Computational Cognitive Science Coursework 2

s1623705

March 24, 2020

1 Part 1: Drift Diffusion Process (23 points)

1.1 Task (a): Initial Simulations (9)

The Wiener diffusion process produces the following paths, using the provided parameters and a seed of 0 (which is used in later experiments too):

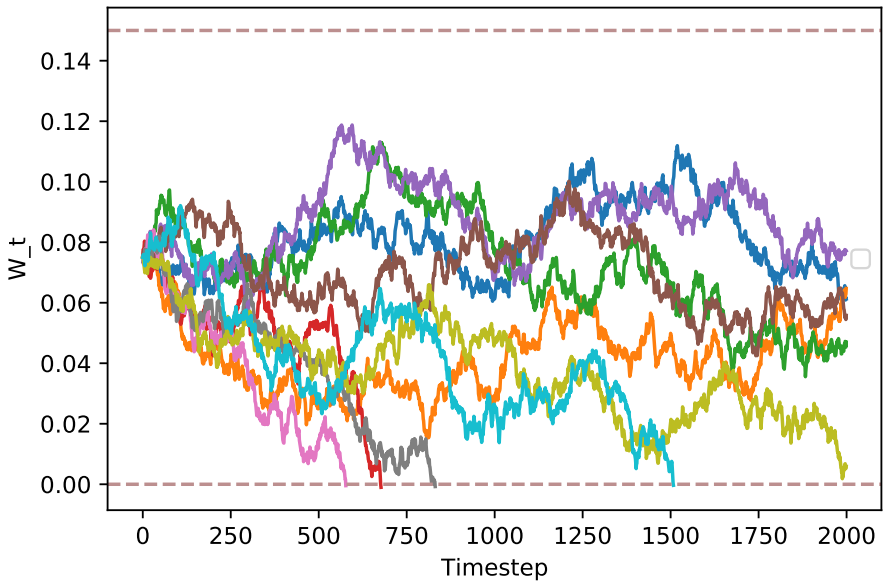


Figure 1: 10 paths produced by the Wiener diffusion process

The following histograms capture the distribution of response times. To build a better picture of this distribution, 1000 trials have been run:

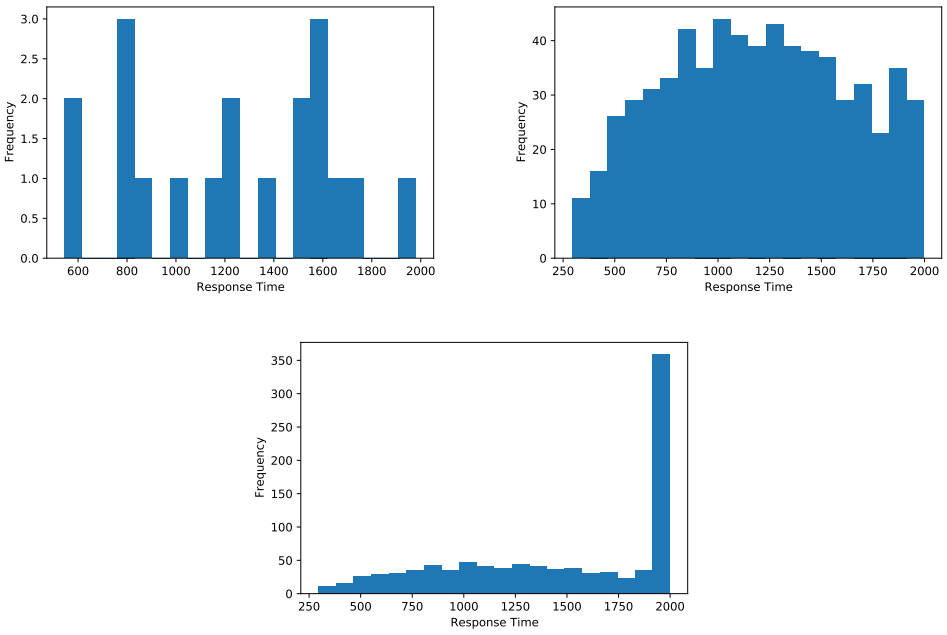


Figure 2: Positive (left), negative (right) and overall (bottom) response time histograms (1000 trials)

Correct	Incorrect	Timeout
1.9%	65.2%	32.9%

Table 1: Response constitution (1000 trials)

CAN SOMETHING BE INFERRED ABOUT THE EXPERIMENT?

1.2 Task (b): Exploring parameter settings (5)

For a fixed boundary separation of 0.15, drift rates close to 0 cause the accuracy to become close to 0.5, suggesting each new piece of evidence is uninformative. As the drift rate grows positively large, the accuracy tends to 1, whilst growing negatively large tends towards 0. The average response time also grows large as drift rate tends to 0. These results suggest that when the drift rate is informative, a hypothesis is reached both more often and more quickly.

For a fixed mean drift rate of -0.04, small boundary separations allows for a high accuracy, with a low average response time. As boundary separation grows, the accuracy tends to 0, whilst the average response time grows large. This suggests that if the threshold to accept a hypothesis is low, it will be accepted more often, and in a short amount of time.

There is also self-evident dependence between mean drift rate and boundary separation. If the starting position is always inbetween the two hypothesis boundaries, varying the size of mean drift rate and boundary separation together will not have an effect on the accuracy or average response time, as each drift will remain the same relative to the distance of the boundary from the starting point.

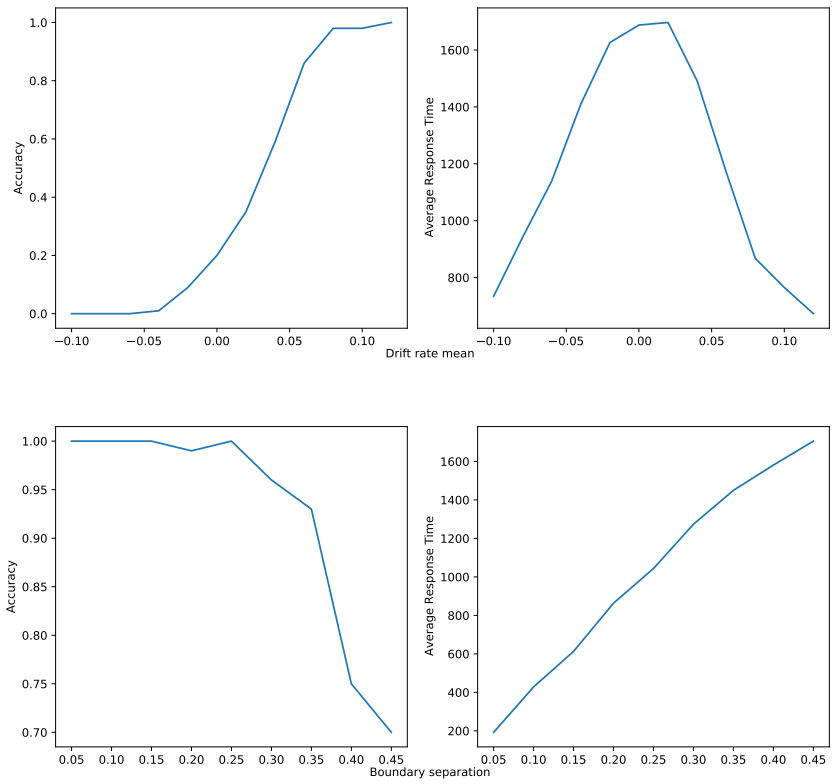
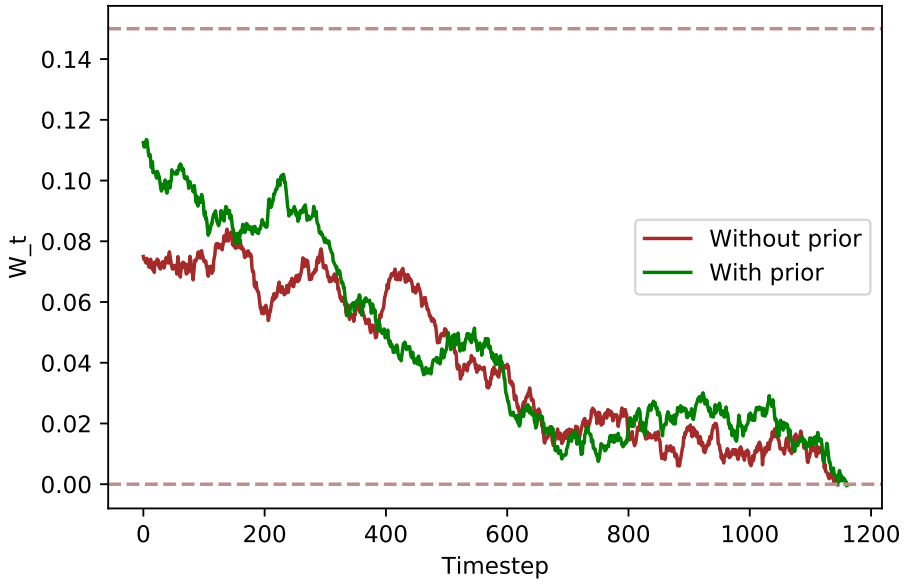


Figure 3: Effect of varying drift rate mean and boundary separation on accuracy and average response times

1.3 Task (c): Prior information (4)

To incorperate this prior information, we can simply place the starting point of our simulations closer to the positive hypothesis threshold than the negative, specifically, at 0.75 * boundary condition. This will lower the evidence requirements for accepting this hypothesis whilst increasing the evidence requirements for the negative hypothesis. Figure 1.3 illustrates the changing of the starting point, whilst Table 2 shows how the response constitution has changed. All other parameters have remained the same as those which produced the results in Table 1.



Correct	Incorrect	Timeout
1.36%	34.2%	52.2%

Table 2: Response constitution with priors incorperated (1000 trials)

1.4 Task (d): Group differences (5)

FILL OUT THIS ANSWER ASAP.

2 Part 2: Model fitting (77 points)

2.1 Task (a): Exploring the data (4)

Table 7 shows how many rewards each participant received during the trials, and the number of times each participant chose stimulus B over the trials.

If a participant were to choose randomly between the two stimuli in each trial, the expected number of rewards received would be 132. All participants performed better than this random expectation, and so we can deduce that learning took place to some extent during the trials. On the other hand, the top performing participant received 165 rewards during the trials, revealing that the range of rewards received is small over participants.

2.2 Task (b): Simulations (7)

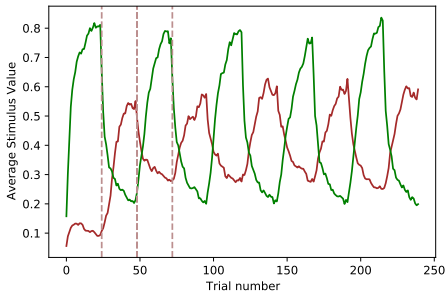


Figure 4: Average stimulus values over over 100 simulations

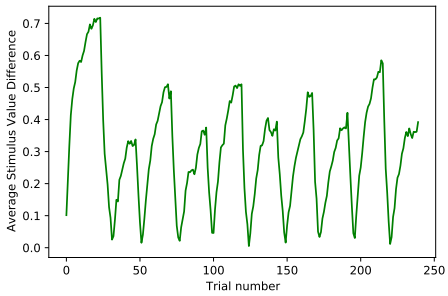


Figure 5: Average stimulus value difference over over 100 simulations

Figure 4 shows the stimulus values for for 240 trials, averaged over 100 simulations. It can be seen that the values tend towards the value of the underlying probability of reward. Vertical lines at trial 24 and 48 show the times at which the underlying reward distribution changed. This evolution makes sense as the reinforcement model begins to build a good reward likelihood for each stimulus, which changes every 24 trials, causing the model to have to approximate different probabilities.

Figure 5 shows the difference between the values of the stimulus throughout the trials. This can be interpreted as how likely the model is to choose one stimulus over the other. On the approach to a reward distribution change, the difference between values becomes larger, indicating more certainty of one stimulus over the other. Once the reward distribution changes, the difference between stimulus value diminishes to near 0.

2.3 Task (c): Exploring parameter settings (6)

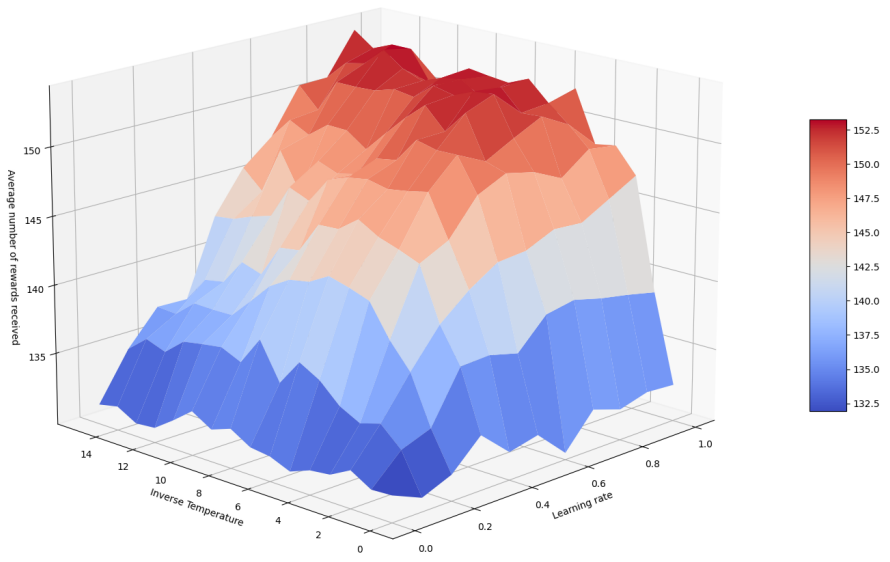


Figure 6: Effect of varying Learning Rate and Inverse Temperature on Average Rewards Received

Figure 6 shows the effect of varying the model parameters on the number of rewards received over all trials. The graph is colour coded by the z axis, average number of rewards received, to allow for a better interpretation. It shows that the learning rate effects the rewards received greatly. As the learning rate increases from 0, the average number of rewards increases with it, until it begins to plateau at around 0.6. The same goes for inverse temperature. As inverse temperature increases from 0 to 5, the average number of rewards received increases rapidly, then begins to plateau.

2.4 Task (d): Likelihood function (6)

The negative log likelihood for the second participant is 96.256. Listing 2 contains the function used to calculate this.

2.5 Task (e): Model fitting (7)

The following parameter optimizations were performed using the Nelder-Mead method. (Although the coursework specifies a gradient-based method must be used, this question was raised on piazza, and the Instructor response suggested Nelder-Mead was appropriate).

	Mean	Variance
Learning Rate	0.369	0.0154
Inverse Temperature	5.683	1.647

Table 3: Mean and variance of estimated parameters over all participants

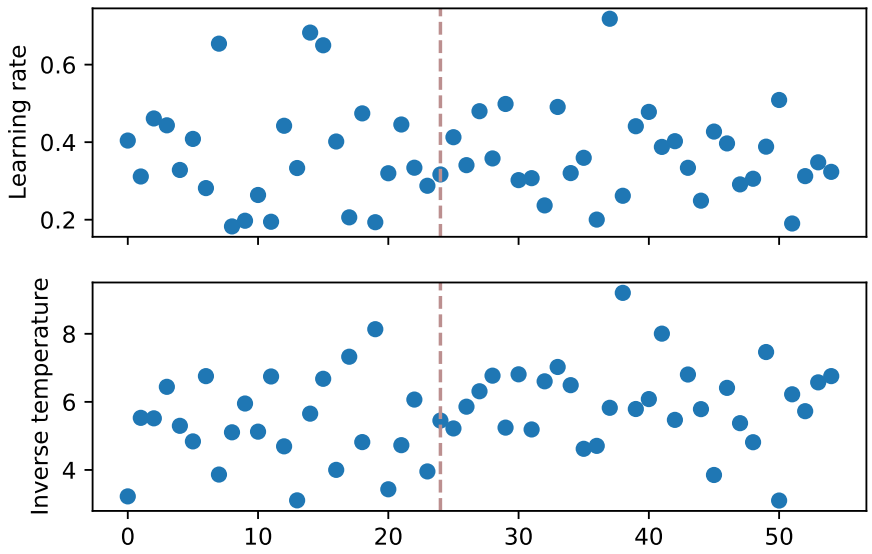


Figure 7: Optimized parameter fits for individual participants

When using Nelder-Mead, all the parameter values found in figure 7 are within a sane range, even with a non constrained optimization. This is not the case for BFGS optimization, which tended to produce large and unreasonable values. As such, this justified the switch from the recommended optimization strategy.

The dotted line in figure 7 shows the divide between the 24 MDD patients and the 31 healthy control participants. On visual inspection there does not seem to be a difference between these types of participant's optimal parameters.

Pearson's correlation coefficient between estimated parameters among all participants shows no particular significance, a value of -0.209 between the learning rate and inverse temperature. Further, the group specific correlations do not yield anything of significance, with Pearson correlation coefficients of -0.250 and -0.172 for the MDD and healthy participants respectively.

2.6 Task (f): Group Comparison (5)

As the assumption of equal variance cannot be assumed, Welch's t-test is suitable for ascertaining whether the MDD and control groups have significantly different estimated parameter distributions for the reinforcement model. For these samples, we have 53 degrees of freedom. Table 4 shows the results from this test. We can interpret these results by setting a value for what we can consider 'significant'. In this case 0.05 will be used. We observe that the p-value for the learning rate is above this, in which case we can accept the hypothesis that the two groups have similar distributions. In the case of inverse temperature, we observe a value of 0.049, which suggests that we should reject the hypothesis that the two groups have similar distributions.

	t-statistic	p-value
Learning Rate	0.100	0.921
Inverse Temp.	-2.020	0.049

Table 4: Welch's t-test between the two groups of participants (MDD and control)

2.7 Task (g): Parameter recovery (8)

In order to check the reliability of our parameter estimates we check whether we can run a simulation with fixed parameters, then perform the parameter estimation to recover the fixed parameters. Sampling these parameters from a known distribution gives us a better idea of how well parameters can be recovered. Table 3 shows the mean and variance of the parameter estimates of all participants, and so we can use these to build a distribution to sample from. Figure 8 illustrates this distribution.

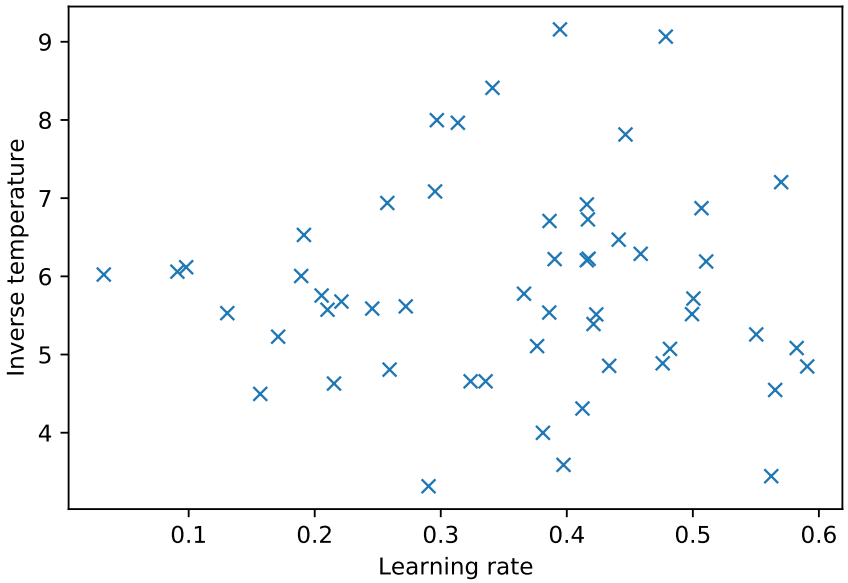


Figure 8: Multivariate gaussian of learning rate and inverse temperature; Modelled with mean and variance found in Table 3

We see high correlation between the parameters drawn from the sampling gaussian and the parameters recovered via model fitting. Table 5 shows the results. These high correlations are expected. An illustration of each correlation can be seen in Figure 9

	Correlation
Learning Rate	0.906
Inverse Temp.	0.795

Table 5: Pearson's correlation coefficients between fitted and simulated parameter values.

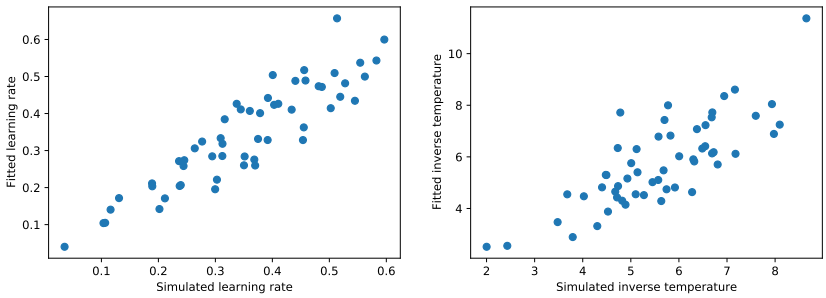


Figure 9: Fitted and simulated parameter values, showing strong correlation

The number of trials and number of simulations can be seen to affect the extent of correlation between the fitted and simulated parameter values. Figure 10 shows that and increase in both of these causes an increase in correlation. The trials were increased in batches of 48, still using the changes in reward probabililties every 24 trials.

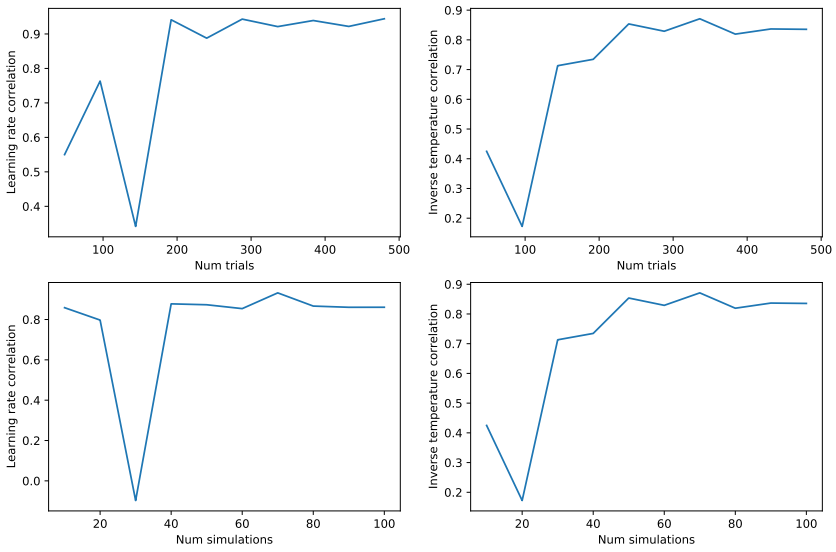


Figure 10: Varying number of trials and number of simulations, and observing effect on correlation between fitted and simulated parameters

2.8 Task (h): Alternative models (8)

Figures 11 and 12 show the fitted parameters of each participant, for models 2 and 3 respectively. The results show that model 2 and model 1 are equivalent (identical mean and covariance), yielding the same results. This is not surprising as both the inverse temperature and the reward sensitivity affect the degree to which a reward updates the expectations of the reinforcement model. The introduction of variable initial stimulus value does have an effect on the model. Performing Welch's independent t-test on the MDD and healthy patients model 3 fits shows that the reward sensitivities come from significantly different populations, whilst the other 3 parameters do not hold enough evidence to reject the hypothesis that they are drawn from the same population.

We can say that the inverse temperature has the same effect of reward sensitivity if the initial stimulus values are 0.

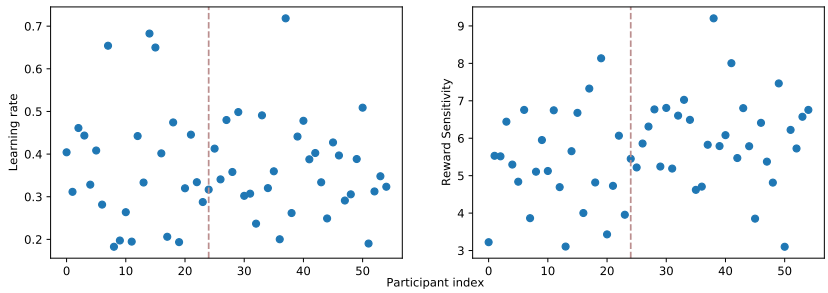


Figure 11: Model 2 fitted parameters for each individual

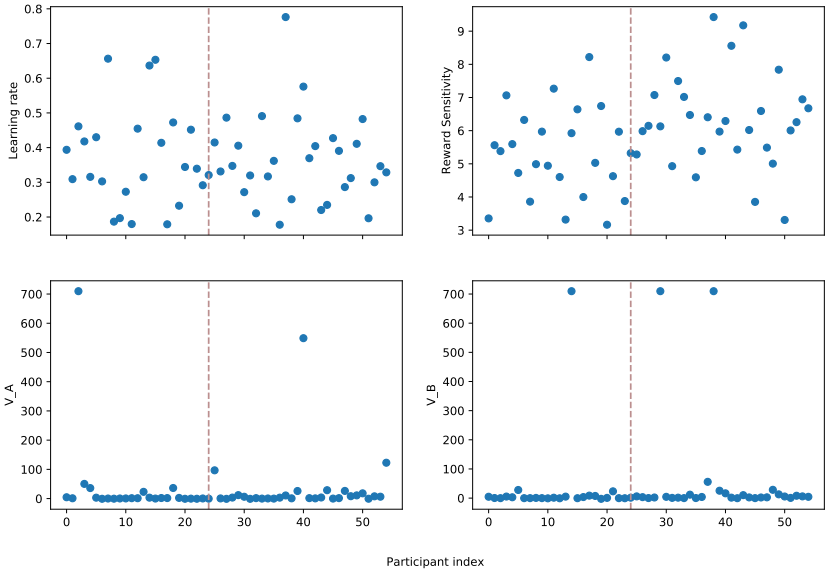


Figure 12: Model 3 fitted parameters for each individual

2.9 Task (i): Model comparison (6)

Given the discussion had in the previous section, Task (h), it makes sense that model 1 and 2 are equivalent in their total negative log likelihood and therefore in their AIC and BIC. Model 3 can be seen to have a lower AIC and BIC, and so can be regarded as the "best" model.

	NLL	AIC	BIC
Model 1	4653.841	9311.682	9318.644
Model 2	4653.841	9311.682	9318.644
Model 3	4557.487	9122.974	9136.897

Table 6: NLL, AIC and BIC for all models

2.10 Task (j): Model recovery and confusion matrix (6)

The following 2 confusion matrices show the goodness of fit for each model, when fitted to simulated data created with each of the models. It is intended to show how well the model's data can be recovered, which in turn suggests how reliable the comparison data in Task (h) is.

For each of the models, parameters were randomly sampled from Gaussians with means and variances disclosed in Table **??**

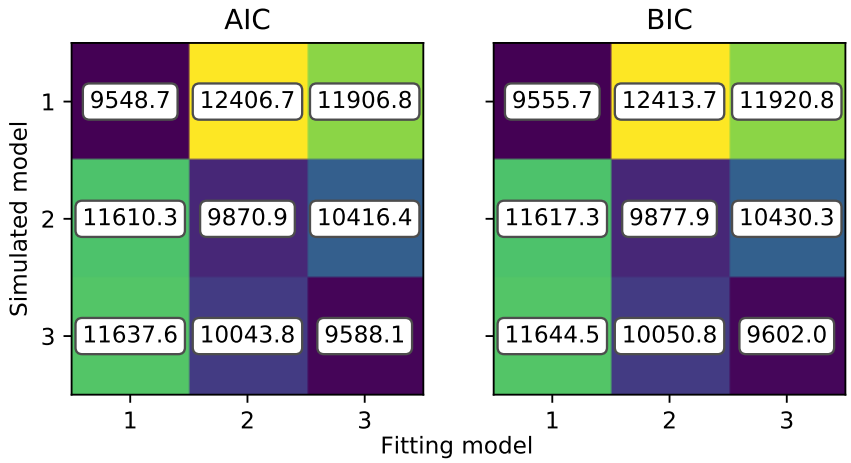


Figure 13: AIC and BIC confusion matrices illustrating performance of model fitting with data simulated by all other models

2.11 Task (k): Discussion (14)

Appendix A Python Code

Participant	Rewards Received	Times chosen Stimulus B
0	154	95
1	139	106
2	143	100
3	156	100
4	156	96
5	145	109
6	134	119
7	165	98
8	137	85
9	151	95
10	140	96
11	144	92
12	144	126
13	151	106
14	143	102
15	158	96
16	146	78
17	139	122
18	161	104
19	155	43
20	149	111
21	145	95
22	157	89
23	154	80
24	156	125
25	146	88
26	147	131
27	141	93
28	145	82
29	149	97
30	152	73
31	153	102
32	150	92
33	149	98
34	144	123
35	156	112
36	163	110
37	151	108
38	157	106
39	159	90
40	160	92
41	155	86
42	146	82
43	152	87
44	155	82
45	155	92
46	148	100
47	145	59
48	150	76
49	151	97
50	142	88
51	150	112
52	163	61
53	149	101
54	148	78

Table 7: Rewards received and number of times stimulus B was chosen over all trials for each participant

```python
import numpy as np


def wiener_diffusion(W_t_old, drift_rate_mean, drift_rate_variance, time_step):
    return (
        W_t_old
        + (drift_rate_mean * time_step)
        + (drift_rate_variance * float(np.random.normal(0, np.sqrt(time_step), 1)))  # type: ignore
    )


def simulate(
    time_step,
    starting_point,
    drift_rate_mean,
    drift_rate_variance,
    boundary_separation,
    max_steps=2000,
):

    W_t = []
    W_t.append(starting_point)

    for i in range(max_steps - 1):
        W_t.append(
            wiener_diffusion(
                W_t_old=W_t[i],
                drift_rate_mean=drift_rate_mean,
                drift_rate_variance=drift_rate_variance,
                time_step=time_step,
            )
        )
        if W_t[i + 1] < 0:
            return "h_neg", W_t
        elif W_t[i + 1] > boundary_separation:
            return "h_pos", W_t

    return "", W_t


def simulate_many(
    num_simulations,
    time_step,
    starting_point,
    drift_rate_mean,
    drift_rate_variance,
    boundary_separation,
    max_steps=2000,
):
    results = []
    for i in range(num_simulations):
        hypothesis, W_t = simulate(
            time_step=time_step,
            starting_point=starting_point,
            drift_rate_mean=drift_rate_mean,
            drift_rate_variance=drift_rate_variance,
            boundary_separation=boundary_separation,
            max_steps=max_steps,
        )

        results.append((hypothesis, W_t))

    return results
```

Listing 1: Part 1 main functionality

```python
import numpy as np
import pandas as pd
from typing import Tuple, Dict
from scipy.optimize import minimize


def get_new_stimulus_value(current_stimulus_value, learning_rate, reward_received: bool):
    return current_stimulus_value + (learning_rate * (int(reward_received) - current_stimulus_value))


def get_probability_of_stimulus_a(stimulus_values: Dict[int, float], inverse_temperature: float):
    return np.exp(inverse_temperature * stimulus_values[1]) \
           / (np.exp(inverse_temperature * stimulus_values[1]) + np.exp(inverse_temperature * stimulus_values[2]))


def simulate_single_trial(stimulus_values: Dict[int, float], reward_probabilities: Dict[int, float], learning_rate: float, inver
    prob_a = get_probability_of_stimulus_a(stimulus_values, inverse_temperature=inverse_temperature)
    choice = np.random.choice([1,2], p=(prob_a, 1-prob_a))

    reward_received = np.random.choice([True, False],
                                       p=(reward_probabilities[choice],
                                          1-reward_probabilities[choice]))

    stimulus_values[choice] = get_new_stimulus_value(stimulus_values[choice],
                                                     learning_rate=learning_rate,
                                                     reward_received=reward_received)

    return stimulus_values, choice, reward_received


def simulate_trials(learning_rate: float, inverse_temperature: float, num_48_trial_batches=5) -> Tuple[pd.DataFrame, np.array, n
    reward_probabilities_1 = {1: 0.4, 2: 0.85}
    reward_probabilities_2 = {1: 0.65, 2: 0.30}

    stimulus_values = {1: 0, 2: 0}

    all_choices = []
    all_reward_received = []
    all_stimulus_values = []
    for i in range(num_48_trial_batches):
        for _ in range(24):
            stimulus_values, choice, reward_received = simulate_single_trial(stimulus_values, reward_probabilities_1, learning_r
            all_choices.append(choice)
            all_reward_received.append(reward_received)
            all_stimulus_values.append(stimulus_values.copy())

        for _ in range(24):
            stimulus_values, choice, reward_received = simulate_single_trial(stimulus_values, reward_probabilities_2, learning_r
            all_choices.append(choice)
            all_reward_received.append(reward_received)
            all_stimulus_values.append(stimulus_values.copy())

    return pd.DataFrame(all_stimulus_values, columns=['A', 'B']), np.array(all_choices), np.array(all_reward_received)

def get_negative_log_likelihood(parameters: np.ndarray, choices: pd.Series, rewards_received: pd.Series):
    learning_rate = parameters[0]
    inverse_temperature = parameters[1]
    num_trials = choices.shape[0]
    V = [0,0]
    choice_probabilities = np.empty(shape=num_trials)
    for i in range(num_trials):
        choice_index = choices[i] - 1
        # choice_probabilities[i] = 1 / (1 + np.exp(-(V[choice_index] - V[not choice_index])))
        choice_probabilities[i] = np.exp(inverse_temperature * V[choice_index]) \
                    / (np.exp(inverse_temperature * V[choice_index]) + np.exp(inverse_temperature * V[int(not choice_index)]))
        V[choice_index] += (learning_rate * (rewards_received[i] - V[choice_index]))
    return -np.sum(np.log(choice_probabilities))


def get_total_negative_log_likelihood(parameters: pd.DataFrame, choices=pd.read_csv('data/choices.csv', header=None), rewards=pd
    return sum(get_negative_log_likelihood(parameters=parameters[i], choices=choices.iloc[i], rewards_received=rewards.iloc[i])
```



```python
def get_individual_parameter_estimates(choices, rewards, initial_parameters=np.array([0.5, 5])):
    optimal_params_list = []
    for i in range(choices.shape[0]):
```