Computational Cognitive Neuroscience: Computational modelling of behavioural data

March 26, 2023

I. Introduction

This report explores and attempts to understand whether anxiety leads people to differ in how they learn to predict adverse events in the future. Anxiety, by definition in this report, is related to worry and pessimistic about negative events that may occur in the future. This report is on modelling behavioural impairments during a probabilistic reversal learning task, inspired by an experiment carried out by Andrea D'Olimpio for his Master by Research in the Seri'es' Lab. The experiment required participants to engage in an avoidance learning task consisting of a Pavlovian and an instrumental phase. Participants who self-identified as either anxious or calm were recruited from the general population.

Before starting the task, all participants were asked to complete the State-Trait Anxiety Inventory (Spielberger et al., 1999), form Y, a widely-used questionnaire for assessing anxiety levels. This questionnaire has two distinct forms, Y1 and Y2, which measure state and trait anxiety, respectively. The scores for each form range from 20 (not anxious) to 80 (very anxious).

During the task, participants had to choose between two stimuli, one leading to a loud, unpleasant noise (i.e., an aversive outcome) and the other resulting in a neutral outcome (i.e., no sound). In the Pavlovian phase, the computer would select the stimulus for each trial, and the subjects would simply observe. During the instrumental phase, participants were asked to make choices to maximize their chances of avoiding the aversive outcome.

The focus of this report is on the instrumental phase of the experiment. The subject population includes 50 participants, 25 each to the group with and without anxiety denoted by their STAI score. The experiment consisted of 160 trials in which the participant chose either stimulus A or B. Every 40 trials, the probability of an aversive outcome led by certain stimuli changes according to the following ratios: 70/30, 80/20, 60/40, 65/35 (e.g., stimulus A leads to the aversive outcome 70% of the time and stimuli B 30% of the time during the first 40 trials). Participants were unaware of either the probabilities or their changes, and they had to infer them from the observed outcomes, aiming to avoid the aversive outcome as much as possible.

The experiment's author modelled the participants' behaviour through 9 reinforcement learning models, and this report aims to re-implement 3 of them. The base model (Model 1) is a simple model in which the value of the chosen stimulus i is updated on trial t after observing outcome o (0 for no sound and 1 for sound) as follows;

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

where the initial values of $V^{(0)}$ are set to 0.5, and when the outcome of one stimulus choice is observed, the value of the alternative option is not updated. Intuitively, the values of options A and B at trial $(t+1)$ are updated based on the prediction error of the chosen option at trial (t). It means that if the outcome is better than expected, the subjective value of the chosen option is increased, and if the outcome is worse than expected, the subjective value of the chosen option is decreased.

The probability of choosing stimulus A as opposed to stimulus B on trial (t) is modelled using a softmax function;

$$p(A|\ V^{(t)}, \beta) = \frac{exp(-\beta \cdot V_A^{(t)})}{exp(-\beta \cdot V_A^{(t)}) + exp(-\beta \cdot V_B^{(t)})} \tag{2}$$

where the β parameter takes a negative value as the task is to be avoided; hence, choose the less likely option to lead to the aversive outcome.

To sum up, there are two parameters to be focused on for the base model: the learning rate α in the observation model and the inverse temperature β in the decision model. The hypothesis is that trait anxiety could be related to changes in avoidance learning (e.g., an altered learning rate) or some change in decision-making (inverse temperature).

II. Data exploration

The mean, standard deviation, and median of the STAI score for the whole population are 42.72, 14.73, and 40.0, respectively. Separately for the two groups, the mean STAI scores are 55.24 for the anxious and 30.2 for the healthy group. The t-test for mean comparison shows the test statistics to be 11.17 for a null hypothesis that the two groups have the same mean for STAI scores. Thus, the null is rejected in favour of the alternative hypothesis that the two groups have different means for the STAI score.

The cut-off STAI score for a participant to be considered in the anxious group is 43. An STAI score of less than or equal to 43 is treated as healthy in this experiment. Amongst the 50 participants, the first 25 self-reported belonging to the anxious group, and the last 25 were recruited from the calm population. It is important to ensure that the participants do fit into the correct group based on their STAI score. Upon computation, it was noticed that 29 of the 50 participants belonged to the healthy group based on the STAI score. The obtained indices of subjects with $STAI \leq 43$ are,

$$[\mathbf{3}, \mathbf{15}, \mathbf{16}, \mathbf{24}, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49]$$

with 0 as the starting index. It can be seen that although the healthy controls have reported correctly in line with their STAI score, four individuals (bold index in the list above) in the anxious group have misreported their condition. Therefore, it can pose biases to the experiment. (However, for the rest of the report, the first 25 participants will be assumed to be highly anxious and the last 25 as low anxious.)

The number of times each participant in the anxious group chooses option A is computed as,

$$[52, 46, 32, 58, 37, 26, 28, 50, 38, 51, 53, 30, 31, 41, 36, 33, 41, 41, 28, 37, 53, 33, 40, 67, 31]$$

and the number of times each participant in the healthy group chooses option A is,

$$[45, 29, 6, 25, 7, 12, 29, 21, 29, 16, 36, 39, 27, 41, 25, 39, 28, 29, 36, 45, 24, 16, 53, 31, 22]$$

The average number of times option A is chosen for the whole population is 34.46. Comparing the two groups, the anxious one had nine individuals choosing option A, less than the average count, while the healthy one had 17.

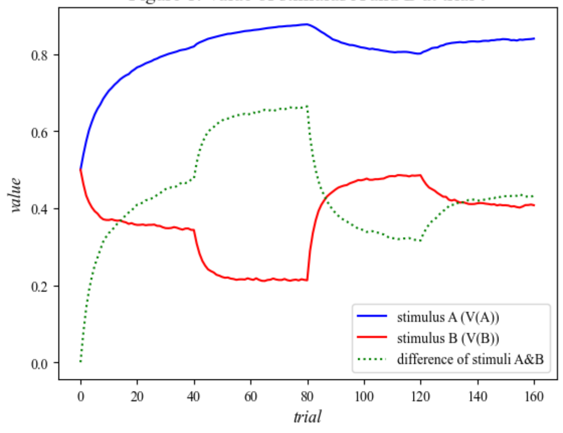
From equation (2), the expected number of aversive sounds experienced by participants who respond randomly throughout the experiment would be 80. It is because if the participant makes a random choice (in other words, learning nothing), the β parameter will become 0, and the overall probability of choosing option A becomes $\frac{1}{2}$. Overall, the participants are said to have performed well given that they did not make decisions randomly and obtained an average aversive outcome rate of 38.5%. A note to be made here is that the anxious group had a higher average percentage of receiving aversive outcomes at 40.28% while the healthy group achieved 36.65%.

III. Simulation

A python simulation to generate data is carried out to implement the model using equations (1) and (2). The simulation generates 160 choices of either option A or B with parameter settings for the two equations to be $\alpha = 0.4$, $\beta = 7$, and $V_0 = 0.5$. The simulation is carried out 10,000 times, and the evolution of the average of values of options A and B and their differences are illustrated in figure 1.

The evolution of V(A) and V(B) represents how the values of the two options change over time as more choices are made. The evolution of the difference in values (V(A) - V(B)) captures the relative preference between the two options over time. The shape of the evolution makes sense as the agent makes more choices, the values V(A) and V(B) converge to the true expected rewards of the options, and the learning rate α controls the convergence rate. A higher β value indicates more greedy choices, meaning that the agent is more likely to choose the option with the higher value. As the simulation progresses and the agent learns the true values, it becomes more likely to consistently choose the option with a higher expected reward. At the beginning of the simulation, the difference between V(A) and V(B) is positive and increasing, meaning that the agent is learning to prefer option A over option B. However, the difference starts to decrease around the 80th trial, where the preference for A over B gets weaker. Eventually, the difference converges to some values around 0.4.

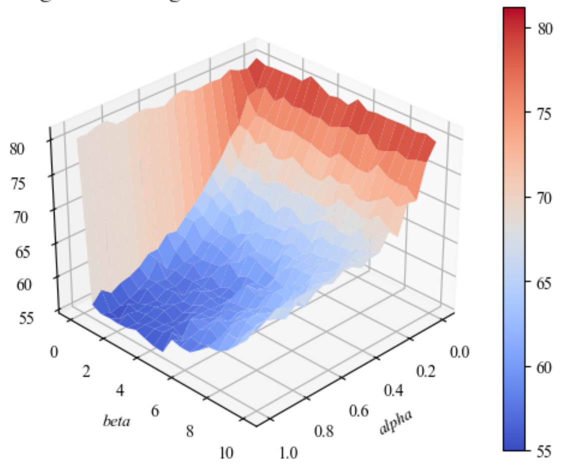
Figure 1: Value of stimulus A and B at trial t



IV. Parameter Setting Exploration

To explore how the parameter setting for α and β can affect the average number of aversive stimuli received, the simulation done in section III is run again but with different parameter values. The parameters in this simulation take values between (0,1) for α and (0,10) for β, and 20 evenly spaced numbers over the specified interval are used.

Figure 2: Average number of aversive outcomes



The simulation follows mainly the same procedure as in section III, forming the model based on equations (1) and (2) and counting the times an aversive outcome has occurred for each parameter setting. That said, there is a difference in that the number of simulations to be run for averaging is reduced to 100 instead of 10,000 to reduce computational time.

Figure 2 shows the result of the simulation. It can be seen that the lowest number of aversive outcomes is achieved by having a parameter setting of α reaching 1 and β reaching 0. As α gets smaller, the number of aversive outcomes grows, holding β constant for comparison. On the other hand, as β gets smaller, the number of aversive outcomes also gets smaller for a given α. An interesting difference is that for α, regardless of the value of β, the trend is very clear and higher α leads to reduced aversive outcomes. Whereas for β, such a trend in relation to the number of aversive outcomes is more obvious when α is larger. When α is reaching 0, the differences in the number of aversive outcomes due to β are almost invisible. A higher alpha value may lead to faster adaptation to changes in the environment, while a higher beta value may lead to more cautious behaviour and fewer aversive outcomes. However, there may be trade-offs between these parameters, as increasing one parameter may negatively affect performance if the other parameter is not also adjusted appropriately. Overall, it can be said that both α and β are important in explaining the number of received aversive outcomes, with the former being significant irrespective of its counterpart and the latter holding more clear significance when α is large.

V. Likelihood function

The likelihood of the parameters is needed to obtain the parameter values that best capture each participant's behaviour. The negative log-likelihood function can be used to obtain the negative log-likelihood of a given set of parameters (learning rate α and inverse temperature β) and observations (choices and outcomes).

$$NLL = -\sum_{c \in Choices} log[p(c|V,\theta)] \tag{3}$$

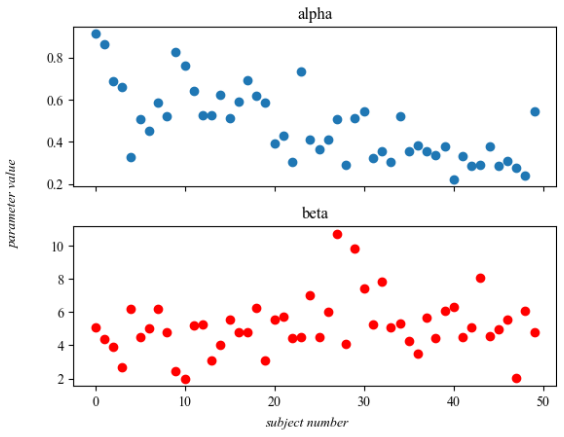
where θ is the parameter vector containing α and β. Upon computation in python, the 4th and 5th participants' NLL are obtained to be 101.61 and 52.69, respectively, with parameter settings of $\alpha = 0.4$, $\beta = 7$, and $V_0 = 0.5$.

VI. Model fitting

To find the best-fit parameter settings of α and β, an unconstrained optimisation (minimisation) is run using the Nelder-Mead algorithm in python's 'scipy.optimize' library. The objective is to find a set of parameters that would minimise the NLL for each individual using the NLL function computed in section V. The initial parameter values are set at $\alpha = 0.4$ and $\beta = 7$

The mean and variance of the fitted parameter value of the learning rate (α) are 0.48 and 0.029, respectively. The same for the inverse temperature (β) are 5.17 and 2.76, respectively. From figure 3, it can be seen that there is a clear decreasing trend in the fitted parameter for the learning rate (α), where the anxious group (the first 25 subjects) has higher values compared to the healthy group (the last 25 subjects). This can be explained by the fact that the anxious group is more careful about aversive future outcomes; hence, a higher learning rate to adapt and learn faster to avoid aversive outcomes as much as possible. Whereas for the fitted inverse temperature parameter, the trend is less obvious; a subtle increasing trend can be observed if inspected closely, with the last 25 individuals seeming to have a slightly higher value. Although the fitted alpha parameter all settled within the reasonable range of (0,1), there is one anomaly for the fitted beta parameter, which goes above the range of (0,10) at index 27 (the 28th individual).

Figure 3: Fitted parameter value for alpha and beta



Pearson's correlation between the estimated parameters (across all participants) is -0.24, with the p-value at 0.098, meaning that there is a negative correlation between the parameters, but it is unable to reject the null hypothesis that the distributions underlying the samples are uncorrelated and normally distributed. Separately for the anxious and healthy group, Pearson's correlation between the estimated parameters is -0.47 and 0.41, respectively. The p-value for the same null above 0.018 and 0.042, respectively, for the two groups, meaning that the correlation is statistically significant at the 5% significance level. It can be seen that the α and β parameters have different positioning and relation depending on whether the subject is anxious or not.

The negative correlation between α and β for the anxious could be interpreted as increased sensitivities to potentially negative outcomes and uncertainty compared to the healthy group. The higher the learning rate for an individual in the anxious group, the more sensitive they are to recent feedback, causing them to rapidly update their beliefs based on new information, which may lead them to prioritize recent experiences more heavily. Whereas for a lowered inverse temperature (given negative correlation), anxious individuals might be more prone to exploration, as they are uncertain about the potential outcomes of their actions. A lower β value corresponds to more random exploration and less deterministic choices. In this case, anxious individuals could be less confident in their learned values and more likely to explore different options.

VII. Group comparison

Using a two-sample t-test, a comparison can be made by examining whether the estimated parameter values are significantly different across groups. Two t-tests are carried out, each for the fitted alpha and beta parameters. The result is that both fitted alpha and beta are statistically significantly different across the two groups. The t-statistics for the fitted alpha parameter t-test is 6.04 with a p-value of 4.54×10^{-7}. The t-statistics for the fitted beta parameter t-test is -2.23, with a p-value of 0.031. In both tests, the null hypothesis is that the two groups have identical average values for the alpha and beta parameters. The degree of freedom for both tests is 48, as there are 50 observations and 2 are used for the comparison.

The higher learning rate in the anxious group suggests that individuals with trait anxiety might be more sensitive to recent feedback, updating their beliefs more rapidly based on new information. This is consistent with the hypothesis that trait anxiety could be related to changes in avoidance learning. The anxious group might prioritize recent experiences more heavily due to increased sensitivity to potentially negative outcomes or uncertainty, leading to a faster learning rate. The lower inverse temperature in the anxious group indicates that these individuals might be more prone to exploration and less confident in their learned values, leading to less deterministic choices. This supports the hypothesis that trait anxiety could be related to changes in decision-making. The lower inverse temperature could be a reflection of the anxious group's increased uncertainty or heightened sensitivity to negative outcomes, causing them to explore different options.

If the data showing that the anxious group has a higher learning rate and a lower inverse temperature compared to the healthy group were real, it could be said that anxious individuals may exhibit a faster learning rate in avoidance learning, which could be attributed to their heightened sensitivity to negative outcomes. This faster learning rate might make anxious individuals more reactive to changes in their environment but could also make their decision-making less stable or more prone to fluctuations. The lower inverse temperature in the anxious group suggests that their decision-making process might be characterized by increased exploration and less deterministic choices. The lower inverse temperature might indicate that anxious individuals are less confident in their learned values, causing them to explore different options.

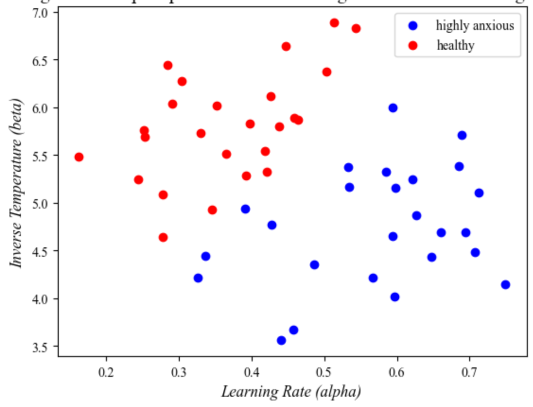
VIII. Parameter recovery

To check the reliability and identifiability of the parameter estimates, 50 sets of parameter values are sampled from a multivariate normal distribution. The sampling distribution is defined as follows,

$$\begin{pmatrix}\alpha_{A,H}\\ \beta_{A,H}\end{pmatrix} \sim N\left[\begin{pmatrix}\mu_\alpha\\ \mu_\beta\end{pmatrix} \quad \begin{pmatrix}0.01 & 0\\ 0 & 0.5\end{pmatrix}\right]$$

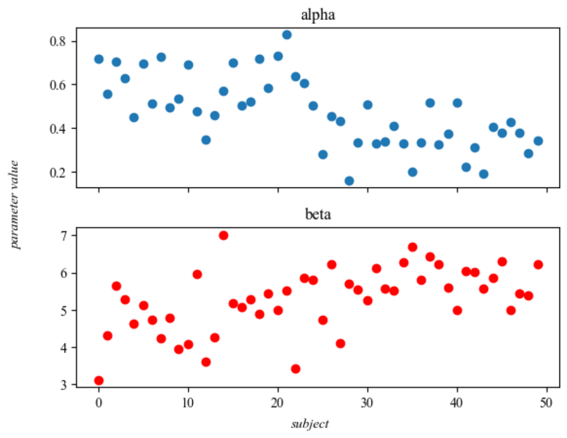
where μ_α is a vector of $(0.59, 0.36)$ and μ_β is a vector of $(4.66, 5.68)$, indicating differing mean of alpha and beta for the anxious and healthy groups. The sampled parameters are distributed as in figure 4's scatter plot.

Figure 4: Sampled parameter value for high anxious and control group



As can be observed, the parameter values clearly have different distributions for anxious and healthy groups. The distribution of the sampled parameters for the anxious group (blue) has a higher learning rate but a lower inverse temperature as defined by their fitted parameter means, as explored in the previous sections.

Figure 5: Fitted parameter value for alpha and beta post sampling



The sampled parameter is then used to simulate 50 sets of data (as in section III). The newly sampled parameter values are fitted to these simulated data sets (as in section VI). This process is carried out five times, and Pearson's correlation between the sampled and fitted parameters all lay within the range of (0.9999999999999998, 1). The consistently high correlations indicate that the model is reliably recovering the simulated parameters, suggesting that the parameter estimates are identifiable. A pattern is observed that as the simulation was carried out, the correlation statistics increased, despite their negligibly small differences. The result met expectations made before the simulation that the correlation should be reasonably high as the model fitting is done based on data simulated by itself. It indicates that the model can accurately recover the underlying parameters from the simulated data. This is important for validating the model and ensuring it can be used to make meaningful inferences about the data.

IX. Alternative model

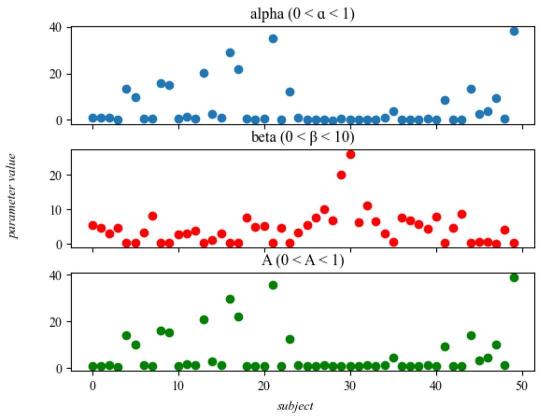
An alternative model (Model 2) is formulated by introducing a new parameter, A.

$$V_i^{(t+1)} = A \cdot V_i^{(t)} + \alpha \cdot (o^{(t)} - V_i^{(t)}) \tag{4}$$

Action selection for this model is performed by the same softmax function described by equation(2). In this model, if A is close to 1, it means the person's choice heavily relies on the previous value $V^{(t)}$ and does not update the value function significantly. On the contrary, if A is close to 0, it means the person's choice does not rely much on the previous value $V^{(t)}$ and updates the value function primarily based on the prediction error $(o^{(t)} - V_i^{(t)})$. Hence, the A parameter can be described as a 'memory' factor that accounts for the possibility that the influence of past experiences on the value function may decay over time.

A model fitting is performed where the negative log-likelihood of this new model is minimised through the Nelder-Mead algorithm in the 'scipy.optimize' package to find the best-fit parameter settings of α, β, and A for the given data. The NLL computed in this model for the 4th and 5th participants is 82.20 and 64.15, respectively, indicating a mixed performance of the model compared to Model 1's NLL computed in section VI. At first, the optimisation was done without constraint as in the same procedure as section VI, but the results were hard to interpret as fitted values for all three parameters had values went outside of their reasonable range ($0 < \alpha < 1$, $0 < \beta < 10$, $0 < A < 1$). Figure 6 illustrates the distribution of the fitted parameter obtained with unconstrained optimisation.

Figure 6: Fitted parameter value for alpha, beta, and A (Model 2, without constraint)



This problem may be explained by model misspecification for Model 2. It can be that this model is performing poorly in capturing the underlying data-generating process. Therefore, the optimization may compensate for this by pushing the parameters outside their reasonable range to minimize the error. For better interpretation, an alternative procedure, constraint optimisation, is carried out with constraints put on the range of the three parameters to be fitted.

Figure 7: Fitted parameter value for alpha, beta, and A (Model 2, with constraint)

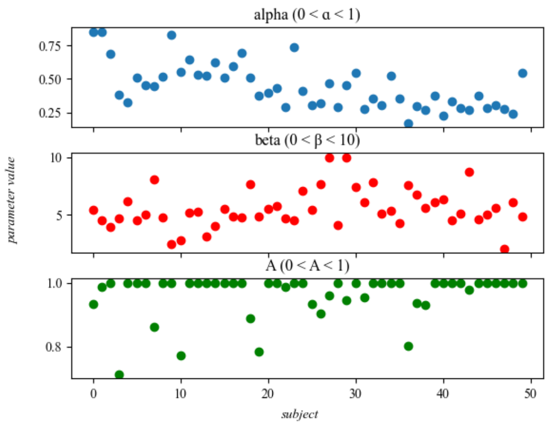


Figure 7 illustrates the updated results; the three parameters now fall within their reasonable ranges. The same behaviours of the α and β parameters can be observed from this model as in Model 1 that the former is higher for participants in the anxious group (first 25 subjects), and the latter omits differing trends between anxious and healthy groups. It is interesting to observe that most participants have their A parameter at values very close to 1, indicating that the past value $V^{(t)}$ has a strong influence on the updated value $V^{(t+1)}$ and the updated value function is influenced by both the past value and the observed outcome. It is also important to note that a few participants have A parameter less than 1, meaning that for them, the updated value function relies more heavily on the most recent outcome $o^{(t)}$ and the learning rate α. This implies that the participant is more sensitive to recent events and less influenced by past experiences when updating value estimates.

X. Model comparison

The sections so far have obtained negative log-likelihood values for each participant for Models 1 and 2. The negative log-likelihood for the two models at first glance is hard to make a comparison as some are lower than others, and vice versa. In quite a few cases, the NLL for Model 1 is lower than Model 2, although the differences are small. It can be explained by that Model 1 is a simpler model that better fits the data without over-fitting. It is possible that the additional memory parameter A in Model 2 does not provide any significant improvement in explaining the data (as most of them are close to 1), and as a result, Model 1 has a better fit.

For further comparisons, the AIC and BIC scores for the two models are computed,

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

where NLL is the negative log-likelihood, p is the number of parameters, and n is the number of observations. These two benchmarks help to compare models with different numbers of parameters by penalizing model complexity.

The results show that the AIC for Model 1 is lower than Model 2 at 6404.06 compared to 6987.50. The same is for BIC at 6711.58 for Model 1 as opposed to 7448.77 for Model 2. Considering the lower AIC and BIC values, Model 1 appears to be the "best" model in this case. It suggests that Model 1 better balances model fit and complexity for the given data. The simpler model seems to capture the underlying structure of the data without the need for the additional memory parameter A present in Model 2. However, this result can be simply a special case for the theoretical model when applied to the specific observational data provided for the experiment, where the participants exhibited some unaccounted-for characteristics.

XI. Model recovery and confusion matrix

To check the reliability of the model comparison procedure, data is simulated multiple times, as in section III, for each model. The simulated data set is then fitted to each model (as in section VI) to obtain AIC and BIC for model comparison.

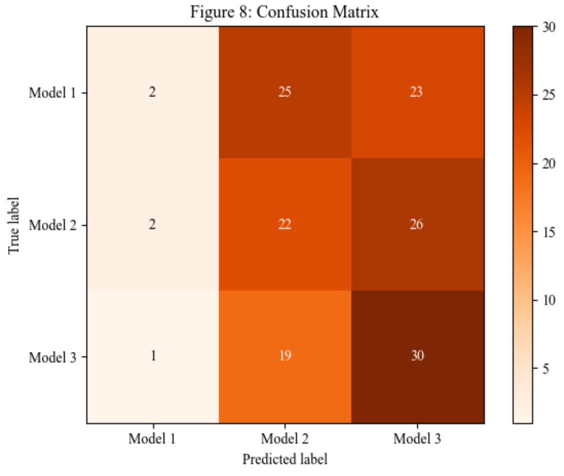
Table 1 shows the AIC and BIC scores for all models (Model 3 for section X), with the row showing which model generated the data and the column showing which model the data is fitted to. Figure 8 shows the confusion matrix for this process, showing the correct identification rate for each model.

TABLE I
AIC (BIC) of simulated data fitted to models

Model	Model 1	Model 2	Model 3
Model 1	15025.19 (15332.71)	13512.65 (13973.93)	14311.58 (14772.85)
Model 2	12510.44 (12817.96)	11089.27 (11550.54)	11096.96 (11558.24)
Model 3	14770.14 (15077.66)	13351.96 (13813.24)	13177.30 (13638.58)

It can be seen that simulated data generated from Model 2 has the lowest AIC and BIC upon fitting to all three models. On the contrary, the confusion matrix shows that Model 3 has the highest classification accuracy, as its diagonal element has the highest value. The result implies that the data generated from Model 2 may have some characteristics that make it more amenable to being modelled by all three models. It could mean that the data is less complex or that the underlying structure of the data is more easily captured by the models, which contradicts the earlier finding that for the observational

Figure 8: Confusion Matrix



data, Model 1 has the lower AIC. Whereas Model 3 seems to perform well in classification accuracy, as shown by the higher diagonal elements in the confusion matrix, better at correctly identifying the true underlying model when fitting the data generated by each model. This contradiction can be due to different characteristics of the data generated by each model or the models' ability to capture the underlying structure of the data. It can be argued that the data used in this report may have been generated by either Model 2 or 3.

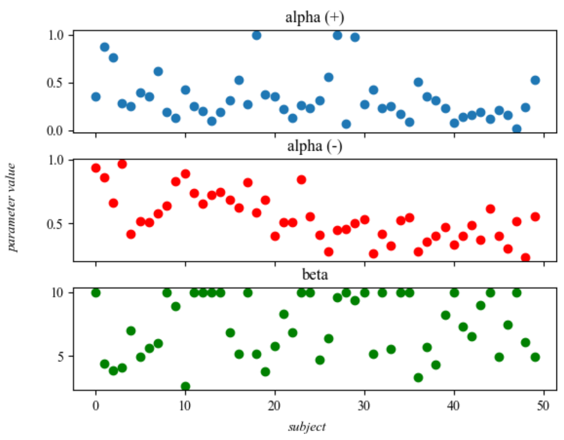
XII. Discussion and extra model

Lastly, another model, Model 3, is introduced with two learning rates, one for neutral outcomes ($o^{(t)} = 0$), α^+ and one for punishments, α^-,

$$V_i^{(t+1)} = V_i^{(t)} + ((1 - o^{(t)}) \cdot \alpha^+ + o^{(t)} \cdot \alpha^-) \cdot (o^{(t)} - V_i^{(t)}) \qquad (5)$$

Action selection for this model is again performed by the same softmax function defined in equation (2). The negative log-likelihood of the model is computed, and the parameter optimisation is carried out as in section IX using the Nelder-Mead Algorithm. Similarly to section IX, the optimisation is done with constraints on the range the three parameters could take on to improve interoperability ($0 < \alpha^+ < 1$, $0 < \alpha^- < 1$, $0 < \beta < 10$).

Figure 9: Fitted parameter value for alpha and beta (with constraint)



The result shows a clear decreasing trend for the α^- parameter. The AIC and BIC for Model 3 are also computed using the original observational data, and they are 6055.81 and 6517.09, respectively, the lowest for all three models formed so far. It indicates that Model 3 best fits the observation data of this particular experiment.

Group comparison t-test is carried out to test for Model 3's estimated parameters. The α^+ and β parameters for the anxious and healthy group returned to have differences that are statistically insignificant, with a t-statistic of 0.86 and p-value of 0.39 for α^+ and a t-statistic of -0.52 and p-value of 0.60 for β. It is unable to reject the null of the same means for α^+ and β for the anxious and healthy group at a 5% significance level. On the other hand, the α^- parameter for the anxious and healthy group does have a statistically significant difference with a t-statistic of 6.80 and a p-value of 3.11×10^{-8}. The degree of freedom for the three tests carried out is all 48. Thus, this model shows that the main difference between the anxious and healthy groups is how each group learns from negative outcomes. The higher average of α^- obtained for the anxious group implies that they are more sensitive to negative outcomes or losses, and their learning is more strongly influenced by these outcomes than the healthy group. This increased sensitivity to negative outcomes might make the anxious group more prone to adjusting their beliefs or expectations more rapidly after experiencing a negative outcome. Consequently, this could lead to heightened avoidance behaviour or a stronger focus on avoiding potential threats, which is a finding consistent with the conclusion in section VII.

From this interesting result, given the differences in learning processes, specifically the heightened sensitivity to negative outcomes for the anxious group, anxiety treatments can be adapted to meet these specific circumstances. It could be beneficial for therapies to be less 'hard' or negative. For example, for exposure therapy, it may be helpful to adopt a more gradual approach, with smaller steps and less intense negative experiences initially. This can help anxious individuals build up their tolerance to negative outcomes while not overwhelming them. Furthermore, therapies for these anxious participants could focus more on positive reinforcement. During therapy, positive reinforcement can play a crucial role in helping anxious individuals overcome their heightened sensitivity to negative outcomes. By emphasizing and praising their successes, therapists can encourage patients to focus more on the positive aspects of their experiences.

References

[1] Spielberger, C. D., Sydeman, S. J., Owen, A. E., and Marsh, B. J. (1999). Measuring anxiety and anger with the State-Trait Anxiety Inventory (STAI) and the State-Trait Anger Expression Inventory (STAXI).

[2] Lecture slides from the module 'Computational Cognitive Neuroscience' by Professor Peggy Series.

code

March 28, 2023

[2]:
```
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import confusion_matrix
plt.rcParams["font.family"] = "Times New Roman"
```

[3]:
```
stai_scores = np.array(pd.read_csv(r'stai_scores.csv', header=None)[0])
inst_choices = np.array(pd.read_csv(r'inst_choices.csv', header=None))
inst_outcomes = np.array(pd.read_csv(r'inst_outcomes.csv', header=None))
```

0.0.1 Data exploration

[4]:
```
# Exploring the data: mean, median, standard deviation of STAI
overall_mean = np.mean(stai_scores); print(overall_mean)
overall_std = np.std(stai_scores); print(overall_std)
overall_median = np.median(stai_scores); print(overall_median)

anxious_mean = np.mean(stai_scores[0:25]); print(anxious_mean)
anxious_std = np.std(stai_scores[0:25]); print(anxious_std)
anxious_median = np.median(stai_scores[0:25]); print(anxious_median)

control_mean = np.mean(stai_scores[25:]); print(control_mean)
control_std = np.std(stai_scores[25:]); print(control_std)
control_median = np.median(stai_scores[25:]); print(control_median)
print(sp.stats.ttest_ind(stai_scores[:25], stai_scores[25:], equal_var=False))
```

```
42.72
14.730974170094793
40.0
55.24
9.84390166549829
57.0
30.2
4.857983120596448
29.0
Ttest_indResult(statistic=11.174860157303042, pvalue=4.2293214098115444e-13)
```

[5]:
```python
# Exploring the data: cut-off = 43, ANXIOUS group if STAI>43
cut_off_stai = np.zeros(len(stai_scores))
for i in range(len(stai_scores)):
    if stai_scores[i] <= 43:
        # cut_off_stai = 1 if the subject is in the healthy control group
        cut_off_stai[i] = 1

healthy_num = sum(cut_off_stai)
healthy_index = np.where(cut_off_stai==1)
print(healthy_num)
print(healthy_index)
```

```
29.0
(array([ 3, 15, 16, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37,
       38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49]),)
```

[6]:
```python
# Exploring the data: number of times each subject choosing stimuli A
chose_a_count = []
chose_a_percent = []
for i in range(len(inst_choices)):
    chose_a_count.append(np.array(np.where(inst_choices[i]==1)).size)
    chose_a_percent.append(chose_a_count[-1]/len(inst_choices[i]))
print(chose_a_count)
print(chose_a_percent)
print(np.average(chose_a_count))
```

```
[52, 46, 32, 58, 37, 26, 28, 50, 38, 51, 53, 30, 31, 41, 36, 33, 41, 41, 28, 37,
53, 33, 40, 67, 31, 45, 29, 6, 25, 7, 12, 29, 21, 29, 16, 36, 39, 27, 41, 25,
39, 28, 29, 36, 45, 24, 16, 53, 31, 22]
[0.325, 0.2875, 0.2, 0.3625, 0.23125, 0.1625, 0.175, 0.3125, 0.2375, 0.31875,
0.33125, 0.1875, 0.19375, 0.25625, 0.225, 0.20625, 0.25625, 0.25625, 0.175,
0.23125, 0.33125, 0.20625, 0.25, 0.41875, 0.19375, 0.28125, 0.18125, 0.0375,
0.15625, 0.04375, 0.075, 0.18125, 0.13125, 0.18125, 0.1, 0.225, 0.24375,
0.16875, 0.25625, 0.15625, 0.24375, 0.175, 0.18125, 0.225, 0.28125, 0.15, 0.1,
0.33125, 0.19375, 0.1375]
34.46
```

[7]:
```python
print(np.where(chose_a_count[0:25]<=np.average(chose_a_count)))
print(np.where(chose_a_count[25:]<=np.average(chose_a_count)))
```

```
(array([ 2,  5,  6, 11, 12, 15, 18, 21, 24]),)
(array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 12, 14, 16, 17, 20, 21, 23, 24]),)
```

[8]:
```python
print(np.average(inst_outcomes))
print(np.average(np.average(inst_outcomes[:25],axis=1)))
print(np.average(np.average(inst_outcomes[25:],axis=1)))
```

```
0.384625
```

```
0.40275
0.3665
```

0.0.2 Simulation

```
[9]: np.random.seed(1)
     prob_list = [[0.3,0.7],[0.2,0.8],[0.4,0.6],[0.35,0.65]]

     def outcome(a,b,p,V0):

         outcome = np.zeros(160)
         choice = []
         value_a = [V0]
         value_b = [V0]
         prob = np.repeat(p,40,axis=0)

         for i in range(160):
             p_a = np.exp(-b*value_a[i])/(np.exp(-b*value_a[i])+np.
      ↪exp(-b*value_b[i]))
             p_b = 1-p_a
             choice.append(np.random.choice([0,1],p=[p_a,p_b]))
             if choice[-1] == 0:
                 outcome[i] = np.random.choice([0,1],p=prob[i])
                 value_a.append(value_a[-1]+a*(outcome[i]-value_a[-1]))
                 value_b.append(value_b[-1])
             else:
                 outcome[i] = np.random.choice([0,1],p=1-prob[i])
                 value_b.append(value_b[-1]+a*(outcome[i]-value_b[-1]))
                 value_a.append(value_a[-1])

         return outcome, choice, value_a, value_b, sum(outcome)


     # if correct, 'np.mean(test)' should be around 58
     test = []
     for i in range(1000):
         test.append(outcome(0.4,7,prob_list,0.5)[-1])
     np.mean(test)
```

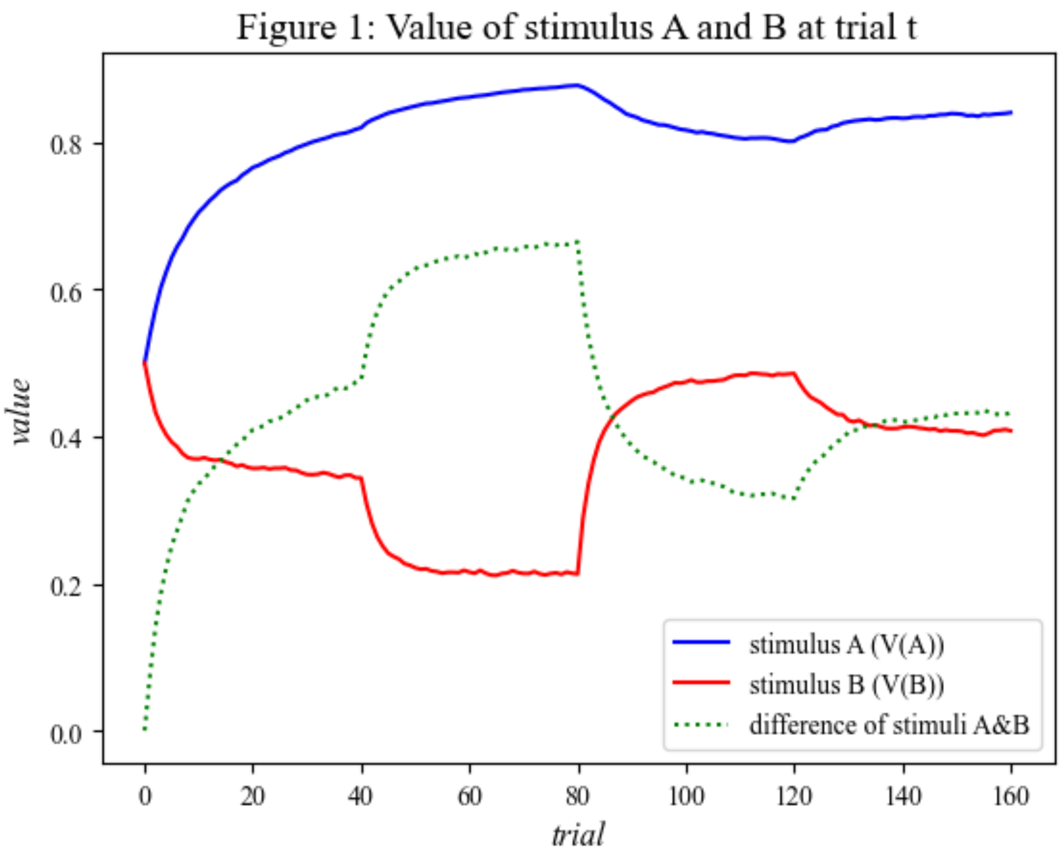
```
[9]: 58.124
```

```
[10]: np.random.seed(1)
      n = 10000
      value_a = np.zeros([n,161])
      value_b = np.zeros([n,161])
      for i in range(n):
          value_a[i] = outcome(0.4,7,prob_list,0.5)[2]
```

```
    value_b[i] = outcome(0.4,7,prob_list,0.5)[3]
value_a = np.mean(value_a,axis=0)
value_b = np.mean(value_b,axis=0)
```

[11]:
```
plt.title('Figure 1: Value of stimulus A and B at trial t', fontsize =␣
 ↪'x-large')
plt.plot(value_a, color = 'b', label = "stimulus A (V(A))")
plt.plot(value_b, color = 'r', label = "stimulus B (V(B))")
plt.plot(np.array(value_a)-np.array(value_b), color = 'g', linestyle =␣
 ↪'dotted', label = "difference of stimuli A&B")
plt.xlabel('trial', fontsize = 'large', fontstyle = 'italic')
plt.ylabel('value', fontsize = 'large', fontstyle = 'italic')
plt.legend(loc='lower right')
```

[11]:
```
<matplotlib.legend.Legend at 0x13cb15280>
```

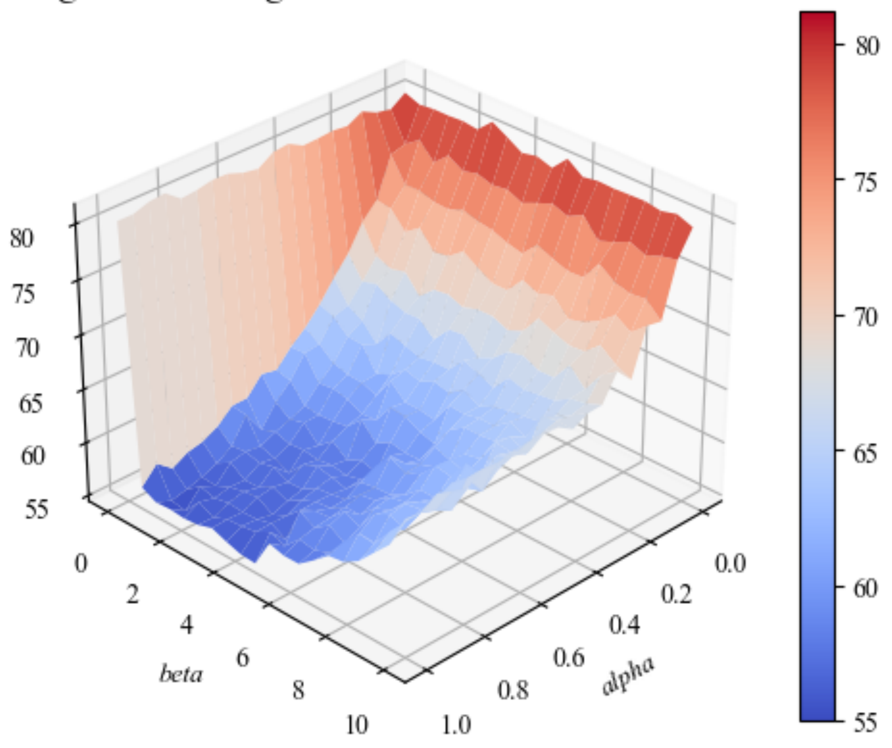


[12]:
```
np.random.seed(1)
num = 20
n = 100
```

```
sim_mat = np.zeros([num,num])
alpha = (np.linspace(0,1,num))
beta = (np.linspace(0,10,num))
for i in range(num):
    for j in range(num):
        sim = np.zeros(n)
        for k in range (n):
            sim[k] = outcome(alpha[i],beta[j],prob_list,0.5)[-1]
        sim_mat[i][j] = np.mean(sim)
```

[13]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.view_init(30, 45)
X, Y = np.meshgrid(alpha, beta)
plot = ax.plot_surface(X, Y, sim_mat, cmap='coolwarm', vmin=np.min(sim_mat),␣
 ↪vmax=np.max(sim_mat))
ax.set_xlabel('alpha',fontstyle = 'italic')
ax.set_ylabel('beta',fontstyle = 'italic')
ax.set_title('Figure 2: Average number of aversive outcomes',fontsize =␣
 ↪'x-large')
fig.colorbar(plot)

plt.show()
```

Figure 2: Average number of aversive outcomes



0.0.3 Likelihood function

[14]:
```
def nll(params, choices, outcome, V0):
    value_a = [V0]
    value_b = [V0]
    p_a = []
    c_a = np.zeros(len(choices))
    c_a[np.where(choices==1)] = 1
    c_b = np.zeros(len(choices))
    c_b[np.where(choices==2)] = 1
    ll = 0
    a, b = params

    for i in range(len(choices)):
        if choices[i] == 1:
            value_a.append(value_a[-1]+a*(outcome[i]-value_a[-1]))
            value_b.append(value_b[-1])
        else:
            value_b.append(value_b[-1]+a*(outcome[i]-value_b[-1]))
            value_a.append(value_a[-1])

        p_a.append(np.ma.exp(-b*(value_a[i]))*(np.ma.exp(-b*(value_a[i]))+np.ma.
 ↪exp(-b*(value_b[i])))**(-1))
        ll -= ((np.ma.log(p_a[-1])*c_a[i]) + (np.ma.log((1-p_a[-1]))*c_b[i]))

    return ll

## test should be around 61
test = nll([0.4,7],inst_choices[8],inst_outcomes[8],0.5); print(test)
nll4 = nll([0.4,7],inst_choices[3],inst_outcomes[3],0.5); print(nll4)
nll5 = nll([0.4,7],inst_choices[4],inst_outcomes[4],0.5); print(nll5)
```

```
60.877333438783324
101.6050206224349
52.68861467295329
```

0.0.4 Model fitting

[15]:
```
fitted_param = []
for i in range(len(stai_scores)):
    fitted_param.append(sp.optimize.minimize(nll,x0=(0.4,7),␣
 ↪args=(inst_choices[i],inst_outcomes[i],0.5),method='Nelder-Mead').x)
```

```
[16]: fitted_alpha, fitted_beta = np.array(fitted_param).T

      mean_alpha = np.mean(fitted_alpha); print(mean_alpha)
      var_alpha = np.var(fitted_alpha); print(var_alpha)
      mean_beta = np.mean(fitted_beta); print(mean_beta)
      var_beta = np.var(fitted_beta); print(var_beta)

      mean_alpha_anxious = np.mean(fitted_alpha[:25]); print(mean_alpha_anxious)
      mean_alpha_control = np.mean(fitted_alpha[25:]); print(mean_alpha_control)
      mean_beta_anxious = np.mean(fitted_beta[:25]); print(mean_beta_anxious)
      mean_beta_control = np.mean(fitted_beta[25:]); print(mean_beta_control)

      print(sp.stats.pearsonr(fitted_alpha, fitted_beta))
      print(sp.stats.pearsonr(fitted_alpha[:25], fitted_beta[:25]))
      print(sp.stats.pearsonr(fitted_alpha[25:], fitted_beta[25:]))
```

```
0.4758865798481678
0.028823177110009483
5.165364859938008
2.7646122988261386
0.5874739980369317
0.36429916165940396
4.655724358126161
5.6750053617498555
PearsonRResult(statistic=-0.2369656492300812, pvalue=0.09753716137988659)
PearsonRResult(statistic=-0.46745261030155233, pvalue=0.01846598704926908)
PearsonRResult(statistic=0.40900638521961774, pvalue=0.04234454107340218)
```

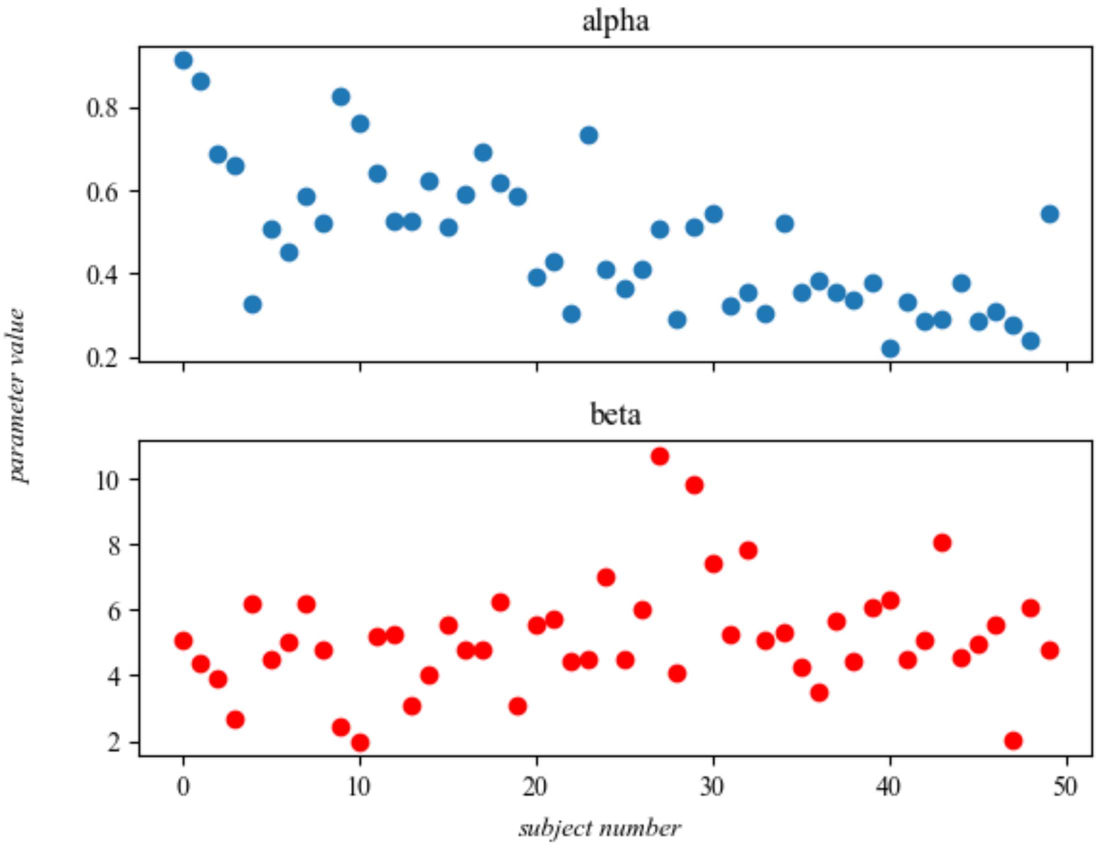
```
[17]: np.where(fitted_beta>10)
```

```
[17]: (array([27]),)
```

```
[18]: fig, ax = plt.subplots(2, sharex=True)
      fig.suptitle('Figure 3: Fitted parameter value for alpha and beta')
      fig.text(0.5, 0.03, 'subject number', horizontalalignment ='center',␣
       ↪verticalalignment = 'center', fontstyle = 'italic')
      fig.text(0.03, 0.5, 'parameter value', horizontalalignment ='center',␣
       ↪verticalalignment = 'center', rotation = 'vertical', fontstyle = 'italic')
      fig.subplots_adjust(hspace = 0.25)
      ax[0].scatter(np.arange(0,50,1),fitted_alpha)
      ax[0].set_title('alpha')
      ax[1].scatter(np.arange(0,50,1),fitted_beta, color = 'red')
      ax[1].set_title('beta')
```

```
[18]: Text(0.5, 1.0, 'beta')
```

Figure 3: Fitted parameter value for alpha and beta



0.0.5 Group comparison

```
[19]: dof = len(fitted_alpha[:25]) + len(fitted_alpha[25:]) - 2; print(dof)
      print(sp.stats.ttest_ind(fitted_alpha[:25], fitted_alpha[25:], equal_var=False))
      print(sp.stats.ttest_ind(fitted_beta[:25], fitted_beta[25:], equal_var=False))
```

```
48
Ttest_indResult(statistic=6.042165319443778, pvalue=4.539257722057438e-07)
Ttest_indResult(statistic=-2.2309569492149133, pvalue=0.0311105314742668 66)
```

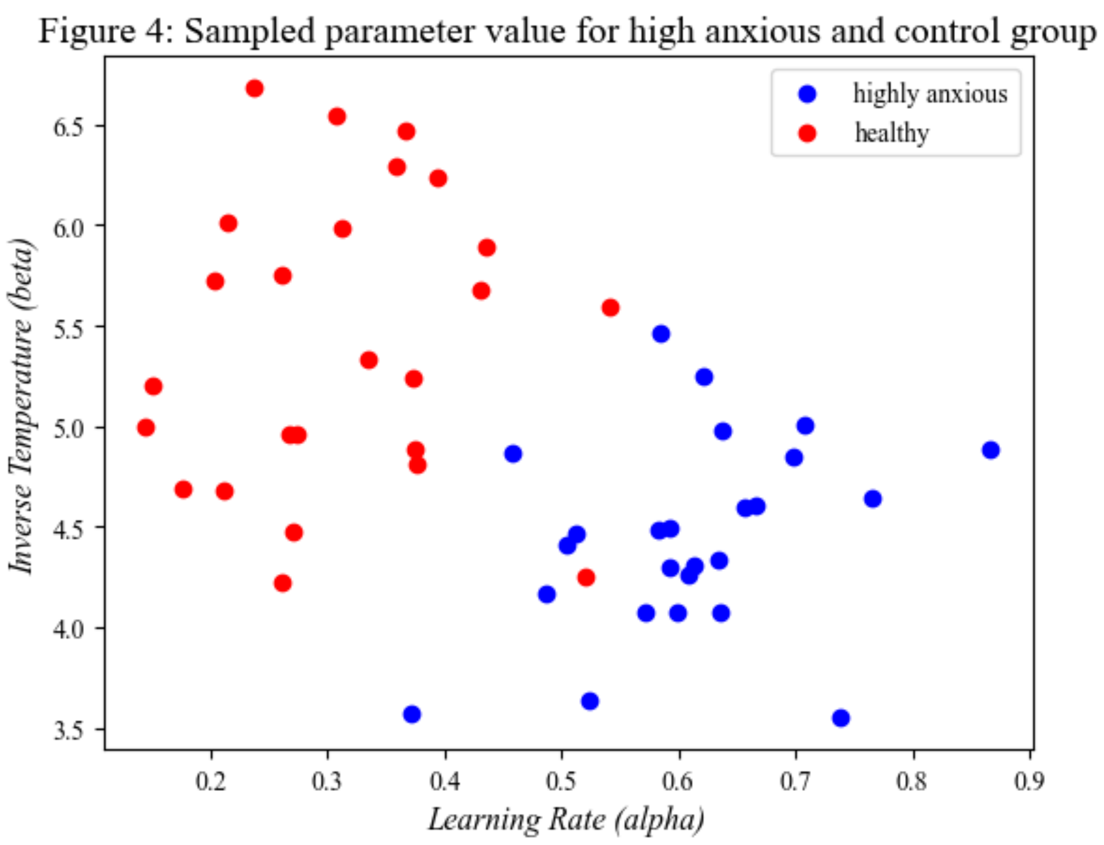
0.0.6 Parameter recovery

```
[20]: def gaussian(mean_alpha_anxious, mean_beta_anxious,mean_alpha_control,␣
       ↪mean_beta_control):
          var_learning_rate = 0.01
          var_inv_temp = 0.5
          cov = 0
```

```
    # anxious group
    mean = [mean_alpha_anxious, mean_beta_anxious]
    covariance = [[var_learning_rate, cov], [cov, var_inv_temp]]
    param_values = np.random.multivariate_normal(mean, covariance, 25)

    min_learning_rate = 0
    max_learning_rate = 1
    min_inv_temp = 0
    max_inv_temp = 10

    while True:
        invalid_indices = np.where((param_values[:, 0] < min_learning_rate) |
                                (param_values[:, 0] > max_learning_rate) |
                                (param_values[:, 1] < min_inv_temp) |
                                (param_values[:, 1] > max_inv_temp))[0]
        if len(invalid_indices) == 0:
            break
        param_values[invalid_indices] = np.random.multivariate_normal(mean,␣
 ↪covariance, len(invalid_indices))


    # control group
    mean = [mean_alpha_control, mean_beta_control]
    covariance = [[var_learning_rate, cov], [cov, var_inv_temp]]
    param_values_c = np.random.multivariate_normal(mean, covariance, 25)

    while True:
        invalid_indices = np.where((param_values_c[:, 0] < min_learning_rate) |
                                (param_values_c[:, 0] > max_learning_rate) |
                                (param_values_c[:, 1] < min_inv_temp) |
                                (param_values_c[:, 1] > max_inv_temp))[0]
        if len(invalid_indices) == 0:
            break
        param_values_c[invalid_indices] = np.random.multivariate_normal(mean,␣
 ↪covariance, len(invalid_indices))

    return np.concatenate((param_values, param_values_c))

param_values = gaussian(mean_alpha_anxious,␣
 ↪mean_beta_anxious,mean_alpha_control, mean_beta_control)

plt.title('Figure 4: Sampled parameter value for high anxious and control␣
 ↪group', fontsize = 'x-large')
plt.scatter(param_values[:25, 0], param_values[:25, 1],color = 'b', label =␣
 ↪"highly anxious")
plt.scatter(param_values[25:, 0], param_values[25:, 1],color = 'r', label =␣
 ↪"healthy")
```

```
plt.xlabel('Learning Rate (alpha)',fontsize = 'large', fontstyle = 'italic')
plt.ylabel('Inverse Temperature (beta)',fontsize = 'large', fontstyle =␣
  ↪'italic')
plt.legend(loc='upper right')
```

```
[20]: <matplotlib.legend.Legend at 0x13ce8ce80>
```



```
[21]: def param_sim(param_values):
          sim_choice = []
          sim_outcome = []
          for i in range(50):
              sim_choice.
   ↪append((outcome(param_values[i][0],param_values[i][1],prob_list,0.5))[1])
              sim_outcome.
   ↪append((outcome(param_values[i][0],param_values[i][1],prob_list,0.5))[0])

          fitted_param_1 = []
          for i in range(len(stai_scores)):
```

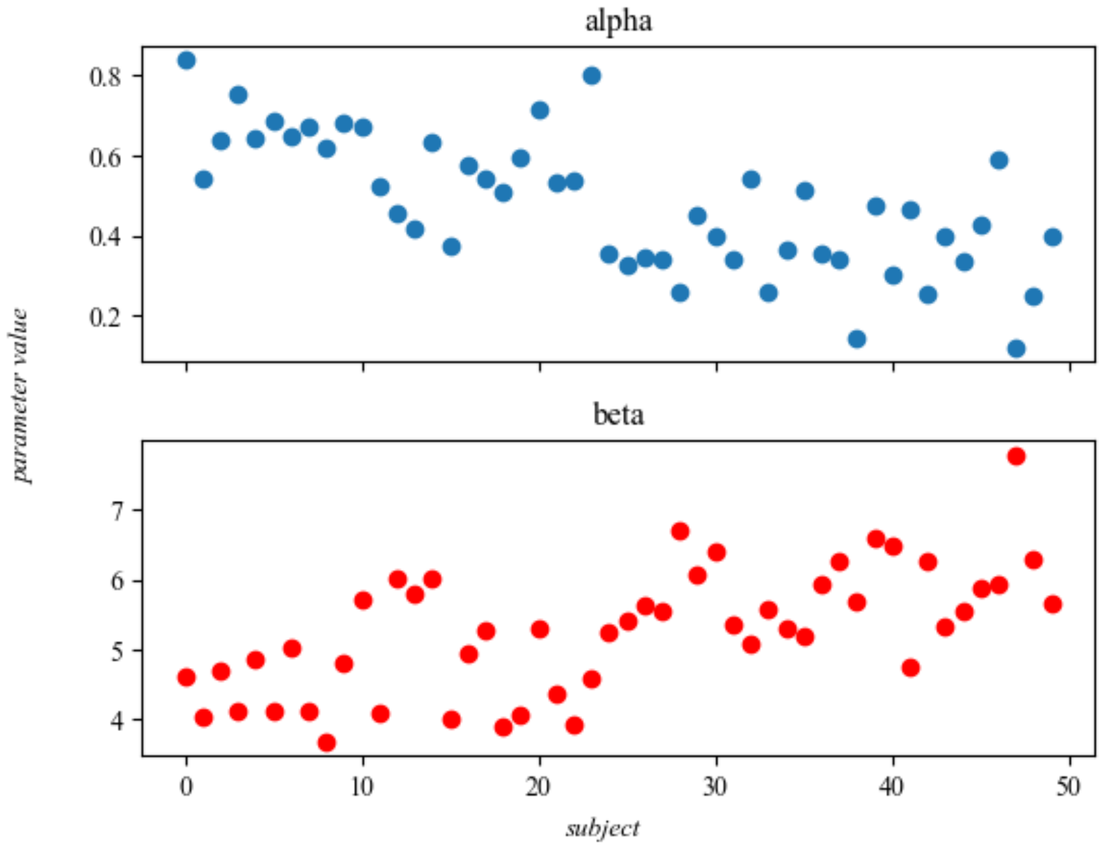
```
        fitted_param_1.append(sp.optimize.
↪minimize(nll,x0=(param_values[i][0],param_values[i][1]),␣
↪args=(sim_choice[i],sim_outcome[i],0.5),method='Nelder-Mead').x)

    fitted_alpha_1, fitted_beta_1 = np.array(fitted_param_1).T
    print(sp.stats.pearsonr(fitted_alpha_1, np.array(param_values).T[0]))
    print(sp.stats.pearsonr(fitted_beta_1, np.array(param_values).T[1]))
```

[22]:
```
for i in range(5):
    param_values = gaussian(mean_alpha_anxious,␣
↪mean_beta_anxious,mean_alpha_control, mean_beta_control)
    param_sim(param_values)

fig, ax = plt.subplots(2, sharex=True)
fig.suptitle('Figure 5: Fitted parameter value for alpha and beta post␣
↪sampling')
fig.text(0.5, 0.03, 'subject', horizontalalignment ='center', verticalalignment␣
↪= 'center', fontstyle = 'italic')
fig.text(0.03, 0.5, 'parameter value', horizontalalignment ='center',␣
↪verticalalignment = 'center', rotation = 'vertical', fontstyle = 'italic')
fig.subplots_adjust(hspace = 0.25)
ax[0].scatter(np.arange(0,50,1),np.array(param_values).T[0])
ax[0].set_title('alpha')
ax[1].scatter(np.arange(0,50,1),np.array(param_values).T[1], color = 'red')
ax[1].set_title('beta')
```

```
PearsonRResult(statistic=0.9999999999999998, pvalue=0.0)
PearsonRResult(statistic=0.9999999999999999, pvalue=0.0)
PearsonRResult(statistic=1.0, pvalue=0.0)
PearsonRResult(statistic=1.0, pvalue=0.0)
PearsonRResult(statistic=1.0, pvalue=0.0)
PearsonRResult(statistic=0.9999999999999997, pvalue=0.0)
PearsonRResult(statistic=1.0, pvalue=0.0)
PearsonRResult(statistic=1.0, pvalue=0.0)
PearsonRResult(statistic=1.0, pvalue=0.0)
PearsonRResult(statistic=1.0, pvalue=0.0)
```

[22]:
```
Text(0.5, 1.0, 'beta')
```

Figure 5: Fitted parameter value for alpha and beta post sampling



0.0.7 Alternative model

[23]:
```python
def outcome_alt(A,a,b,p,V0):

    outcome = np.zeros(160)
    choice = []
    value_a = [V0]
    value_b = [V0]
    prob = np.repeat(p,40,axis=0)

    for i in range(160):
        p_a = np.exp(-b*value_a[i])/(np.exp(-b*value_a[i])+np.
 ↪exp(-b*value_b[i]))
        p_b = 1-p_a
        choice.append(np.random.choice([0,1],p=[p_a,p_b]))
        if choice[-1] == 0:
            outcome[i] = np.random.choice([0,1],p=prob[i])
            value_a.append(A*value_a[-1]+a*(outcome[i]-value_a[-1]))
            value_b.append(value_b[-1])
        else:
            outcome[i] = np.random.choice([0,1],p=1-prob[i])
            value_b.append(A*value_b[-1]+a*(outcome[i]-value_b[-1]))
            value_a.append(value_a[-1])

    return outcome, choice, value_a, value_b, sum(outcome)

def nll_alt(params, choices, outcome, V0):
    value_a = [V0]
    value_b = [V0]
    p_a = []
    c_a = np.zeros(len(choices))
    c_a[np.where(choices==1)] = 1
    c_b = np.zeros(len(choices))
    c_b[np.where(choices==2)] = 1
    ll = 0
    A, a, b = params

    for i in range(len(choices)):
        if choices[i] == 1:
            value_a.append(A*value_a[-1]+a*(outcome[i]-value_a[-1]))
            value_b.append(value_b[-1])
        else:
            value_b.append(A*value_b[-1]+a*(outcome[i]-value_b[-1]))
            value_a.append(value_a[-1])

        p_a.append(np.ma.exp(-b*(value_a[i]))*(np.ma.exp(-b*(value_a[i]))+np.ma.
 ↪exp(-b*(value_b[i])))**(-1))
        ll -= ((np.ma.log(p_a[-1])*c_a[i]) + (np.ma.log((1-p_a[-1]))*c_b[i]))

    return ll

test_alt = []
for i in range(1000):
    test_alt.append(outcome_alt(0.5,0.4,5,prob_list,0.5)[-1])
print(np.mean(test_alt))

test_alt1 = nll_alt([0.5,0.4,5],inst_choices[8],inst_outcomes[8],0.5);␣
 ↪print(test_alt1)
nll4_alt = nll_alt([0.5,0.4,5],inst_choices[3],inst_outcomes[3],0.5);␣
 ↪print(nll4_alt)
nll5_alt = nll_alt([0.5,0.4,5],inst_choices[4],inst_outcomes[4],0.5);␣
 ↪print(nll5_alt)

bounds = ((0,1),(0,1),(0,10))
```

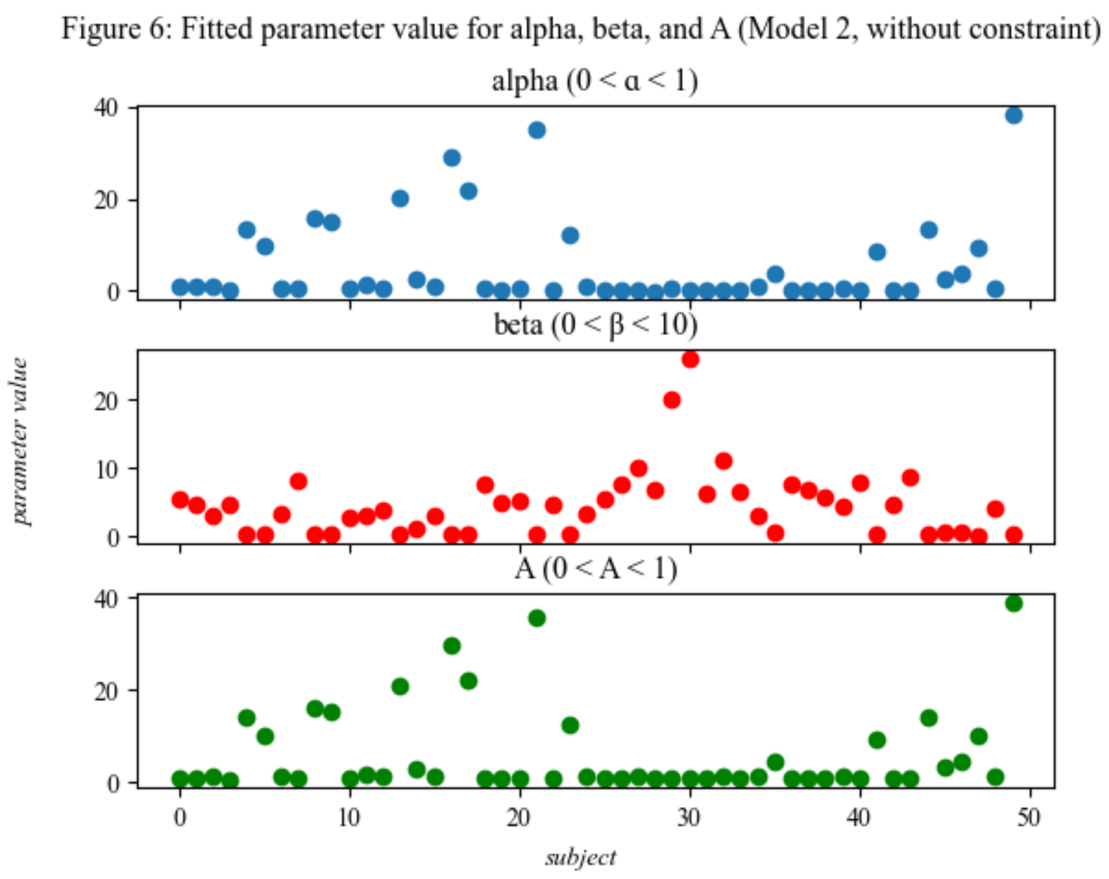
```
fitted_param_alt = []
for i in range(len(stai_scores)):
    fitted_param_alt.append(sp.optimize.minimize(nll_alt,x0=(0.5,0.4,5),␣
↪args=(inst_choices[i],inst_outcomes[i],0.5),method='Nelder-Mead').x)
fitted_a_alt, fitted_alpha_alt, fitted_beta_alt = np.array(fitted_param_alt).T
```

```
68.736
68.16237790958874
82.20317887028379
64.15342352148336
```

```
/var/folders/k8/kf3w914s7_l6n18t6bqjl9280000gn/T/ipykernel_36138/771713305.py:43
: RuntimeWarning: divide by zero encountered in double_scalars
  p_a.append(np.ma.exp(-b*(value_a[i]))*(np.ma.exp(-b*(value_a[i]))+np.ma.exp(-b
*(value_b[i])))**(-1))
/var/folders/k8/kf3w914s7_l6n18t6bqjl9280000gn/T/ipykernel_36138/771713305.py:43
: RuntimeWarning: invalid value encountered in double_scalars
  p_a.append(np.ma.exp(-b*(value_a[i]))*(np.ma.exp(-b*(value_a[i]))+np.ma.exp(-b
*(value_b[i])))**(-1))
```

```
[24]: fig, ax = plt.subplots(3, sharex=True)
fig.suptitle('Figure 6: Fitted parameter value for alpha, beta, and A (Model 2,␣
↪without constraint)')
fig.text(0.5, 0.03, 'subject', horizontalalignment ='center', verticalalignment␣
↪= 'center', fontstyle = 'italic')
fig.text(0.03, 0.5, 'parameter value', horizontalalignment ='center',␣
↪verticalalignment = 'center', rotation = 'vertical', fontstyle = 'italic')
fig.subplots_adjust(hspace = 0.25)
ax[0].scatter(np.arange(0,50,1),fitted_alpha_alt)
ax[0].set_title('alpha (0 <   < 1)')
ax[1].scatter(np.arange(0,50,1),fitted_beta_alt, color = 'red')
ax[1].set_title('beta (0 <   < 10)')
ax[2].scatter(np.arange(0,50,1),fitted_a_alt, color = 'green')
ax[2].set_title('A (0 < A < 1)')
```

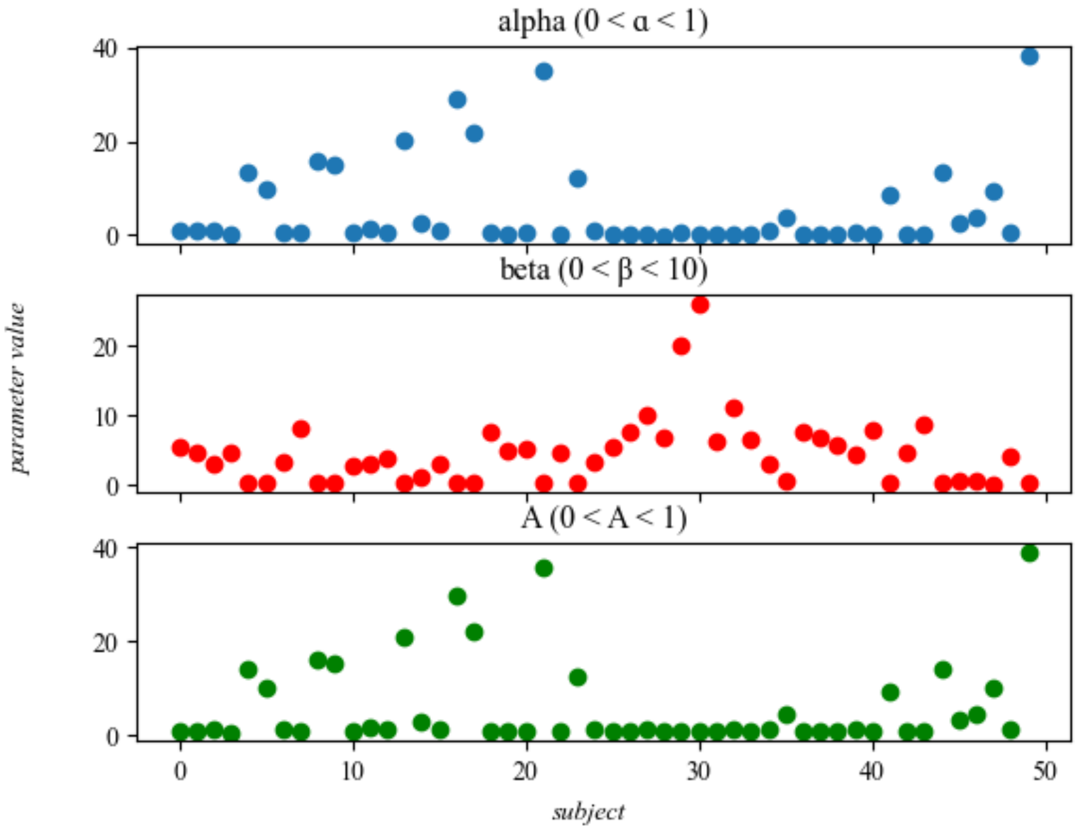
```
[24]: Text(0.5, 1.0, 'A (0 < A < 1)')
```



```
[25]: fig, ax = plt.subplots(3, sharex=True)
      fig.suptitle('Figure 7: Fitted parameter value for alpha, beta, and A (Model 2,␣
       ↪with constraint)')
      fig.text(0.5, 0.03, 'subject', horizontalalignment ='center', verticalalignment␣
       ↪= 'center', fontstyle = 'italic')
      fig.text(0.03, 0.5, 'parameter value', horizontalalignment ='center',␣
       ↪verticalalignment = 'center', rotation = 'vertical', fontstyle = 'italic')
      fig.subplots_adjust(hspace = 0.25)
      ax[0].scatter(np.arange(0,50,1),fitted_alpha_alt)
      ax[0].set_title('alpha (0 <   < 1)')
      ax[1].scatter(np.arange(0,50,1),fitted_beta_alt, color = 'red')
      ax[1].set_title('beta (0 <   < 10)')
      ax[2].scatter(np.arange(0,50,1),fitted_a_alt, color = 'green')
      ax[2].set_title('A (0 < A < 1)')
```

```
[25]: Text(0.5, 1.0, 'A (0 < A < 1)')
```

Figure 7: Fitted parameter value for alpha, beta, and A (Model 2, with constraint)



0.0.8 Model comparison

[26]:
```python
def AIC(nll,p):
    return 2*nll+2*p

def BIC(nll,p):
    return 2*nll+p*np.ma.log(160)

p1 = 2
p2 = 3
nll_list = []
nll_alt_list = []
AIC_list = []
AIC_alt_list = []
BIC_list = []
BIC_alt_list = []
for i in range(50):
    nll_list.append(nll([0.4,7],inst_choices[i],inst_outcomes[i],0.5))
    nll_alt_list.append(nll_alt([0.5,0.4,7],inst_choices[i],inst_outcomes[i],0.
↪5))
    AIC_list.append(AIC(nll_list[i],p1))
    AIC_alt_list.append(AIC(nll_alt_list[i],p2))
    BIC_list.append(BIC(nll_list[i],p1))
    BIC_alt_list.append(BIC(nll_alt_list[i],p2))

print(nll_list)
print(nll_alt_list)
print(np.sum(AIC_list))
print(np.sum(AIC_alt_list))
print(np.sum(BIC_list))
print(np.sum(BIC_alt_list))
```

```
[71.89501709469803, 73.99902852070569, 63.48782166698683, 101.6050206224349,
52.68861467295329, 49.57025421866699, 51.67640689447347, 45.83387519851256,
60.877333438783324, 96.87827871571137, 121.82979052419863, 55.29336425971195,
45.302956732572476, 89.96392053484107, 69.0933773743769, 49.75817622629023,
61.54695515722878, 59.49679091453134, 40.11875819260712, 88.01597053519866,
61.01663621129176, 48.3936151673412, 74.3030016861751, 67.12210773918608,
38.78085225784171, 78.6687645503986, 45.88392040154698, 14.90590504976406,
68.62440500070787, 14.216069440154536, 22.19957601403611, 61.98829560327881,
24.47260689318613, 57.10556070235257, 39.12374571586506, 75.054059133254,
82.86536965051516, 63.258209037338545, 77.92305108365144, 35.92353527738445,
70.25209035730177, 62.85353933223457, 73.12094788753735, 59.81012524236908,
71.10489274283533, 69.4479720246508, 40.93620401075032, 145.89307779287049,
60.93348773619517, 46.915889434129014]
[52.203947410800026, 55.099495389326734, 58.26546164394766, 85.92844463028467,
62.04020303395902, 53.485918055101294, 61.503461099355185, 52.25456883750361,
65.74883527384853, 83.52318499397687, 95.00457783995863, 55.29605706621358,
52.03676814872957, 82.07836896272387, 64.11960062355841, 57.584026515153504,
63.114596464905425, 55.03054296838704, 48.12880114821906, 74.24192815160292,
69.35747305840083, 59.655783966520524, 77.24256682738037, 60.934426821272055,
53.57053058727222, 88.00364872888476, 62.468728855052774, 34.9890073803676,
73.8616604463814, 31.18071470341474, 37.929273409057764, 78.30899532262715,
46.2897710979633, 68.35207303118385, 46.629665381470254, 79.16245431049198,
77.06678569000591, 73.30888358730361, 89.86523726642008, 48.70772587693919,
84.3309586018311, 66.941429569421, 88.91501072999763, 93.3098597089969,
83.95456922548551, 82.25179959556341, 51.88963475864121, 130.51971394972014,
77.72763573021165, 50.33299726884269]
6404.058449343255
6987.4956074893535
6711.575830866638
7448.771679774429
```

0.0.9 Model recovery and confusion matrix

[27]:
```python
nn = 50
outcome_m = np.zeros([n,160])
choice_m = np.zeros([n,160])
outcome_m_alt = np.zeros([n,160])
choice_m_alt = np.zeros([n,160])

for i in range(nn):
    outcome_m[i] = outcome(0.4,7,prob_list,0.5)[0]
    choice_m[i] = outcome(0.4,7,prob_list,0.5)[1]
    outcome_m_alt[i] = outcome_alt(0.5,0.4,5,prob_list,0.5)[0]
    choice_m_alt[i] = outcome_alt(0.5,0.4,5,prob_list,0.5)[1]
```

[28]:
```python
p1 = 2
p2 = 3
nll_list_m = []
nll_alt_list_m = []
AIC_list_m = []
AIC_alt_list_m = []
BIC_list_m = []
BIC_alt_list_m = []

nll_list_m1 = []
nll_alt_list_m1 = []
AIC_list_m1 = []
AIC_alt_list_m1 = []
BIC_list_m1 = []
BIC_alt_list_m1 = []

for i in range(50):
    nll_list_m.append(nll([0.4,7],choice_m[i],outcome_m[i],0.5))
    nll_alt_list_m.append(nll_alt([0.5,0.4,7],choice_m[i],outcome_m[i],0.5))
    AIC_list_m.append(AIC(nll_list_m[i],p1))
    AIC_alt_list_m.append(AIC(nll_alt_list_m[i],p2))
    BIC_list_m.append(BIC(nll_list_m[i],p1))
    BIC_alt_list_m.append(BIC(nll_alt_list_m[i],p2))

    nll_list_m1.append(nll([0.4,7],choice_m_alt[i],outcome_m_alt[i],0.5))
    nll_alt_list_m1.append(nll_alt([0.5,0.
 ↪4,7],choice_m_alt[i],outcome_m_alt[i],0.5))
    AIC_list_m1.append(AIC(nll_list_m1[i],p1))
    AIC_alt_list_m1.append(AIC(nll_alt_list_m1[i],p2))
    BIC_list_m1.append(BIC(nll_list_m1[i],p1))
    BIC_alt_list_m1.append(BIC(nll_alt_list_m1[i],p2))

print(np.sum(AIC_list_m))
print(np.sum(AIC_alt_list_m))
print(np.sum(BIC_list_m))
print(np.sum(BIC_alt_list_m))

print(np.sum(AIC_list_m1))
print(np.sum(AIC_alt_list_m1))
print(np.sum(BIC_list_m1))
print(np.sum(BIC_alt_list_m1))
```

```
15025.191471074295
13512.652071364038
15332.708852597678
13973.928143649115
12510.44003285552
11089.268885177486
12817.957414378903
11550.54495746256
```

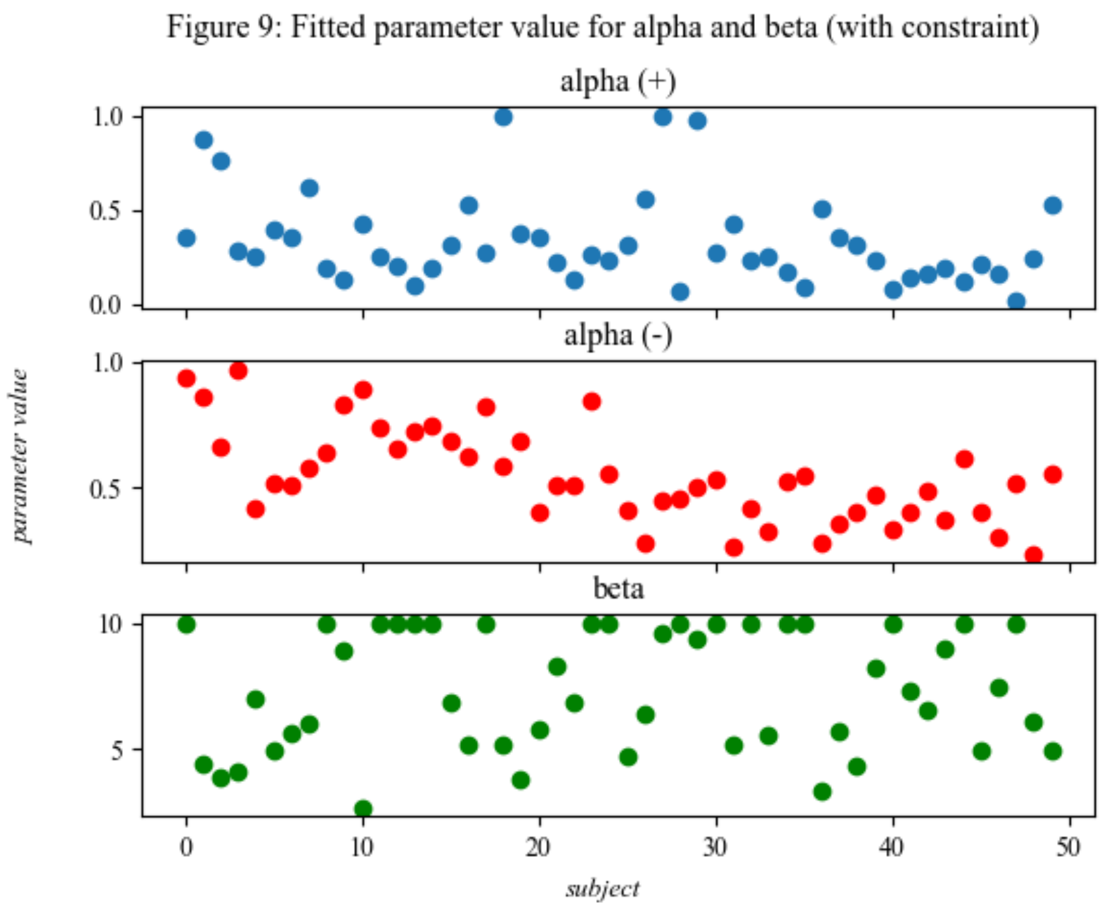
0.0.10 Discussion and extra model

[29]:
```
def outcome_ex(ap,an,b,p,V0):

    outcome = np.zeros(160)
    choice = []
    value_a = [V0]
    value_b = [V0]
    prob = np.repeat(p,40,axis=0)

    for i in range(160):
        p_a = np.exp(-b*value_a[i])/(np.exp(-b*value_a[i])+np.
 ↪exp(-b*value_b[i]))
        p_b = 1-p_a
        choice.append(np.random.choice([0,1],p=[p_a,p_b]))
        if choice[-1] == 0:
            outcome[i] = np.random.choice([0,1],p=prob[i])
            value_a.
 ↪append(value_a[-1]+((1-outcome[i])*ap+outcome[i]*an)*(outcome[i]-value_a[-1]))
            value_b.append(value_b[-1])
        else:
            outcome[i] = np.random.choice([0,1],p=1-prob[i])
            value_b.
 ↪append(value_b[-1]+((1-outcome[i])*ap+outcome[i]*an)*(outcome[i]-value_b[-1]))
            value_a.append(value_a[-1])

    return outcome, choice, value_a, value_b, sum(outcome)

def nll_ex(params, choices, outcome, V0):
    value_a = [V0]
    value_b = [V0]
    p_a = []
    c_a = np.zeros(len(choices))
    c_a[np.where(choices==1)] = 1
    c_b = np.zeros(len(choices))
    c_b[np.where(choices==2)] = 1
    ll = 0
    ap, an, b = params

    for i in range(len(choices)):
        if choices[i] == 1:
            value_a.
 ↪append(value_a[-1]+((1-outcome[i])*ap+outcome[i]*an)*(outcome[i]-value_a[-1]))
            value_b.append(value_b[-1])
        else:
            value_b.
 ↪append(value_b[-1]+((1-outcome[i])*ap+outcome[i]*an)*(outcome[i]-value_b[-1]))
            value_a.append(value_a[-1])

        p_a.append(np.ma.exp(-b*(value_a[i]))*(np.ma.exp(-b*(value_a[i]))+np.ma.
 ↪exp(-b*(value_b[i])))**(-1))
        ll -= ((np.ma.log(p_a[-1])*c_a[i]) + (np.ma.log((1-p_a[-1]))*c_b[i]))

    return ll


bounds = ((0,1),(0,1),(0,10))

fitted_param_ex = []
for i in range(len(stai_scores)):
    fitted_param_ex.append(sp.optimize.minimize(nll_ex,x0=(0.25,0.5,6),␣
 ↪args=(inst_choices[i],inst_outcomes[i],0.5),method='Nelder-Mead', bounds =␣
 ↪bounds).x)
fitted_ap_ex, fitted_an_ex, fitted_beta_ex = np.array(fitted_param_ex).T
```

```
[30]: fig, ax = plt.subplots(3, sharex=True)
fig.suptitle('Figure 9: Fitted parameter value for alpha and beta (with␣
 ↪constraint)')
fig.text(0.5, 0.03, 'subject', horizontalalignment ='center', verticalalignment␣
 ↪= 'center', fontstyle = 'italic')
fig.text(0.03, 0.5, 'parameter value', horizontalalignment ='center',␣
 ↪verticalalignment = 'center', rotation = 'vertical', fontstyle = 'italic')
fig.subplots_adjust(hspace = 0.25)
ax[0].scatter(np.arange(0,50,1),fitted_ap_ex)
ax[0].set_title('alpha (+)')
```

```
ax[1].scatter(np.arange(0,50,1),fitted_an_ex, color = 'red')
ax[1].set_title('alpha (-)')
ax[2].scatter(np.arange(0,50,1),fitted_beta_ex, color = 'green')
ax[2].set_title('beta')
```

```
[30]: Text(0.5, 1.0, 'beta')
```



```
[31]: print(sp.stats.ttest_ind(fitted_ap_ex[:25], fitted_ap_ex[25:],equal_var=False))
print(sp.stats.ttest_ind(fitted_an_ex[:25], fitted_an_ex[25:],equal_var=False))
print(sp.stats.ttest_ind(fitted_beta_ex[:25], fitted_beta_ex[25:
  ↪],equal_var=False))

print(np.mean(fitted_an_ex[:25]))
print(np.mean(fitted_an_ex[25:]))
```

```
Ttest_indResult(statistic=0.8619068384825213, pvalue=0.3930480785659023)
Ttest_indResult(statistic=6.796807499245872, pvalue=3.107958313010415e-08)
Ttest_indResult(statistic=-0.5233537098985926, pvalue=0.6031603092790152)
0.6762895118972989
```

```
0.41897380104088966
```

[32]:
```
nll_list_ex = []
AIC_ex = []
BIC_ex = []
for i in range(50):
    nll_list_ex.append(nll_ex([0.25,0.5,6],inst_choices[i],inst_outcomes[i],0.
 ↪5))
    AIC_ex.append(AIC(nll_list_ex[i],p2))
    BIC_ex.append(BIC(nll_list_ex[i],p2))
print(np.sum(AIC_ex))
print(np.sum(BIC_ex))
```

```
6055.809202083026
6517.085274368101
```

[33]:
```
outcome_m_ex = np.zeros([n,160])
choice_m_ex = np.zeros([n,160])

for i in range(nn):
    outcome_m_ex[i] = outcome_ex(0.25,0.5,6,prob_list,0.5)[0]
    choice_m_ex[i] = outcome_ex(0.25,0.5,6,prob_list,0.5)[1]

p2 = 3
nll_ex_list_m = []
AIC_ex_list_m = []
BIC_ex_list_m = []

for i in range(50):
    nll_ex_list_m.append(nll_ex([0.25,0.5,6],choice_m_ex[i],outcome_m_ex[i],0.
 ↪5))
    AIC_ex_list_m.append(AIC(nll_ex_list_m[i],p2))
    BIC_ex_list_m.append(BIC(nll_ex_list_m[i],p2))

print(np.sum(AIC_ex_list_m))
print(np.sum(BIC_ex_list_m))
```

```
13177.304146106908
13638.58021839198
```

[34]:
```
p1 = 2
p2 = 3
nll_list_m = []
nll_alt_list_m = []
nll_ex_list_m = []
AIC_list_m = []
AIC_alt_list_m = []
```

```
BIC_list_m = []
BIC_alt_list_m = []
AIC_ex_list_m = []
BIC_ex_list_m = []

nll_list_m1 = []
nll_alt_list_m1 = []
nll_ex_list_m1 = []
AIC_list_m1 = []
AIC_alt_list_m1 = []
BIC_list_m1 = []
BIC_alt_list_m1 = []
AIC_ex_list_m1 = []
BIC_ex_list_m1 = []

nll_list_m2 = []
nll_alt_list_m2 = []
nll_ex_list_m2 = []
AIC_list_m2 = []
AIC_alt_list_m2 = []
BIC_list_m2 = []
BIC_alt_list_m2 = []
AIC_ex_list_m2 = []
BIC_ex_list_m2 = []

for i in range(50):
    nll_list_m.append(nll([0.4,7],choice_m[i],outcome_m[i],0.5))
    nll_alt_list_m.append(nll_alt([0.5,0.4,7],choice_m[i],outcome_m[i],0.5))
    nll_ex_list_m.append(nll_ex([0.25,0.5,6],choice_m[i],outcome_m[i],0.5))
    AIC_list_m.append(AIC(nll_list_m[i],p1))
    AIC_alt_list_m.append(AIC(nll_alt_list_m[i],p2))
    AIC_ex_list_m.append(AIC(nll_ex_list_m[i],p2))
    BIC_list_m.append(BIC(nll_list_m[i],p1))
    BIC_alt_list_m.append(BIC(nll_alt_list_m[i],p2))
    BIC_ex_list_m.append(BIC(nll_ex_list_m[i],p2))

    nll_list_m1.append(nll([0.4,7],choice_m_alt[i],outcome_m_alt[i],0.5))
    nll_alt_list_m1.append(nll_alt([0.5,0.
 ↪4,7],choice_m_alt[i],outcome_m_alt[i],0.5))
    nll_ex_list_m1.append(nll_ex([0.25,0.
 ↪5,6],choice_m_alt[i],outcome_m_alt[i],0.5))
    AIC_list_m1.append(AIC(nll_list_m1[i],p1))
    AIC_alt_list_m1.append(AIC(nll_alt_list_m1[i],p2))
    AIC_ex_list_m1.append(AIC(nll_ex_list_m1[i],p2))
    BIC_list_m1.append(BIC(nll_list_m1[i],p1))
    BIC_alt_list_m1.append(BIC(nll_alt_list_m1[i],p2))
    BIC_ex_list_m1.append(BIC(nll_ex_list_m1[i],p2))
    nll_list_m2.append(nll([0.4,7],choice_m_ex[i],outcome_m_ex[i],0.5))
    nll_alt_list_m2.append(nll_alt([0.5,0.4,7],choice_m_ex[i],outcome_m_ex[i],0.
 ↪5))
    nll_ex_list_m2.append(nll_ex([0.25,0.5,6],choice_m_ex[i],outcome_m_ex[i],0.
 ↪5))
    AIC_list_m2.append(AIC(nll_list_m2[i],p1))
    AIC_alt_list_m2.append(AIC(nll_alt_list_m2[i],p2))
    AIC_ex_list_m2.append(AIC(nll_ex_list_m2[i],p2))
    BIC_list_m2.append(BIC(nll_list_m2[i],p1))
    BIC_alt_list_m2.append(BIC(nll_alt_list_m2[i],p2))
    BIC_ex_list_m2.append(BIC(nll_ex_list_m2[i],p2))


print(np.sum(AIC_list_m))
print(np.sum(AIC_alt_list_m))
print(np.sum(AIC_ex_list_m))
print(np.sum(BIC_list_m))
print(np.sum(BIC_alt_list_m))
print(np.sum(BIC_ex_list_m))
print('')
print(np.sum(AIC_list_m1))
print(np.sum(AIC_alt_list_m1))
print(np.sum(AIC_ex_list_m1))
print(np.sum(BIC_list_m1))
print(np.sum(BIC_alt_list_m1))
print(np.sum(BIC_ex_list_m1))
print('')
print(np.sum(AIC_list_m2))
print(np.sum(AIC_alt_list_m2))
print(np.sum(AIC_ex_list_m2))
print(np.sum(BIC_list_m2))
print(np.sum(BIC_alt_list_m2))
print(np.sum(BIC_ex_list_m2))
```

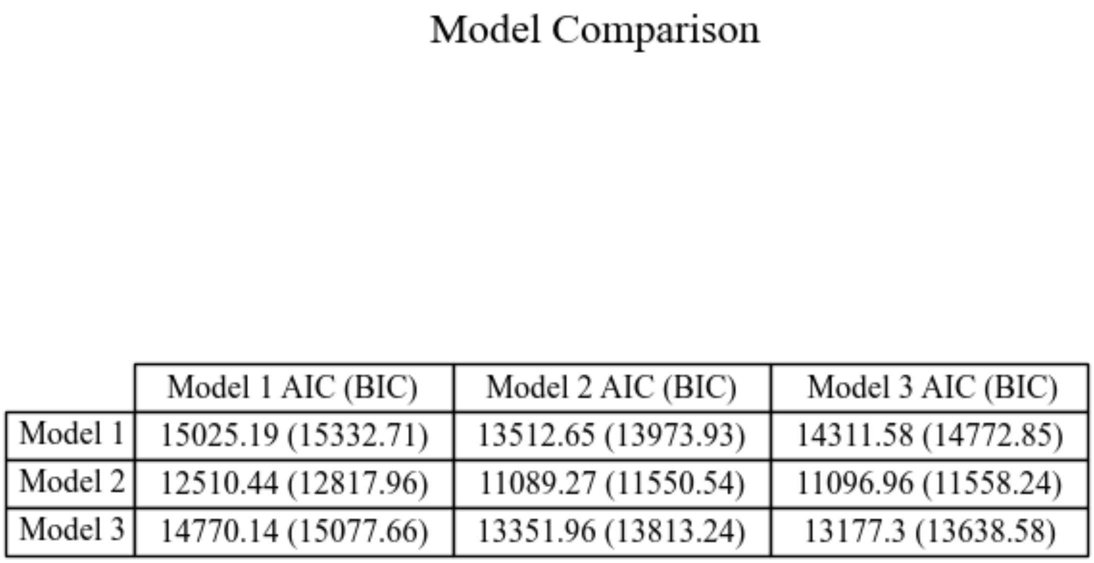
```
15025.191471074295
13512.652071364038
14311.575557501656
15332.708852597678
13973.928143649115
14772.851629786728

12510.44003285552
11089.268885177486
11096.96025033126
12817.957414378903
11550.54495746256
```

```
11558.236322616332

14770.13985509459
13351.963949455756
13177.304146106908
15077.657236617973
13813.24002174083
13638.58021839198
```

[35]:
```python
data = np.round([
    [15025.191471074295, 15332.708852597678, 13512.652071364038, 13973.
 ↪928143649115, 14311.575557501656, 14772.851629786728],
    [12510.44003285552, 12817.957414378903, 11089.268885177486, 11550.
 ↪54495746256, 11096.96025033126, 11558.236322616332],
    [14770.13985509459, 15077.657236617973, 13351.963949455756, 13813.
 ↪24002174083, 13177.304146106908, 13638.58021839198]], 2)

combined_data = []

for row in data:
    combined_row = [f"{row[0]} ({row[1]})", f"{row[2]} ({row[3]})", f"{row[4]}␣
 ↪({row[5]})"]
    combined_data.append(combined_row)

df = pd.DataFrame(combined_data)

column_names = ['Model 1 AIC (BIC)', 'Model 2 AIC (BIC)', ' Model 3 AIC (BIC)']
index_labels = ['Model 1', 'Model 2', 'Model 3']

df.columns = column_names
df.index = index_labels

print(df)

df.columns = column_names
df.index = index_labels

fig, ax = plt.subplots()
ax.axis('off')
ax.axis('tight')
table = ax.table(cellText=df.values, colLabels=df.columns, rowLabels=df.index,␣
 ↪cellLoc='center', loc='center')
table.auto_set_font_size(False)
table.set_fontsize(12)
table.scale(1, 1.5)
fig.suptitle("Model Comparison", fontsize=16)
ax.set_ylabel("Y-Axis Label", fontsize=14, labelpad=20, rotation=0, va='center')
```

```python
plt.show()
```

```
           Model 1 AIC (BIC)    Model 2 AIC (BIC)    Model 3 AIC (BIC)
Model 1  15025.19 (15332.71)  13512.65 (13973.93)  14311.58 (14772.85)
Model 2  12510.44 (12817.96)  11089.27 (11550.54)  11096.96 (11558.24)
Model 3  14770.14 (15077.66)  13351.96 (13813.24)   13177.3 (13638.58)
```


Model Comparison

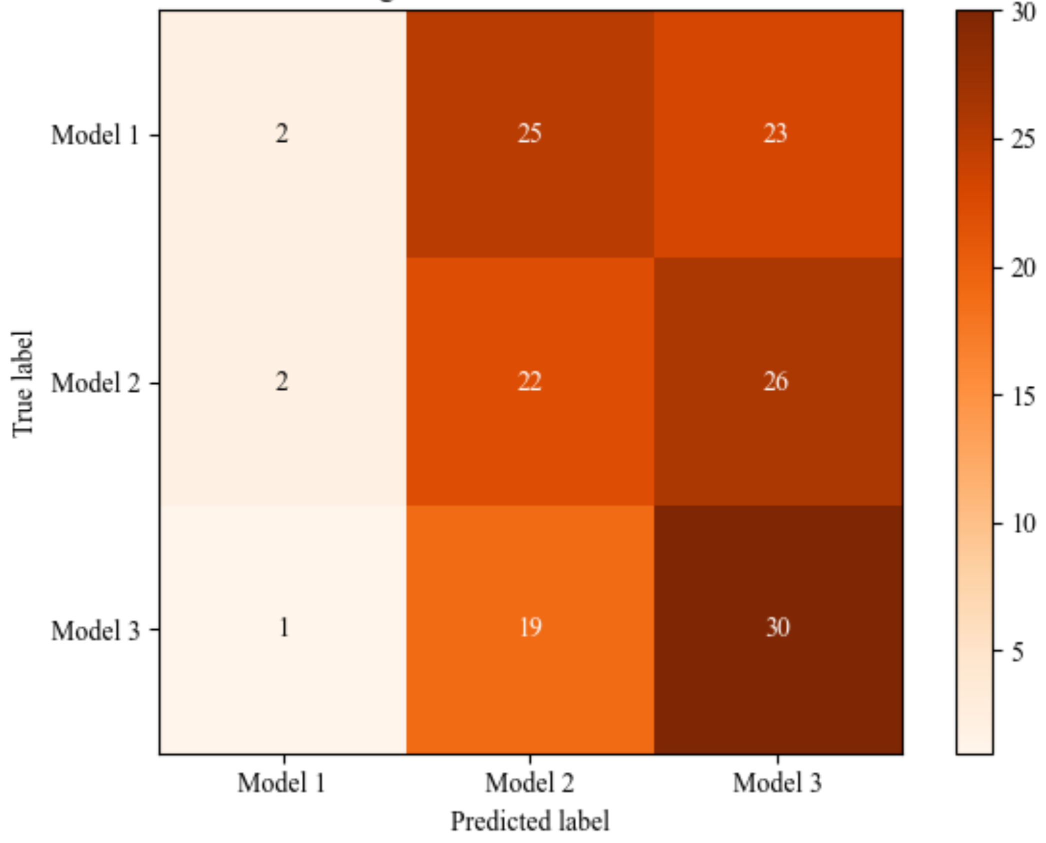
	Model 1 AIC (BIC)	Model 2 AIC (BIC)	Model 3 AIC (BIC)
Model 1	15025.19 (15332.71)	13512.65 (13973.93)	14311.58 (14772.85)
Model 2	12510.44 (12817.96)	11089.27 (11550.54)	11096.96 (11558.24)
Model 3	14770.14 (15077.66)	13351.96 (13813.24)	13177.3 (13638.58)

```python
[36]: true_labels = [0,1,2]*50

      predicted_labels = []
      for i in range(50):
          aic_m = [AIC_list_m[i], AIC_alt_list_m[i], AIC_ex_list_m[i]]
          aic_m1 = [AIC_list_m1[i], AIC_alt_list_m1[i], AIC_ex_list_m1[i]]
          aic_m2 = [AIC_list_m2[i], AIC_alt_list_m2[i], AIC_ex_list_m2[i]]

          predicted_labels.extend([np.argmin(aic_m), np.argmin(aic_m1), np.
       ↪argmin(aic_m2)])

      cm = confusion_matrix(true_labels, predicted_labels)
```

```python
print(cm)
confusion_matrix = np.array([[2, 25, 23], [2, 22, 26], [1, 19, 30]])

fig, ax = plt.subplots()
im = ax.imshow(confusion_matrix, interpolation='nearest', cmap=plt.cm.Oranges)
ax.figure.colorbar(im, ax=ax)

labels = ['Model 1', 'Model 2', 'Model 3']
ax.set(xticks=np.arange(confusion_matrix.shape[1]),
       yticks=np.arange(confusion_matrix.shape[0]),
       xticklabels=labels, yticklabels=labels,
       title="Figure 8: Confusion Matrix",
       ylabel='True label',
       xlabel='Predicted label')

thresh = confusion_matrix.max() / 2.
for i in range(confusion_matrix.shape[0]):
    for j in range(confusion_matrix.shape[1]):
        ax.text(j, i, format(confusion_matrix[i, j], 'd'),
                ha="center", va="center",
                color="white" if confusion_matrix[i, j] > thresh else "black")

fig.tight_layout()
plt.show()
```

```
[[ 2 25 23]
 [ 2 22 26]
 [ 1 19 30]]
```

Figure 8: Confusion Matrix


Model 1
Model 2
Model 3
True label
Model 1
Model 2
Model 3
Predicted label