '#FFC75F', zorder = 1)
    ax[1].vlines(x = i+1, ymin = np.min(theta_opt_A1[i, :]), ymax = \
                 np.max(theta_opt_A1[i, :]), color = '#FFC75F', zorder = 1)
for i in range(2):
    ax[i].axvline(x = 25.5, ymin = 0, ymax = 1, linestyle = "dashed", \
                  color = '#FFC75F', zorder = 2)
for i in range(3):
    ax[0].scatter(np.arange(1, 51, 1), theta_opt_A[:, i], label = label_A[i], \
                  color = color[i], zorder = 3)
    ax[1].scatter(np.arange(1, 51, 1), theta_opt_A1[:, i], label = label_A1[i], \
                  color = color[i], zorder = 3)
ax[0].set_xlabel('Participant index')
ax[1].set_xlabel('Participant index')
ax[0].set_ylabel('Parameter value')
ax[0].legend(loc = 'upper right')
ax[1].legend(loc = 'upper right')
ax[0].set_title('Unrestricted A')
ax[1].set_title('Restricted A ($=1/(1+e^{A^{*}})$)')
# plt.suptitle('Parameter optimization of NLL by Nelder-Mead')
theta_opt_A1[:, 1] = 10 * theta_opt_A1[:, 1] # warning
plt.show()

# output
print("Mean of fitted parameter values [alpha, beta, A] by NLL(A):", \
      [round(i,2) for i in theta_mean_A])
print("Mean of fitted parameter values [alpha, beta, A] by restricted NLL(A):", \
      [round(i,2) for i in theta_mean_A1])
print("Variance of fitted parameter values [alpha, beta, A] by NLL(A):", \
      [round(i,2) for i in theta_var_A])
print("Variance of fitted parameter values [alpha, beta, A] by restricted NLL(A):", \
      [round(i,2) for i in theta_var_A1])
print("Pearson's correlation coefficient (total) between alpha and A:", \
      round(theta_cor_A1,2))
```

```
# image export
# plt.savefig('f5.pgf', format='pgf')
```

Python code: section (k)

```
# (k): Discussion and extra model (15)

# update strategy 3
def update_3(V, action, outcome, alpha):
    V[action-1] = V[action-1] + ((1 - outcome) * alpha[0] + outcome * alpha[1]) \
    * (outcome - V[action-1])
    return V

# simulation() remains the same as before by setting the config:
# update = update_3
# alpha = [alpha_pos, alpha_neg]

# negative log likelihood
def nll_aa(theta, choices, outcomes, V0):
    # theta = [alpha_pos, alpha_neg, beta]
    length_num = len(choices)
    V = copy.deepcopy(V0)
    nll_aa = np.zeros(length_num)
    for i in range(length_num):
        c = int(choices[i])
        c_alt = 1 + int(c==1)
        nll_aa[i] = 1 / (1 + np.exp(-theta[2] * (V[c_alt-1] - V[c-1])))
        V[c-1] = V[c-1] + ((1 - int(outcomes[i])) * theta[0] + int(outcomes[i]) \
                           * theta[1]) * (int(outcomes[i]) - V[c-1])
    nll_aa = -np.sum(np.log(nll_aa))
    return nll_aa

# optimization
V0 = [0.5, 0.5]
theta_opt_aa = np.zeros([50,3])
nll_aa_obj = np.zeros(50)
for i in range(50):
    nll_aa_result = minimize(nll_aa, [0.25, 0.35, 6], \
                             args = (choices.loc[i], outcomes.loc[i], V0), \
                             method = "Nelder-Mead")
    theta_opt_aa[i, :] = nll_aa_result['x']
    nll_aa_obj[i] = nll_aa_result['fun']

# statistics
theta_mean_aa = np.mean(theta_opt_aa, axis = 0)
theta_var_aa = np.var(theta_opt_aa, axis = 0)
a_pos_mean_1 = theta_opt_aa[0:25, 0]
a_pos_mean_2 = theta_opt_aa[25:50, 0]
a_neg_mean_1 = theta_opt_aa[0:25, 1]
a_neg_mean_2 = theta_opt_aa[25:50, 1]
beta_aa_mean_1 = theta_opt_aa[0:25, 2]
beta_aa_mean_2 = theta_opt_aa[25:50, 2]
t_a_pos = stats.ttest_ind(a_pos_mean_1, a_pos_mean_2)
t_a_neg = stats.ttest_ind(a_neg_mean_1, a_neg_mean_2)
t_beta_aa = stats.ttest_ind(beta_aa_mean_1, beta_aa_mean_2)

# plot
theta_opt_aa[:, 2] = 0.1 * theta_opt_aa[:, 2] # warning
fig, ax =  plt.subplots(1, 2, figsize = (8,3.7))
label_aa = [r'$\alpha^+$', r'$\alpha^-$', r'0.1$\beta$']
color = ['#FF9671', '#FF6F91', '#D65DB1']
```

```
for i in range(50):
    for j in range(2):
        ax[j].vlines(x = i+1, ymin = np.min(theta_opt_aa[i, :]), \
                     ymax = np.max(theta_opt_aa[i, :]), color = '#FFC75F', zorder = 1)
for i in range(3):
    for j in range(2):
        ax[j].scatter(np.arange(1, 51, 1), theta_opt_aa[:, i], \
                      label = label_aa[i], color = color[i], zorder = 3)
        ax[j].axhline(y = np.mean(theta_opt_aa[:, i]), xmin = 0, xmax = 1, \
                      linestyle = "dashed", color = color[i], zorder = 2)
        for k in range(2):
            ax[j].axhline(y = np.mean(theta_opt_aa[(25*k):(25*(k+1)), i]), \
                          xmin = 0.5*k, xmax = 0.5*(k+1), color = color[i], zorder = 2)
for i in range(2):
    ax[i].axvline(x = 25.5, ymin = 0, ymax = 1, linestyle = "dashed", \
                  color = '#FFC75F', zorder = 2)
    ax[i].set_xlabel('Participant index')
    ax[i].set_ylabel('Parameter value')
    ax[i].legend(loc = 'upper right')
    ax[i].set_xlabel('Participant index')
ax[0].set_title(r'Model with NLL($\alpha^{\pm}$)')
ax[1].set_title('Specified range')
ax[1].set_ylim(-0.1, 1.6)
theta_opt_aa[:, 2] = 10 * theta_opt_aa[:, 2] # warning
plt.show()

# output
print("Mean of fitted parameter values [alpha+, alpha-, beta] by NLL(aa):", \
      [round(i, 2) for i in theta_mean_aa])
print("Variance of fitted parameter values [alpha+, alpha-, beta] by NLL(aa):", \
      [round(i, 2) for i in theta_var_aa])
print("Mean of fitted parameter values [alpha+, alpha-, beta] in group 1:", \
      round(np.mean(a_pos_mean_1), 2), round(np.mean(a_neg_mean_1), 2), \
      round(np.mean(beta_aa_mean_1), 2))
print("Mean of fitted parameter values [alpha+, alpha-, beta] in group 2:", \
      round(np.mean(a_pos_mean_2), 2), round(np.mean(a_neg_mean_2), 2), \
      round(np.mean(beta_aa_mean_2), 2))
print("The t-statistic of alpha+:", round(t_a_pos[0], 2))
print("The p-value of alpha+:", round(t_a_pos[1], 2))
print("The t-statistic of alpha-:", round(t_a_neg[0], 2))
print("The p-value of alpha-:", round(t_a_neg[1], 2))
print("The t-statistic of beta:", round(t_beta_aa[0], 2))
print("The p-value of beta:", round(t_beta_aa[1], 2))

# image export
# plt.savefig('f6.pgf', format='pgf')
```

Python code: section (i)

```
# (i): Model comparison (8)

# plot
fig, ax =  plt.subplots(1, 1, sharex = True, figsize = (7,3.7))
label = ['Model 1', 'Model 2', 'Model 3']
color = ['#FF9671', '#FF6F91', '#D65DB1']
ax.scatter(np.arange(1, 51, 1), nll_a_obj, label = label[0], \
           color = color[0], zorder = 3)
ax.scatter(np.arange(1, 51, 1), nll_A_obj, label = label[1], \
           color = color[1], zorder = 3)
ax.scatter(np.arange(1, 51, 1), nll_aa_obj, label = label[2], \
           color = color[2], zorder = 3)
for i in range(50):
```

```
    plt.vlines(x = i+1, ymin = np.min([nll_a_obj[i], nll_A_obj[i], nll_aa_obj[i]]), \
               ymax = np.max([nll_a_obj[i], nll_A_obj[i], nll_aa_obj[i]]), \
               color = '#FFC75F', zorder = 1)
plt.axvline(x = 25.5, ymin = 0, ymax = 1, linestyle = "dashed", \
            color = '#FFC75F', zorder = 2)
plt.axhline(y = np.mean(nll_a_obj), xmin = 0, xmax = 1, \
            linestyle = "dashed", color = color[0], zorder = 2)
plt.axhline(y = np.mean(nll_A_obj), xmin = 0, xmax = 1, \
            linestyle = "dashed", color = color[1], zorder = 2)
plt.axhline(y = np.mean(nll_aa_obj), xmin = 0, xmax = 1, \
            linestyle = "dashed", color = color[2], zorder = 2)
plt.axhline(y = np.mean(nll_a_obj[0:25]), xmin = 0, xmax = 0.5, \
            color = color[0], zorder = 2)
plt.axhline(y = np.mean(nll_A_obj[0:25]), xmin = 0, xmax = 0.5, \
            color = color[1], zorder = 2)
plt.axhline(y = np.mean(nll_aa_obj[0:25]), xmin = 0, xmax = 0.5, \
            color = color[2], zorder = 2)
plt.axhline(y = np.mean(nll_a_obj[25:50]), xmin = 0.5, xmax = 1, \
            color = color[0], zorder = 2)
plt.axhline(y = np.mean(nll_A_obj[25:50]), xmin = 0.5, xmax = 1, \
            color = color[1], zorder = 2)
plt.axhline(y = np.mean(nll_a_obj[25:50]), xmin = 0.5, xmax = 1, \
            color = color[2], zorder = 2)
ax.set_title('NLL values')
ax.set_xlabel('Participant index')
ax.set_ylabel('NLL objective function values')
ax.legend(loc = 'upper right')
plt.show()

# AIC and BIC
aic_nll_a = np.sum(2 * nll_a_obj + 2 * 2)
aic_nll_A = np.sum(2 * nll_A_obj + 2 * 3)
aic_nll_aa = np.sum(2 * nll_aa_obj + 2 * 3)
aic_nll_a_1 = np.sum(2 * nll_a_obj[0:25] + 2 * 2)
aic_nll_A_1 = np.sum(2 * nll_A_obj[0:25] + 2 * 3)
aic_nll_aa_1 = np.sum(2 * nll_aa_obj[0:25] + 2 * 3)
aic_nll_a_2 = np.sum(2 * nll_a_obj[25:50] + 2 * 2)
aic_nll_A_2 = np.sum(2 * nll_A_obj[25:50] + 2 * 3)
aic_nll_aa_2 = np.sum(2 * nll_aa_obj[25:50] + 2 * 3)
bic_nll_a = np.sum(2 * nll_a_obj + 2 * np.log(160))
bic_nll_A = np.sum(2 * nll_A_obj + 3 * np.log(160))
bic_nll_aa = np.sum(2 * nll_aa_obj + 3 * np.log(160))
bic_nll_a_1 = np.sum(2 * nll_a_obj[0:25] + 2 * np.log(160))
bic_nll_A_1 = np.sum(2 * nll_A_obj[0:25] + 3 * np.log(160))
bic_nll_aa_1 = np.sum(2 * nll_aa_obj[0:25] + 3 * np.log(160))
bic_nll_a_2 = np.sum(2 * nll_a_obj[25:50] + 2 * np.log(160))
bic_nll_A_2 = np.sum(2 * nll_A_obj[25:50] + 3 * np.log(160))
bic_nll_aa_2 = np.sum(2 * nll_aa_obj[25:50] + 3 * np.log(160))

# output
print("AIC of NLL(a):", round(aic_nll_a, 2), \
      round(aic_nll_a_1, 2), round(aic_nll_a_2, 2))
print("AIC of NLL(A):", round(aic_nll_A, 2), \
      round(aic_nll_A_1, 2), round(aic_nll_A_2, 2))
print("AIC of NLL(aa):", round(aic_nll_aa, 2), \
      round(aic_nll_aa_1, 2), round(aic_nll_aa_2, 2))
print("BIC of NLL(a):", round(bic_nll_a, 2), \
      round(aic_nll_a_1, 2), round(aic_nll_a_2, 2))
print("BIC of NLL(A):", round(bic_nll_A, 2), \
      round(aic_nll_A_1, 2), round(aic_nll_A_2, 2))
print("BIC of NLL(aa):", round(bic_nll_aa, 2), \
      round(aic_nll_aa_1, 2), round(aic_nll_aa_2, 2))
```

```
# image export
# plt.savefig('s3.pgf', format='pgf')
```

Python code: section (j)

```
# (j) Model recovery and confusion matrix (10)

# config
theta_mean_set = np.zeros([6, 3])
theta_std_set = np.zeros([6, 3])
range_025_975 = np.zeros([6, 3, 2])
# 95% interval in normal distribution, excluding extreme samples

for i in range(2):
    theta_mean_set[i, 0:2] = np.mean(theta_opt[(25*i):(25*(i+1)), :], \
                                     axis = 0) # group i mean[alpha, beta]
    theta_std_set[i, 0:2] = np.std(theta_opt[(25*i):(25*(i+1)), :], \
                                   axis = 0) # group i std[alpha, beta]
    theta_mean_set[i+2, :] = np.mean(theta_opt_A[(25*i):(25*(i+1)), :], \
                                     axis = 0) # group i mean[alpha, beta, A]
    theta_std_set[i+2, :] = np.std(theta_opt_A[(25*i):(25*(i+1)), :], \
                                   axis = 0) # group i std[alpha, beta, A]
    theta_mean_set[i+4, :] = np.mean(theta_opt_aa[(25*i):(25*(i+1)), :], \
                                     axis = 0) # group i mean[alpha+, alpha-, beta]
    theta_std_set[i+4, :] = np.std(theta_opt_aa[(25*i):(25*(i+1)), :], \
                                   axis = 0) # group i std[alpha+, alpha-, beta]
for i in range(6):
    range_025_975[i, :, 0] = theta_mean_set[i, :] - 1.96 * theta_std_set[i, :]
    range_025_975[i, :, 1] = theta_mean_set[i, :] + 1.96 * theta_std_set[i, :]
theta_alpha_index = [[0], [0, 2], [0, 1]]
theta_beta_index = [[1], [1], [2]]
param_num_3 = [2, 3, 3]
param_num_6 = [2, 2, 3, 3, 3, 3]
V0 = [0.5, 0.5]
n = 1
update_set = [update_1, update_2, update_3]
decision = softmax_decision
subsample_size = 250
nll_set = [nll, nll_A, nll_aa]
theta_init_set = [np.mean(theta_opt, axis = 0), np.mean(theta_opt_A, axis = 0), \
                  np.mean(theta_opt_aa, axis = 0)]

# sample parameters
theta_subsample = np.zeros([6, 3, subsample_size])
for k in range(6):
    for i in range(param_num_6[k]):
        for j in range(subsample_size):
            while theta_subsample[k, i, j] <= range_025_975[k, i, 0] or \
            theta_subsample[k, i, j] >= range_025_975[k, i, 1] or \
            theta_subsample[k, i, j] == 0:
                theta_subsample[k, i, j] = np.random.normal(theta_mean_set[k, i], \
                                                            theta_std_set[k, i])

# simulation and fit by optimizing 3 different NLLs and statistics
nll_values_3 = np.zeros([3, 3, 2*subsample_size]) # [nll_true, nll_fit, num]
for p in range(3):
    for q in range(2):
        for i in range(subsample_size):
            choices_subsample, outcomes_subsample = \
            simulation(V0, n, update_set[p], decision, \
                       theta_subsample[2*p+q, theta_alpha_index[p], i], \
                       theta_subsample[2*p+q, theta_beta_index[p], i], sample = True)
```

```
            for k in range(3):
                nll_values_3[p, k, subsample_size*q+i] = \
                minimize(nll_set[k], theta_init_set[k], args = \
                        (choices_subsample, outcomes_subsample, V0), \
                        method = "Nelder-Mead")['fun']

# AIC and BIC
aic_nll_3 = np.zeros([3, 3, 2*subsample_size])
bic_nll_3 = np.zeros([3, 3, 2*subsample_size])
for i in range(3):
    aic_nll_3[:, i, :] = 2 * nll_values_3[:, i, :] + 2 * param_num_3[i]
    bic_nll_3[:, i, :] = 2 * nll_values_3[:, i, :] + param_num_3[i] * np.log(160)

# confusion matrix
aic_cm = np.zeros([3, 3])
bic_cm = np.zeros([3, 3])
for i in range(3):
    for j in range(subsample_size):
        aic_win_index = np.argmin(aic_nll_3[i, :, j])
        aic_cm[i, aic_win_index] = aic_cm[i, aic_win_index] + 1
        bic_win_index = np.argmin(bic_nll_3[i, :, j])
        bic_cm[i, bic_win_index] = bic_cm[i, bic_win_index] + 1
aic_cm = np.round(aic_cm / subsample_size, 2)
bic_cm = np.round(bic_cm / subsample_size, 2)

# t-test
alpha_pos_mean_1 = theta_opt_aa[0:25, 0]
alpha_pos_mean_2 = theta_opt_aa[25:50, 0]
alpha_neg_mean_1 = theta_opt_aa[0:25, 1]
alpha_neg_mean_2 = theta_opt_aa[25:50, 1]
beta_mean_1 = theta_opt_aa[0:25, 2]
beta_mean_2 = theta_opt_aa[25:50, 2]
t_alpha_pos = stats.ttest_ind(alpha_pos_mean_1, alpha_pos_mean_2)
t_alpha_neg = stats.ttest_ind(alpha_neg_mean_1, alpha_neg_mean_2)
t_beta = stats.ttest_ind(beta_mean_1, beta_mean_2)

# output
print("Confusion matrix under AIC is:")
print(aic_cm)
print("Confusion matrix under BIC is:")
print(bic_cm)
print("The t-statistic of alpha+:", round(t_alpha_pos[0], 2))
print("The p-value of alpha+:", round(t_alpha_pos[1], 2))
print("The t-statistic of alpha-:", round(t_alpha_neg[0], 2))
print("The p-value of alpha-:", round(t_alpha_neg[1], 2))
print("The t-statistic of beta:", round(t_beta[0], 2))
print("The p-value of beta:", round(t_beta[1], 2))
print("Degrees of freedom:", 47) # (n1+n2-3)
```