2021 ■ Vol. 17 ■ no. 2

Computational modeling of behavioral tasks: An illustration on a classic reinforcement learning paradigm

Praveen Suthaharan [a], Philip R. Corlett [a] & Yuen-Siang Ang [b]

[a]Department of Psychiatry, Connecticut Mental Health Center, Yale University
[b]Institute of High Performance Computing, A*STAR

Abstract ■ There has been a growing interest among psychologists, psychiatrists and neuroscientists in applying computational modeling to behavioral data to understand animal and human behavior. Such approaches can be daunting for those without experience. This paper presents a step-by-step tutorial to conduct parameter estimation in R via three techniques: Maximum Likelihood Estimation (MLE), Maximum A Posteriori (MAP) and Expectation-Maximization with Laplace approximation (EML). We first demonstrate how to simulate a classic reinforcement learning paradigm – the two-armed bandit task – for N = 100 subjects; and then explain how to develop the computational model and implement the MLE, MAP and EML methods to recover the parameters. By presenting a sufficiently detailed walkthrough on a familiar behavioral task, we hope this tutorial could benefit readers interested in applying parameter estimation methods in their own research.

Keywords ■ Computational modeling, reinforcement learning, two-armed bandit, parameter estimation, maximum likelihood estimation, maximum a posteriori, expectation-maximization. **Tools** ■ R.

✉ praveen.suthaharan@yale.edu

10.20982/tqmp.17.2.p105

Acting Editor ■ Denis Cousineau (Université d'Ottawa)

Reviewers
■ One anonymous reviewer

Introduction

Cognitive scientists typically have a theory from which hypotheses are generated. Often, these hypotheses are couched in terms of behavioral differences between experimental conditions and/or groups. To make sense of behavioral data, two broad categories of statistical tools are normally used. Descriptive statistics summarize properties of the data through measures of central tendency and dispersion, while inferential statistics allow a specific hypothesis to be tested via methods such as analysis of variance. However, these tools are increasingly recognized as insufficient to help us understand individual variability in human behavior (Bakeman & Robinson, 2005; Pagano, 2012). Computational modeling of data has emerged as a popular way to investigate mechanisms underlying human thoughts and behavior (Montague, Dolan, Friston, & Dayan, 2012; Corlett & Fletcher, 2014; Wang & Krystal, 2014; Stephan & Mathys, 2014; Huys, Maia, & Frank, 2016; Patzelt, Hartley, & Gershman, 2018; Huys, Browning, Paulus, & Frank, 2020). Moreover, applying computational models to behavioral data may even garner insights into neural signals and brain function (Niv, 2020).

Computational Modeling of Behavioral Data

Computational modeling. As defined by the National Institute of Biomedical Imaging and Bioengineering, computational modeling is the use of computers to simulate and study complex systems using mathematics, physics and computer science. In behavioral science, many facets of computational modeling need to be considered for effective modeling of behavior (Wilson & Collins, 2019). This has led to new insights on various processes including reward learning (Daw, 2011; Huys, Pizzagalli, Bogdan, & Dayan, 2013; Momennejad et al., 2017), impulsivity (Bickel, Odum, & Madden, 1999), motivation (Lockwood et al., 2017), decision-making (Ang et al., 2018), belief-updating (Reed et al., 2020) and cognitive control (Dillon et al., 2015).

Typically, the goal of computational modeling of behavioral data is to derive subject-specific estimates of parameters, or variables that index specific components of the cognitive process of interest, that explain behavior.

Behavioral data. Behavioral data are complex; they capture how agents (e.g., animals, humans, or robots) interact with their environment – commonly in terms of the actions performed (by the agent) and the rewards harvested (from the environment). Here, we examine a classic behavioral paradigm, the N-armed bandit task (Slivkins, 2019), where an individual is faced with N=2 slot machines. One machine has a higher win probability. The agent needs to learn which machine to choose to reap maximum reward. This learning – accumulated by interacting with the machines – is typically modeled using reinforcement learning (RL) theory (Sutton & Barto, 2018). In this paper, we focus on a relatively simple RL model: the Q-learning model (Watkins, 1989).

The remaining sections of this paper are organized as follows. *Tutorial* describes the behavioral task, computational model, and likelihood function. *Tutorial – Part 1* provides a walkthrough of the R code on how to simulate behavioral task data using the Q-learning model. *Tutorial – Part 2* provides a walkthrough on visualizing the behavioral data using the powerful ggplot2 tool. *Tutorial – Part 3* provides a walkthrough on how to implement three estimation methods – Maximum Likelihood Estimation (MLE; Myung, 2003), Maximum A Posteriori (MAP; Cousineau & Hélie, 2013) and Expectation-Maximization (Do & Batzoglou, 2008) with Laplace approximation (EML; Huys et al., 2011) – to recover decision-making behavior from task performance. *Discussion* provides a brief summary of the results of model fitting on the two-armed bandit task, a discussion on the importance of model fitting for elucidating behavioral differences, and limitations. Taken together, the tutorial is intended for readers to grasp a basic and intuitive understanding of how to implement a reinforcement learning model to a two-choice behavioral task to generate behavior and then use parameter estimation techniques to recover behavior.

Task and Model

Two-armed bandit task

We begin with a brief vignette of the two-armed bandit task (see Figure 1).

> Imagine walking into a new casino with only two slot machines available. Unbeknownst to patrons, both machines follow a Gaussian payout distribution; one pays with $(\mu, \sigma) = (1, 2)$ while the other $(\mu, \sigma) = (-1, 2)$. In other words, there is a higher chance of winning money on one machine and a higher chance of losing on the other. To drum up publicity, the owner says that players do not have to pay anything upfront. Instead, you are given $N = 200$ trials to pull any machine you like and the total outcome will be settled at the end. Thus, one has to learn as quickly as possible which machine has the better odds of payoff in order to maximize earnings.

Computational model

How might people learn which slot machine is better over time? Humans adopt algorithmic strategies of reinforcement learning when performing tasks like the bandit task (Schulz & Gershman, 2019). The Q-learning model iteratively updates values attributed to actions in order to improve the learning behavior of an individual (Watkins & Dayan, 1992). This model comprises two parts: Q-learning update rule and Softmax decision rule.

Q-learning update rule. Let t be the trial number, a_t be a participant's action on a trial, A_t and B_t be the specific action of choosing machine A or machine B, respectively, and r_t be the reward obtained by the participant. The Q-learning model posits that, on each trial, the subject assigns an expected value Q_{A_t} and Q_{B_t} to each machine. Assuming no initial bias for either machine, these two variables are set to zero initially:

$$Q_{A_t} = 0, \quad Q_{B_t} = 0$$

where $t = 0$. On every trial t, the participant selects a machine and receives the outcome r_t. The difference between the outcome obtained and the reward expected is quantified by the reward prediction error (RPE):

$$\delta_t = r_t - Q_{a_t} \tag{1}$$

where Q_{a_t} can either be Q_{A_t} or Q_{B_t} depending on the choice of machine at a particular trial. The RPE on that trial is then used to update the value of the chosen machine according to the following equation:

$$Q_{a_{t+1}} = Q_{a_t} + \alpha\delta_t \tag{2}$$

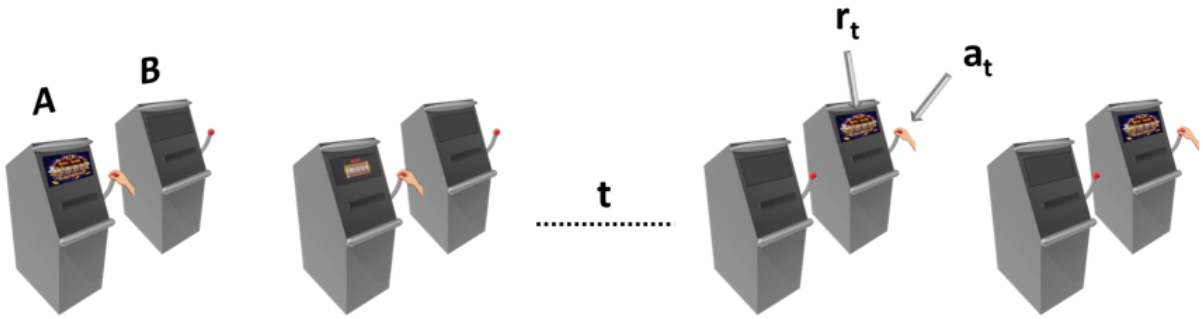
in which α refers to learning rate ($0 \leq \alpha \leq 1$) and acts to scale the impact of δ_t when updating the expected value. In other words, the higher the α value, the faster a participant learns which machine gives a better payout.

Softmax decision rule. The expected values of each lever are then converted to probabilities of choosing via the Softmax equation (Luce, 1986):

$$P(A_t) = P(A_t|\beta Q_{A_t}, \beta Q_{B_t}) = \frac{e^{\beta Q_{A_t}}}{e^{\beta Q_{A_t}} + e^{\beta Q_{B_t}}} \tag{3}$$

$$P(B_t) = P(B_t|\beta Q_{A_t}, \beta Q_{B_t}) = \frac{e^{\beta Q_{B_t}}}{e^{\beta Q_{A_t}} + e^{\beta Q_{B_t}}} \tag{4}$$

Figure 1 ■ Two-armed bandit task. A classic reinforcement learning paradigm used to capture decision-making behavior. The participant is presented with two slot machines, A and B, with different payoff distributions. On every trial t, an individual chooses a lever a_t and is presented with a reward r_t. To maximize winnings, one must learn which machine has the better odds of payoff over the course of the task.



where equations 3 and 4 represent the probabilities of choosing machines A and B, respectively. In reinforcement learning, a decision to be made – for example, choosing a particular machine – requires one to either explore other options or exploit the current choice (Daw, O'doherty, Dayan, Seymour, & Dolan, 2006). This exploration-exploitation mechanism is measured by β, which refers to choice randomness (or more technically, inverse temperature parameter) and represents how random actions are ($0 \leq \beta < \infty$); lower values cause actions to be more equiprobable (i.e., an individual is still exploring other choices across trials), whereas higher values denote more deterministic actions (i.e., an individual sticks with a specific choice across trials).

Likelihood function

The observed choice data can now be used to estimate the behavioral parameters. We fit the Q-learning model to the two-armed bandit task data, generating the likelihood function. Mathematically, after taking the logarithmic of the likelihood function (Etz, 2018), the log-likelihood for choosing machine A on a particular trial t (LL_{A_t}) and for choosing machine B on a particular trial t (LL_{B_t}) can be written as (for derivation refer to Appendix D):

$$LL_{A_t} = log\left(P\left(A_t\right)\right) = Q_1 - \log\left(e^{Q_1} + e^{Q_2}\right) \quad (5)$$

$$LL_{B_t} = log\left(P\left(B_t\right)\right) = Q_2 - \log\left(e^{Q_1} + e^{Q_2}\right), \quad (6)$$

where $Q_1 = \beta Q_{A_t}$ and $Q_2 = \beta Q_{B_t}$. The goal of parameter estimation is to maximize (in other words, to take the sum of the LL for each trial) so as to find the parameters that most likely generated the choice data (see Figure 2). Although the goal of parameter estimation is to maximize the log-likelihood, we want to, in terms of optimization, think of the log-likelihood function as a 'cost' function (other interchangeable names are 'loss' or 'error' function). Thus, as you can imagine, we would like to minimize the negative log-likelihood. Moreover, by default, the optimization algorithm we use minimizes.

Tutorial

Getting Started

Installation of R. The simulation, visualization and estimation procedures of this tutorial are all implemented inside the freely-available R/RStudio environment (Venables, Smith, & R Development Core Team, 2009; Racine, 2012). Therefore, the user needs to install the latest versions of R (https://www.r-project.org/) and RStudio (https://rstudio.com/products/rstudio/) to successfully execute the lines of code presented in this tutorial. Note R version 4.0.0 and RStudio version 1.1.447 were used at the time of publication.

Installation of twochoiceRL. The twochoiceRL package can be installed from GitHub (see https://github.com/psuthaharan/twochoiceRL) by running the following command in the RStudio console:

```
devtools::install_github("psuthaharan/
    twochoiceRL")
```

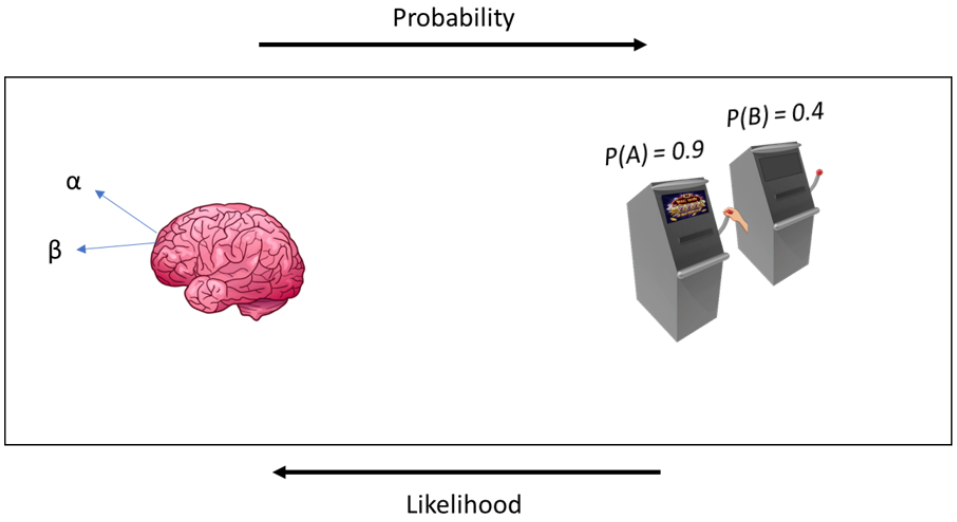
This package can then be activated by running:

```
library(twochoiceRL)
```

It consists of three main functions:

```
?simulate_twochoiceRL
?plot_twochoiceRL
?estimate_twochoiceRL
```

Figure 2 ■ Likelihood as an inversion of probability. We intuitively define likelihood as representing the estimation of behavior given a decision. In the Probability scenario, your brain executes a learning model to generate a decision – choose machine A or machine B. In the Likelihood scenario, you have observed the data (i.e., you know an individual chose machine A) and now consider what behavioral parameters (i.e., α and β) generated an individual's decision to choose machine A. The likelihood function allows us to search for values of the behavioral parameters that most likely directed an individual to make a particular decision.



Note that if you add a `?` before the function name in the console, you will be able to view a detailed description of each function. The `simulate_twochoiceRL` provides the functionality for simulating the two-armed bandit task. The `plot_twochoiceRL` provides the functionality for plotting the behavior of individuals who completed the simulated two-armed bandit task. The `estimate_twochoiceRL` provides the functionality for recovering the behavioral parameters used to generate the simulated behavior. The following sections will go in more detail into these individual functions (see `README` on *GitHub* for a quick walkthrough).

Part 1: Simulating data (`simulate_twochoiceRL.R`)

Overview

We have introduced the idea that fitting the computational model to the task data generates a likelihood function which we can use to estimate parameters. The Q-learning model introduces two behavioral parameters that shape an individual's underlying decision-making process: α represents how quickly an individual learns of the better choice and β represents randomness of an individual's action. In the following section, let's assume we have 100 hypothetical subjects with different α and β values. We will demonstrate how to simulate their behavior in the two-armed bandit task through the Q-learning model and Softmax equation. The main function we focus on in Part 1 is `simulate_twochoiceRL` (see Appendix A, Listing 1). This function takes in several input parameters and executes a few procedures to generate a simulated dataset of individuals performing the two-armed bandit task. For users interested in copying lines 1-100 in Listing 1, please remember to have the `foreach` library installed.

Procedures

Simulate parameters. When running simulations, it is important to set a seed value (see Listing 1, line 14-15). A seed value is any random number that holds data for replicating results. The function allows the user to specify any integer value for `seed_value` (defaults to 528).

The simulation begins with a random selection of values for each participant's choice randomness and learning rate by drawing from a Gaussian distribution (see Listing 1, lines 17-19). We create a variable, `n_subj`, that represents the number of subjects (defaults to 100) that we want to simulate. The rnorm function is then used to generate `n_subj` data points from a normal distribution with $(\mu_1, \sigma_1) = (\text{mean}_x^1, \text{stdev}_x^1)$ for x_1 and $(\mu_2, \sigma_2) = (\text{mean}_x^2, \text{stdev}_x^2)$ for x_2. You will notice that x_1 and x_2 are used as inputs for β and α, respectively (see Listing 1, lines 21-23). We call this re-parameterization whereby

we introduce another variable (e.g., x_1) as input to a function (e.g., the exponential function: e^x) such that it preserves the bounds of the original variable (e.g., β) while allowing us to leverage the unbounded nature of the re-parameterized variable (e.g., x_1). To be more specific, as we have mentioned in the Q-learning model, β is known to be bounded (i.e., constrained) between $[0, \infty)$ and α bounded between [0,1]. Moreover, the process of parameter estimation begins with random restarts (or starting values). These values are fed into the optimization algorithm to output the estimate (i.e., MAP) and the respective Hessian matrix. In the EML procedure, specifically, these estimates and the Laplace values (diagonal elements (or trace) of the Hessian matrix) are used to compute the posterior hyperparameters (or mean and standard deviation of the distribution that we are sampling from for the model parameter estimates) on a particular iteration. The succeeding iterations sample the new random restarts from the previous estimated distribution. If the parameters are bounded, you can imagine that there is a greater possibility for the variance of the estimated distribution to overlap the previously converged estimate, thus not allowing the algorithm to robustly search the entire parameter space. We minimize the possibility of this issue by using an unconstrained optimization technique for parameter estimation, which requires our parameters to be unbounded between $(-\infty, +\infty)$. We remove the constraints by re-parameterizing the model parameters in terms of x_1 and x_2 using the following equations:

$$\beta = e^{x_1} \tag{7}$$

$$\alpha = \frac{1}{1 + e^{-x_2}} \tag{8}$$

We use the exponential function (equation 7) and sigmoid function (equation 8) to transform β and α, respectively, and perform our unconstrained optimization on x_1 and x_2 from $(-\infty, +\infty)$.

Initialize Q-learning vector. Now that we have demonstrated how to simulate the behavioral parameters for `n_subj` subjects, we want to use these parameters to generate simulated choices. To do this, we must first initialize our expected values of choosing machines A and B to 0 (assuming no initial bias for either machine). We create a row vector Q – updated over trials as an individual learns – to hold these values (see Listing 1, line 33-34).

Define softmax function. The probability of an individual choosing a machine, as discussed previously, is computed by the softmax choice model. We code the function based on equations 3 and 4, where the input x is a row vector of the expected values for machine A and machine B. We let $y = x - \max(x)$ to avoid numerical overflow (see Appendix C on how this definition is equivalent to the softmax equations introduced earlier).

```
# Softmax choice function
softmax <- function (x) {
  y <- x - max(x)
  exp(y)/sum(exp(y))
}
```

Simulate data. We have now established three important steps for simulating our two-armed bandit dataset – (1) simulate parameters, (2) initialize Q-learning vector and (3) define the softmax equation. These three procedures – along with the Q-model – are the foundations for simulating our trial-by-trial data. Lines 36-77 in Listing 1 simulates an individual's choice behavior across `n_tr` trials (defaults to 200). In the bandit paradigm, we know that the slot machines each has its own payoff distribution. Specifically, we have earlier assumed that machines A and B follow a standard Gaussian-like payoff distribution $N(\mu = 1, \sigma = 2)$ and $N(\mu = -1, \sigma = 2)$, respectively (see Listing 1, lines 40-44). On each trial, we will first generate the probabilities of an individual choosing each machine using the softmax equation, incorporating the individual's β and expected values Q (see Listing 1, lines 46-47). Following that, either machine A or B will be sampled based on these action probability values (see Listing 1, lines 49-50). A reward will then be given (see Listing 1, lines 52-53). Based on the individual's action, reward and learning rate (α), we will update the expected value using the Q-learning model (see Equation 2) for the next trial (see Listing 1, lines 55-56). We perform this sequence for each `n_tr` trial and save the trial data into a data frame (see Listing 1, lines 58-64). This trial-by-trial data is stored for each `n_subj` subjects in `twochoiceRL_data` (initialized in Listing 1, lines 11-12 and saved in Listing 1, lines 77-78). Note that the `simulate_twochoiceRL` function has an additional parameter, `trials_unique`. This parameter serves to subset the data for estimation purposes. If TRUE (by default), only unique trials (i.e., trials where the individual selected a machine) is kept (see Listing 1, lines 66-68). Otherwise, all trials are kept for plotting purposes (see Listing 1, lines 70-72).

The function `simulate_twochoiceRL` outputs a list containing the trial-by-trial task data, the number of subjects the user specified to simulate, and the true parameter values for x_1 and x_2 that were used to generate the behavior (see Listing 1, lines 93-98).

How do we view the resulting behavioral data for an individual? First, we should store the function output to a variable:

```
# Save simulated data to a variable
data <- simulate_twochoiceRL(
            seed_value = 528,
```

```
    n_subj = 100,
    n_tr = 200,
    mean_x = c(1,0),
    stdev_x = c(1,1),
    mean_payoff = c(1,-1),
    stdev_payoff = c(2,2),
    trials_unique = TRUE,
    progress_bar = TRUE)
```

Now we can view the data for individual 100 (see Figure 3):

```
# View individual 100
View(data$twochoiceRL[[100]])
```

Part 2: Plotting data (`plot_twochoiceRL.R`)

Overview

Viewing data values is good. But, visualizing data is more appealing. We will demonstrate how to plot the behavioral data of individuals using the ggplot2 tool (Wickham, 2016). The main function we focus on in Part 2 is `plot_twochoiceRL` (see Appendix A, Listing 2). This function requires four input parameters – the simulated dataset, the subject data to plot, line colors representing the two choices, and whether a user would like to view it in static form or animated form – to generate a plot of the individual's behavior. For users interested in copying lines 1-68 in Listing 2, please remember to have both the `ggplot2` and `gganimate` libraries installed.

Procedures

Visualizing data. We visualize the change in expected values and choice probabilities across trials for an individual. It's important to set `trial_unique` to FALSE because for plotting purposes we are interested to see the data values for the relevant trials (where the individual selected the machine) and the counterfactual trials (where the individual did not select the machine).

The following lines of code produces Figure 4:

```
# Saving simulated data for plotting
data <- simulate_twochoiceRL(
     trials_unique = FALSE)

# Visualize behavior of subject 100
plot_twochoiceRL(data = data,
     subj = 100,
     colors = c("#009999","#0000FF"),
     plot_type = "static")
```

Part 3: Parameter estimation (`estimate_twochoiceRL.R`)

Overview

Up until now, we have shown how to use simulated behavioral parameters to generate choice data. However, in reality, we will only have participants' choice data to study behavior. We will demonstrate how to recover (or estimate) the behavioral parameters from the choice data using three different parameter estimation techniques. The main function we focus on in Part 3 is `estimate_twochoiceRL` (see Appendix A, Listing 3). It utilizes three separate functions for each estimation method: `mle_twochoiceRL`, `map_twochoiceRL`, and `eml_twochoiceRL`. For brevity, we will unpack the main function but leave the details of these individual estimation scripts for the users to peruse through in the package.

Procedures

Performing MLE. In practice, the true behavioral parameters are not known. Hence, we will start the parameter estimation methods with random guesses for our parameters (see Listing 3, lines 19-20). We create a `mle_twochoiceRL` function that takes as input the simulated data (see Listing 3, line 25), a list of random guesses for our parameters (see Listing 3, line 26), an objective function that calculates the negative log-likelihood, gradient and Hessian (see Listing 3, line 27), an optimization algorithm (Geyer, 2020) used to minimize the objective function (see Listing 3, line 28), the starting and maximum allowed trust region radius (see Listing 3, line 29), and a user-specified number of random restarts (or starting parameter values) to search our parameter space (see Listing 3, line 30). The execution of this function results in the MLE estimates for each parameter (see Listing 3, lines 33-35). We save these parameter estimates along with the true parameter values in a data frame (see Listing 3, lines 37-44) for plotting (see Listing 3, lines 46-53).

To recapitulate, we simulated data in Part 1 and observed which choices (or slot machines) each of our simulated subjects preferred in Part 2. In Part 3, we are focused on recovering their underlying decision-making process by estimating how random their choices are ($x_1; \beta$) and how quickly they learn of the better choice ($x_2; \alpha$). The MLE approach takes initial random guesses of these behavioral parameters and outputs estimated behavioral parameters that the algorithm believes is most probable to have resulted in the simulated subject for choosing a particular machine.

Performing MAP. Let's now consider the case where you not only take initial random guesses of the behavioral parameters but also have some previous information (i.e.,

Figure 3 ■ Simulated two-armed bandit data from a single individual. We examine the first and last 10 trials of subject 100's casino performance to illustrate the underlying decision-making process. The data contains 5 columns – the trial number, the expected value of the chosen machine, the probability of choosing the machine, the participant's chosen machine, and the outcome of the choice. We can see that over time subject 100 learns to prefer machine A (i.e., Action = 1) as evident by the increase in probability.

First 10 trials

Trial	Value	Pr	Action	Reward
1	0.011472946	0.50000000	1	0.07437611
2	0.150651408	0.50248291	1	0.91373052
3	-0.214775990	0.46744276	2	-1.39233659
4	0.028651131	0.57843137	1	-0.64024449
5	-0.001389443	0.55248744	1	-0.16609406
6	0.037147215	0.54604939	1	0.24843364
7	-0.761090403	0.44569485	2	-3.75638950
8	0.146952310	0.66619058	1	0.74898494
9	-0.210501591	0.31301532	2	2.80823291
10	0.051462977	0.57674733	1	-0.47208007

Last 10 trials

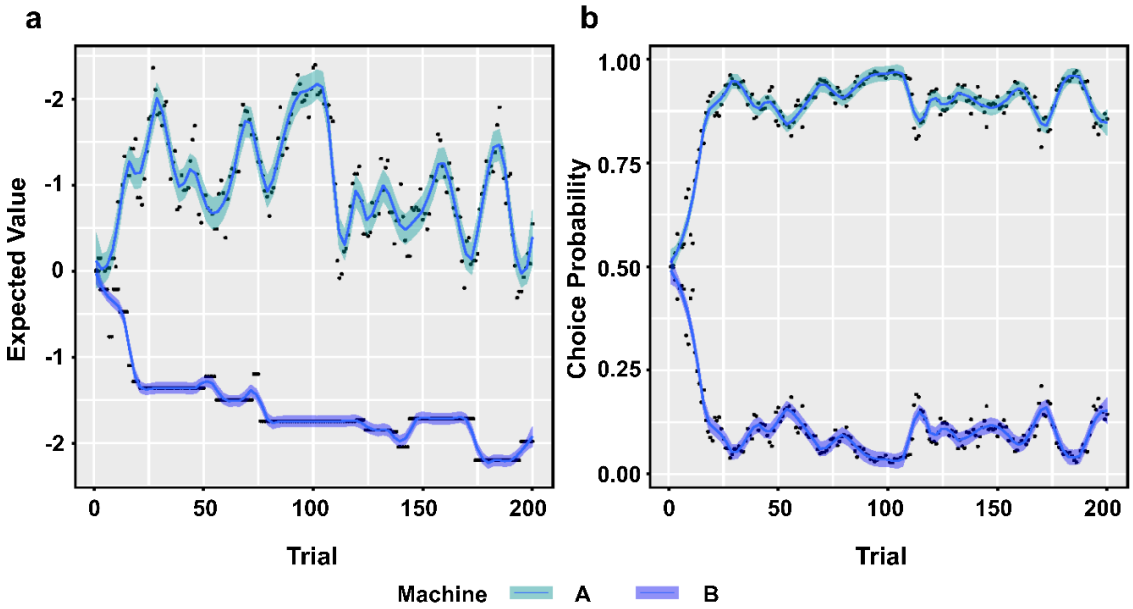
Trial	Value	Pr	Action	Reward
191	0.42397678	0.94693953	1	-3.45783195
192	0.06407611	0.90626909	1	-1.90916516
193	-0.30864166	0.87624644	1	-2.35215577
194	-0.23582758	0.83681341	1	0.16339302
195	-1.97735914	0.15476074	2	-0.77297922
196	-0.07109336	0.81870245	1	0.83210126
197	0.03418583	0.83891613	1	0.61140414
198	0.20643864	0.85085490	1	1.15085577
199	0.08734256	0.86880512	1	-0.56563016
200	0.54946368	0.85660058	1	3.08315301

prior knowledge) of an individual's decision-making process; for example, say we know an individual is a decisive thinker ($\beta > 10$) and learner ($\alpha > 0.8$). With this information (i.e., an informative prior) we can provide a range for our parameters and take initial random guesses from this range, yielding more accurate estimates. We initialize an assumed prior distribution (i.e., a relatively wide normal distribution) for the parameters (see Listing 3, lines 162-166). Therefore, in addition to the input parameters used in the MLE function, the `map_twochoiceRL` function requires prior information (see Listing 3, lines 178-179).

Performing EML. Like the MAP method, we begin by initializing priors for x_1 and x_2, but, unique to EML, also variables for storing information on convergence and iterations of the algorithm (see Listing 3, lines 301-304) where it iteratively estimates the distribution for each parameter estimates. The EML procedure (see Listing 3, lines 309-390) is a while loop that begins with random guesses of our model parameter values for the first iteration (see Listing 3, lines 312-315) and uses the previous MAP estimates as initial guesses for successive iterations (see Listing 3, lines 335-337). The procedure currently loops until the absolute difference between the previous LL and current LL is less than 0.001 (see Listing 3, line 310); this can be adjusted to allow the algorithm to iterate through faster (diff $>> 0.001$) or slower (diff $<< 0.001$). In addition to the input parameters used in the MAP function, the `eml_twochoiceRL` function requires knowledge of the iteration number (see Listing 3, line 354) and the MAP estimates from previous iterations (see Listing 3, line 355). Notably, the EML procedure differs from the other methods in that the priors are iteratively updated using the calculated MAP and the Laplacian values (see Listing 3, lines 376-385) based on the Laplacian approximation method (Huys et al., 2011). These iteratively updated, Laplace-approximated posterior hyperparameters are returned by the EML procedure (see Listing 3, line 421). We graphically illustrate the convergence of these hyperparameters (i.e., the mean and standard deviation of the distribution from which we are sampling our behavioral parameters) per EML iteration (see Appendix B). It's interesting that, in many cases, the posterior hyperparameters converge around the mean values of our simulated parameters. For those other cases, it's plausible that the parameter search of the algorithm found a local minimum. Therefore, a couple of suggestions would be to increase the maximum trust region radius (`tr_rad`) or the total number of restarts (`nRes`) in hopes of finding the global minimum.

The estimation performance of our three methods – MLE, MAP and EML – is summarized in Figure 5. The following lines of code will produce the estimation plots for each technique, respectively:

Figure 4 ■ Learning performance of simulated subject 100. The expected value and choice probability suggest this subject clearly prefers lever A as evident by a higher probability. (a) Expected value for lever A and B across 200 trials. (b) Probability of choosing lever A and B across 200 trials.

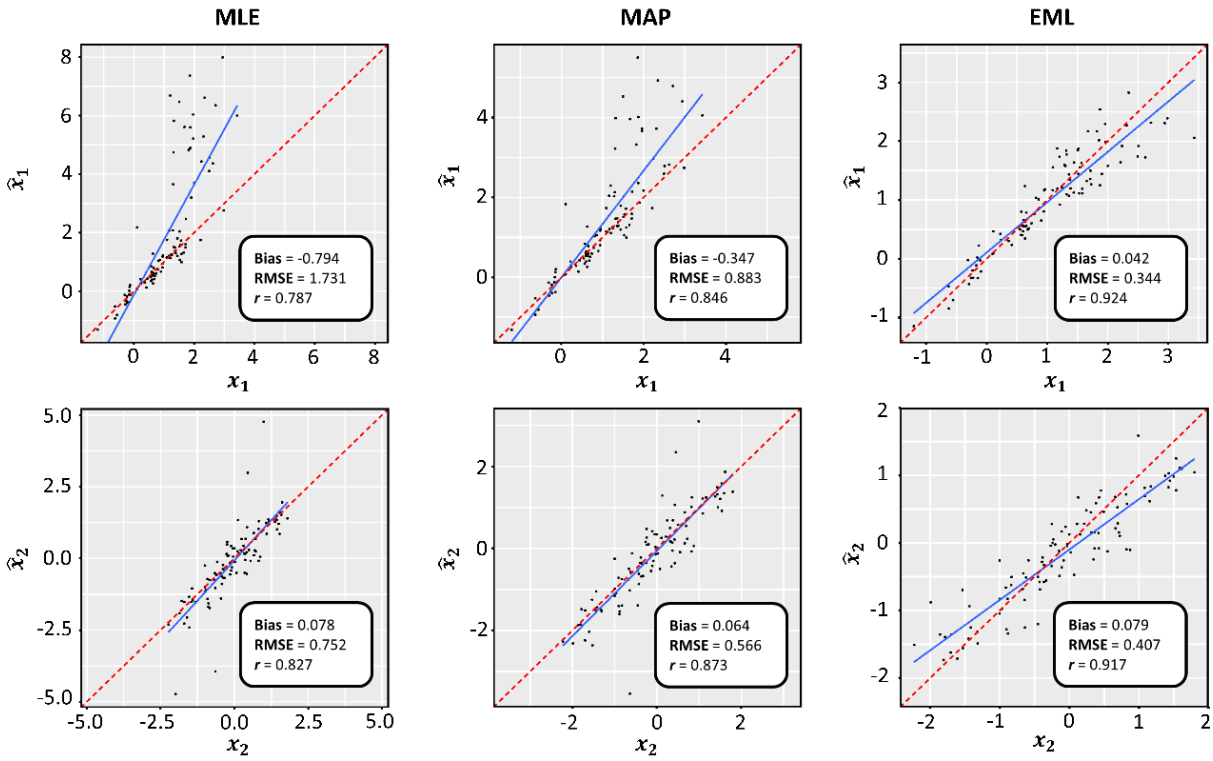



```
# MLE estimation (Figure 5, left panel)
estimate_twochoiceRL(data = data,
             method = "mle",
             plot = TRUE)

# MAP estimation (Figure 5 middle panel)
estimate_twochoiceRL(data = data,
             method = "map",
             prior_mean = c(0,0),
             prior_sd = c(5,5),
             plot = TRUE)

# EML estimation (Figure 5, right panel)
estimate_twochoiceRL(data = data,
             method = "eml",
             prior_mean = c(0,0),
             prior_sd = c(5,5),
             plot = TRUE)
```

Three important measures of fit were considered in assessing the fit between our estimated and true parameters. **Bias** is a measure of closeness between the estimates and truth; values may range in either direction – negative or positive – but, ideally, the bias value should be close to 0 (i.e., unbiased). **Root Mean Square Error** (RMSE) is a measure of spread in the residuals (difference between estimated and true values; prediction error); values – in principle – range from 0 (perfect prediction with no error) to ∞ (very poor prediction with substantial error). **Pearson's correlation coefficient** (r) is a measure of strength in association between two variables; values range from -1 (strong negative relationship) to +1 (strong positive relationship). To summarize, a good fit would, ideally, have a Bias $\sim$ 0, RMSE $\sim$ 0, and r $\sim$ 1 or -1.

There are many important differences to observe when comparing the estimation performances of our methods. Although the strengths in correlations prove relatively strong across methods (x_1: $r_{MLE} = 0.787$, $r_{MAP} = 0.846$, $r_{EML} = 0.924$; x_2: $r_{MLE} = 0.827$, $r_{MAP} = 0.873$, r_{EML} = 0.917; p < 0.0001), it's imperative we evaluate fit with additional measures. A reference line ($y = x$; dotted red line) was added to visually measure underlying overestimation or underestimation in the estimates. It's quite evident – especially from the MLE and MAP results – that values were overestimated (i.e., more points above the reference line) and underestimated; this is also captured in the Bias (negative values) and RMSE ($\gg 0$). The iterative nature of the EML algorithm, estimating the uncertainty of the parame-

Figure 5 ■ Parameter recovery using three estimation methods. We compare the performance of parameter recovery among the MLE, MAP and EML estimation methods. Each figure represents a scatter plot of our true (x-axis) vs estimated (y-axis) points. Ideally, all the points should be close to our reference $y = x$ line (red). We leverage Bias, RMSE and r to evaluate performance of these estimation methods. It's important to note that because of the different axis scales used, it might seem as if MLE and MAP, specifically for x2, are better than EML but as the r suggests that is not the case. Therefore, taking all the metrics into consideration, we can conclude that EML offers improved parameter recovery. Axes range are shared to better assess the quality of the estimates.



ters, minimized the problem of overestimation and underestimation. All things considered, the three methods offer compelling estimation performance, but the EML performance illustrates improved parameter recovery, highlighting the potential usefulness of EML in parameter estimation.

Discussion

Computational modeling serves as an important tool for exploring behavioral data. We can leverage different parameter estimation techniques for robust model fitting – relating parameters to behavioral differences.

Tutorial summary. We presented a tutorial on how to implement three estimation techniques, namely MLE, MAP and EML, in context to a simple and classic behavioral paradigm: the two-armed bandit task. We began with a brief overview of the Q-learning model which is composed of two parts: the Q-updating rule (which iteratively updates an individual's expectation of what the next choice should be) and the Softmax decision rule (which converts expected values of choices obtained from the Q-updating rule to probabilities of choosing a particular choice). This standard RL model is responsible for capturing behavior on the task via two parameters: learning rate (α) and choice randomness (β). We then proceeded to guide users – via R code – on how to simulate behavioral data for 100 people each with different α and β values (true parameters) and demonstrated parameter estimation with MLE, MAP and EML on the simulated data to recover the $\hat{\alpha}$ and $\hat{\beta}$ values (estimated parameters). Moreover, the flexibility of the code – for example, the ability to adjust reward structures of the stimuli, the trust region radius of the optimization algorithm, and priors of the parameters – gives users more control over aspects of model fitting and estimating

of simulated behavior.

Value of computational modeling. Fitting models, like reinforcement learning (RL) models, to behavioral data can provide insight into unclear behavioral phenomena. Several studies have leveraged Q-learning model parameters to characterize behavior between healthy and unhealthy groups of various disorders (Li, Lai, Liu, & Hsu, 2014; Sethi, Voon, Critchley, Cercignani, & Harrison, 2018; McCoy, Jahfari, Engels, Knapen, & Theeuwes, 2019). For example, Li et al fit behavioral data (obtained from the DRT – a two-choice task) to the Q-learning model and found that patients with schizophrenia had higher learning rates (α) than healthy controls and the choice randomness (β) trended negatively with severity of positive psychotic symptoms. The question of whether positive or negative symptoms of schizophrenia were related to dysfunction of RPE signaling was then revealed through model parameter correlations with symptom scales; dysfunctions of RPE signaling during the DRT was more associated with positive symptoms of psychosis. The findings from these studies underscore the importance of utilizing behavioral parameters to ascertain individual and group differences in behavior, and unraveling relationship between behavior and symptoms. Modeling could also have practical utility; for example, parameters could be predictors of response to different classes of antidepressants (Ang et al., 2020; Whitton et al., 2020).

Limitations. Though we provide a detailed walkthrough on modeling behavioral data, there are many other facets of computational modeling that should be considered but were not illustrated in this tutorial for simplicity's sake. These include implementing alternative estimation approaches, leveraging empirical priors, generating surrogate data, and performing model comparison. We also end this section with some additional resources to accompany this tutorial.

Alternative estimation approaches. The performance of MLE and MAP point-estimates resulted in good correlation, albeit overestimation of true values. On the other hand, the EML method, despite starting from relatively wider normal distribution priors, resulted in improved performance compared to MLE and MAP. It is also important to note that the presented approach is not a fully Bayesian approach since we have computed point estimates (MAP) rather than the full posterior. An alternative, robust yet computationally-expensive approach would be the Monte Carlo Markov Chain (MCMC) for parameter estimation (Makowski, Wallach, & Tremblay, 2002).

Empirical priors. Moreover, our prior here is estimated hierarchically on the same data, which has been suggested to increase the risk of overfitting (i.e., may be less generalizable). An alternative is to use empirical priors estimated from separate datasets (Gershman, 2016), although it currently remains unclear how hierarchical and empirical methods compare.

Surrogate data generation. In this tutorial, we have focused on simulated data as it allows us to know the true parameter values and how well they can be recovered. In the real world, however, the true parameters are unknown. So how can we determine whether our parameter recovery is satisfactory or not? One way to do that is to generate surrogate data – using recovered parameters to simulate performance based on the model (i.e., what we did using `simulate_twochoiceRL.R`) – and then compare the surrogate performance to real performance (Moulder, Boker, Ramseyer, & Tschacher, 2018).

Model comparison. It is also important to realize that here, we have focused only on estimating parameters for one model; Q-learning model with two free parameters (α, β). However, in reality, one might conceive several plausible models with different number of parameters that describe the different ways participants perform a task, so typically estimation has to be done for several models. How then does one decide which model is most suitable? Model comparison allows for the selection of the best model among a collection of candidate models. Two of the most common approaches are (i) Akaike Information Criterion (AIC), which selects the model that minimizes mean squared error, and (ii) Bayesian Information Criterion (BIC), which selects the model that maximizes log-likelihood (or minimizes negative log-likelihood) (Vrieze, 2012; Neath & Cavanaugh, 2012, 2). The interested reader is encouraged to refer to a review by Burnham & Anderson for an in-depth discussion on model selection (Burnham & Anderson, 2004).

Additional resources. The literature is rich with best practices for computational modeling and Bayesian parameter estimation. We provide a few that we believe will paint a useful picture of these concepts in concert with this tutorial. Wilson & Collins put together a great guide for those interested in modeling behavioral data (Wilson & Collins, 2019), and Zhang et al discuss the important pitfalls and best practices of using RL models to the fields of social, cognitive and affective neuroscience (Zhang, Lengersdorff, Mikus, Gläscher, & Lamm, 2020). In addition to computational modeling, an overview of Bayesian statistics for parameter estimation is intuitively presented in this primer (van de Schoot et al., 2021). Tutorials and software packages are becoming invaluable assets for practical learning of computational modeling of behavioral data. Ahn et al. have created a compendium of computational modeling functions for a wide-array of behavioral tasks (Ahn, Haines, & Zhang, 2017). Also, check out Nathaniel Haine's post on parameter esti-

mation where he compares the estimation performance of MLE, MAP and MCMC (http://haines-lab.com/post/2018-03-24-human-choice-and-reinforcement-learning-3/).

Conclusion

Modeling cognitive processes using behavioral data can provide researchers with an avenue to deeply understand the brain, which could lead to a better understanding of the mechanisms underlying behavioral differences. Therefore, it is important to educate researchers in this field with such computational approaches. This tutorial was written to elucidate the implementation of (1) a traditional reinforcement learning model to model behavior and (2) parameter estimation methods to recover behavior. We exhort more scientists to leverage these computational tools for estimating unknown parameters in behavioral models to obtain interesting behavioral insights and advance behavioral research.

Authors' note

YSA is supported by the Kaplen Fellowship in Depression from Harvard Medical School as well as the A*STAR National Science Scholarship. The authors thank the Editor and one anonymous Reviewer for detailed comments that have substantially improved the manuscript. Latest updates of the code can be found on Github at https://github.com/psuthaharan/twochoiceRL.

References

Ahn, W. Y., Haines, N., & Zhang, L. (2017). Revealing neurocomputational mechanisms of reinforcement learning and decision-making with the hbayesdm package. *Computational Psychiatry*, *1*, 24–57. doi:10.1162/CPSY_a_00002

Ang, Y. S., Kaiser, R., Deckersbach, T., Almeida, J., Phillips, M. L., Chase, H. W., … Pizzagalli, D. A. (2020). Pretreatment reward sensitivity and frontostriatal resting-state functional connectivity are associated with response to bupropion after sertraline nonresponse. *Biological Psychiatry*, *88*(8), 657–667. doi:10.1016/j.biopsych.2020.04.009

Ang, Y. S., Manohar, S., Plant, O., Kienast, A., Le Heron, C., Muhammed, K., … Husain, M. (2018). Dopamine modulates option generation for behavior. *Current Biology*, *28*(10), 1561–1569. doi:10.1016/j.cub.2018.03.069

Bakeman, R., & Robinson, B. F. (2005). *Understanding statistics in the behavioral sciences*. doi:10.4324/9781410612625

Bickel, W. K., Odum, A. L., & Madden, G. J. (1999). Impulsivity and cigarette smoking: Delay discounting in current, never and ex-smokers. *Psychopharmacology*, *146*(4), 447–454. doi:10.1007/PL00005490

Burnham, K. P., & Anderson, D. R. (2004). Multimodel inference: Understanding aic and bic in model selection. *Sociological methods & research*, *33*(2), 261–304. doi:10.1177/0049124104268644

Corlett, P. R., & Fletcher, P. C. (2014). Computational psychiatry: A rosetta stone linking the brain to mental illness. *The Lancet Psychiatry*, *1*(5), 399–402. doi:10.1016/S2215-0366(14)70298-6

Cousineau, D., & Hélie, S. (2013). Improving maximum likelihood estimation using prior probabilities: A tutorial on maximum a posteriori estimation and an examination of the weibull distribution. *Tutorials in Quantitative Methods for Psychology*, *9*(2), 61–71. doi:10.20982/tqmp.09.2.p061

Daw, N. D. (2011). Trial-by-trial data analysis using computational models. *Decision making, affect, and learning: Attention and performance XXIII*, *23*, 1–99. doi:10.1093/acprof:oso/9780199600434.003.0001

Daw, N. D., O'doherty, J. P., Dayan, P., Seymour, B., & Dolan, R. J. (2006). Cortical substrates for exploratory decisions in humans. *Nature*, *441*(7095), 876–879. doi:10.1038/nature04766

Dillon, D. G., Wiecki, T., Pechtel, P., Webb, C., Goer, F., Murray, L., … Parsey, R. (2015). A computational analysis of flanker interference in depression. *Psychological medicine*, *45*(11), 2333–2344. doi:10.1017/S0033291715000276

Do, C. B., & Batzoglou, S. (2008). What is the expectation maximization algorithm? *Nature biotechnology*, *26*(8), 897–899. doi:10.1038/nbt1406

Etz, A. (2018). Introduction to the concept of likelihood and its applications. *Advances in Methods and Practices in Psychological Science*, *1*(1), 60–69. doi:10.1177/2515245917744314

Gershman, S. J. (2016). Empirical priors for reinforcement learning models. *Journal of Mathematical Psychology*, *71*, 1–6. doi:10.1016/j.jmp.2016.01.006

Geyer, C. J. (2020). Trust: Trust region optimization (Version 0.1.8). Retrieved from http://CRAN.R-project.org/package=trust

Huys, Q. J., Browning, M., Paulus, M., & Frank, M. J. (2020). Advances in the computational understanding of mental illness. *Neuropsychopharmacology*, *1*, 1–19. doi:10.1038/s41386-020-0746-4

Huys, Q. J., Cools, R., Gölzer, M., Friedel, E., Heinz, A., Dolan, R. J., & Dayan, P. (2011). Disentangling the roles of approach, activation and valence in instrumental and pavlovian responding. *PLoS Computational Biology*, *7*(4), 1–99. doi:10.1371/journal.pcbi.1002028

Huys, Q. J., Maia, T. V., & Frank, M. J. (2016). Computational psychiatry as a bridge from neuroscience to clinical applications. *Nature neuroscience*, *19*(3), 404–499. doi:10.1038/nn.4238

Huys, Q. J., Pizzagalli, D. A., Bogdan, R., & Dayan, P. (2013). Mapping anhedonia onto reinforcement learning: A behavioural meta-analysis. *Biology of mood & anxiety disorders*, *3*(1), 12–99. doi:10.1186/2045-5380-3-12

Li, C. T., Lai, W. S., Liu, C. M., & Hsu, Y. F. (2014). Inferring reward prediction errors in patients with schizophrenia: A dynamic reward task for reinforcement learning. *Frontiers in psychology*, *5*, 1282–1299. doi:10.3389/fpsyg.2014.01282

Lockwood, P. L., Hamonet, M., Zhang, S. H., Ratnavel, A., Salmony, F. U., Husain, M., & Apps, M. A. (2017). Prosocial apathy for helping others when effort is required. *Nature human behaviour*, *1*(7), 0131–0199. doi:10.1038/s41562-017-0131

Luce, R. D. (1986). *Response times: Their role in inferring elementary mental organization (no. 8)*. on Demand: Oxford University Press.

Makowski, D., Wallach, D., & Tremblay, M. (2002). Using a Bayesian approach to parameter estimation; comparison of the glue and mcmc methods. *Agronomie*, *22*(2), 191–203. doi:10.1051/agro:2002007

McCoy, B., Jahfari, S., Engels, G., Knapen, T., & Theeuwes, J. (2019). Dopaminergic medication reduces striatal sensitivity to negative outcomes in parkinson's disease. *Brain*, *142*(11), 3605–3620. doi:10.1093/brain/awz276

Momennejad, I., Russek, E. M., Cheong, J. H., Botvinick, M. M., Daw, N. D., & Gershman, S. J. (2017). The successor representation in human reinforcement learning. *Nature Human Behaviour*, *1*(9), 680–692. doi:10.1038/s41562-017-0180-8

Montague, P. R., Dolan, R. J., Friston, K. J., & Dayan, P. (2012). Computational psychiatry. *Trends in cognitive sciences*, *16*(1), 72–80. doi:10.1016/j.tics.2011.11.018

Moulder, R. G., Boker, S. M., Ramseyer, F., & Tschacher, W. (2018). Determining synchrony between behavioral time series: An application of surrogate data generation for establishing falsifiable null-hypotheses. *Psychological methods*, *23*(4), 757–799. doi:10.1037/met0000172

Myung, I. J. (2003). Tutorial on maximum likelihood estimation. *Journal of mathematical Psychology*, *47*(1), 90–100. doi:10.1016/S0022-2496(02)00028-7

Neath, A. A., & Cavanaugh, J. E. (2012). The Bayesian information criterion: Background, derivation, and applications. *4*, 199–203. doi:10.1002/wics.199

Niv, Y. (2020). On the primacy of behavioral research for understanding the brain. In Y. Niv (Ed.), *Current controversies in philosophy of cognitive science* (pp. 134–151). doi:10.4324/9781003026273-16

Pagano, R. R. (2012). *Understanding statistics in the behavioral sciences (vol. 1)*. Washington: Cengage Learning.

Patzelt, E. H., Hartley, C. A., & Gershman, S. J. (2018). Computational phenotyping: Using models to understand individual differences in personality, development, and mental illness. *Personality Neuroscience*, *1*, 1–99. doi:10.1017/pen.2018.14

Racine, J. S. (2012). Rstudio: A platform-independent ide for r and sweave. *Journal of Applied Econometrics*, *27*, 167–172. doi:10.1002/jae.1278

Reed, E. J., Uddenberg, S., Suthaharan, P., Mathys, C. H., Taylor, J. R., Groman, S. M., & Corlett, P. R. (2020). Paranoia as a deficit in non-social belief updating. *Elife*, *9*(1), 1–99. doi:10.7554/eLife.56345

Schulz, E., & Gershman, S. J. (2019). The algorithmic architecture of exploration in the human brain. *Current opinion in neurobiology*, *55*, 7–14. doi:10.1016/j.conb.2018.11.003

Sethi, A., Voon, V., Critchley, H. D., Cercignani, M., & Harrison, N. A. (2018). A neurocomputational account of reward and novelty processing and effects of psychostimulants in attention deficit hyperactivity disorder. *Brain*, *141*(5), 1545–1557. doi:10.1093/brain/awy048

Slivkins, A. (2019). Introduction to multi-armed bandits. *Foundations and Trends in Machine Learning*, *12*(1-2), 1–286. doi:10.1561/2200000068

Stephan, K. E., & Mathys, C. (2014). Computational approaches to psychiatry. *Current opinion in neurobiology*, *25*, 85–92. doi:10.1016/j.conb.2013.12.007

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. doi:10.1109/TNN.1998.712192

van de Schoot, R., Depaoli, S., King, R., Kramer, B., Martens, K., Tadesse, M., ... Yau, C. (2021). Bayesian statistics and modelling. *Nature Review Methods Primers*, *1*, 1–99. doi:10.1038/s43586-020-00001-2

Venables, W. N., Smith, D. M., & R Development Core Team. (2009). *An introduction to r*. Racoon City: Bill & Ted.

Vrieze, S. I. (2012). Model selection and psychological theory: A discussion of the differences between the akaike information criterion (aic) and the Bayesian information criterion (bic). *Psychological methods*, *17*(2), 228–299. doi:10.1037/a0027127

Wang, X. J., & Krystal, J. H. (2014). Computational psychiatry. *Neuron*, *84*(3), 638–654. doi:10.1016/j.neuron.2014.10.018

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. Racoon City: Bill & Ted.

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine learning*, *8*(3-4), 279–292. doi:10.1007/BF00992698

Whitton, A. E., Reinen, J. M., Slifstein, M., Ang, Y. S., McGrath, P. J., Iosifescu, D. V., ... Schneier, F. R. (2020). Baseline reward processing and ventrostriatal dopamine function are associated with pramipexole response in depression. *Brain*, *143*(2), 701–710. doi:10.1093/brain/awaa002

Wickham, H. (2016). *Ggplot2: Elegant graphics for data analysis*. doi:10.1007/978-3-319-24277-4

Wilson, R. C., & Collins, A. G. (2019). Ten simple rules for the computational modeling of behavioral data. *Elife*, *8*, e49547–99. doi:10.7554/eLife.49547

Zhang, L., Lengersdorff, L., Mikus, N., Gläscher, J., & Lamm, C. (2020). Using reinforcement learning models in social neuroscience: Frameworks, pitfalls and suggestions of best practices. *Social cognitive and affective neuroscience*, *15*(6), 695–707. doi:10.1093/scan/nsaa089

Appendix A: Code to simulate, visualize, and estimate two-choice behavior

Listing 1. R code to simulate two-choice behavior.

```
Usage:
# Function to simulate two-choice data with default values
simulate_twochoiceRL(seed_value = 528,
                     n_subj = 100,
                     n_tr = 200,
                     mean_x = c(1,0),
                     stdev_x = c(1,1),
                     mean_payoff = c(1,-1),
                     stdev_payoff = c(2,2),
                     trials_unique = TRUE,
                     progress_bar = TRUE)
```

Source code:

```
simulate_twochoiceRL <- function(seed_value = 528,
                                 n_subj = 100,
                                 n_tr = 200,
                                 mean_x = c(1,0),
                                 stdev_x = c(1,1),
                                 mean_payoff = c(1,-1),
                                 stdev_payoff = c(2,2),
                                 trials_unique = TRUE,
                                 progress_bar = TRUE) { # start simulation

  # initialize list for storing data for all subjects
  twochoiceRL_data <- list()

  # set seed value for replication
  set.seed(seed_value)

  # randomly generate values
  x1  <- rnorm(n_subj,mean_x[1],stdev_x[1])
  x2  <- rnorm(n_subj,mean_x[2],stdev_x[2])

  # re-parameterize model parameters
  beta <- exp(x1) # choice randomness
  alpha <- 1/(1+exp(-x2)) # learning rate

  # create progress bar if user specified
  if (progress_bar == TRUE){
```

```
  pb <- txtProgressBar(min = 0, max = data$subjects, style = 3)
}

# simulate trial-by-trial data for all subjects
for (i in 1:n_subj){ # start loop for subjects

  # initialize expected value to zero for both choices
  Q <- c(0, 0)

  # simulate choice behavior across trials
  simulate.trials <- foreach(t=1:n_tr,
                     .combine = "rbind") %do% { # start loop for trials

    # Mean payoff for choices 1 and 2
    mu    <- c(mean_payoff[1], mean_payoff[2])

    # Standard deviation of payoff for choices 1 and 2
    sigma <- c(stdev_payoff[1], stdev_payoff[2])

    # Generate action probability using softmax
    action_prob <- softmax(beta[i]*Q)

    # Use action probability to sample participant action
    action <- sample(c(1,2), size = 1, prob = action_prob)

    # Generate reward based on action
    reward <- rnorm(1, mean = mu[action], sd = sigma[action])

    # Q-learning
    Q[action] <- Q[action] + alpha[i] * (reward - Q[action])

    # Save data
    df <- data.frame(Trial    = rep(t, 2),        # trial number
                     Value    = Q,                # expected value
                     Pr       = action_prob,      # choice probability
                     Option   = paste(1:2),       # presented stimuli
                     Action   = rep(action, 2),   # chosen stimuli
                     Reward   = rep(reward, 2))   # reward received

    if (trials_unique == TRUE){
      # only keep the trials where option was selected
      df[which(df$Option == df$Action),]

    } else {
      # keep all trials for plotting purposes
      df
    }

  } # end loop for trials

  # store trial-by-trial data for each subject
  twochoiceRL_data[[i]] <- simulate.trials
```

```
79
80      # load progress bar with loop index i
81      if (progress_bar == TRUE){
82        Sys.sleep(0.1)
83        setTxtProgressBar(pb, i)
84      }
85
86    } # End for loop for subjects
87
88    # close progress bar
89    if (progress_bar == TRUE){
90      close(pb)
91    }
92
93    # return list with the data, number of subjects, true x1/x2 values
94    MyList <- list("twochoiceRL" = twochoiceRL_data,
95                   "subjects" = n_subj,
96                   "x1" = x1,
97                   "x2" = x2)
98    return(MyList)
99
100 } # end simulation
```

Listing 2. R code to visualize two-choice behavior.

Usage:

```
# Saving simulated data for plotting purpose
data <- simulate_twochoiceRL(trials_unique = FALSE)

# Function to plot two-choice data with default values
plot_twochoiceRL(data = data,
     subj = 100,
     colors = c("orange","purple"),
     plot_type = "static")
```

Source code:

```
1  plot_twochoiceRL <- function(data = NULL,
2                               subj = 100,
3                               colors = c("orange", "purple"),
4                               plot_type = "static") { # start plotting
5
6    if (plot_type == "static"){ # start static plot
7
8    # Plot expected value across trials
9    g1 <- ggplot(data$twochoiceRL[[subj]],
10                aes(x=data$twochoiceRL[[subj]]$Trial,
11                    y=data$twochoiceRL[[subj]]$Value,
12                    fill=data$twochoiceRL[[subj]]$Option)) +
13     geom_point() + geom_smooth() +
14     xlab("Trial number") + ylab("Expected value") +
15     scale_fill_manual(name = "Machine",
16                       labels = c("A","B"),
```

```
                      values = c(colors[1],colors[2]))

  # Plot choice probability across trials
  g2 <- ggplot(data$twochoiceRL[[subj]],
               aes(x=data$twochoiceRL[[subj]]$Trial,
                   y=data$twochoiceRL[[subj]]$Pr,
                   fill=data$twochoiceRL[[subj]]$Option)) +
     geom_point() + geom_smooth() +
     xlab("Trial number") + ylab("Choice probability") +
     scale_fill_manual(name = "Machine",
                       labels = c("A","B"),
                       values = c(colors[1],colors[2]))

  # arrange plots
  plot <- ggarrange(g1,g2,
                    ncol=2,nrow=1,
                    common.legend = TRUE,
                    legend = 'bottom')

  return(plot)
  } # end static plot

  if (plot_type == "animate"){ # start animation plot
    # Plot choice probability across trials
    plot <- ggplot(data = data$twochoiceRL[[subj]],
                   aes(x = data$twochoiceRL[[subj]]$Trial,
                       y = data$twochoiceRL[[subj]]$Pr,
                       color = data$twochoiceRL[[subj]]$Option)) +
     geom_point(aes(group = seq_along(data$twochoiceRL[[subj]]$Trial)),
                size = 4,
                alpha = 0.7) +
     geom_line(aes(lty = data$twochoiceRL[[subj]]$Option),
               alpha = 0.6) +
     labs(y = "Choice probability",
          x = "Trial",
          title = "") +
     scale_linetype_manual(name = "",
                           labels = c("A","B"),
                           values = c("solid","solid"),
                           guide = FALSE) +
     scale_color_manual(name = "Machine",
                        labels = c("A","B"),
                        values = c(colors[1],colors[2]))

    ## Animated Plot
    animate.plot <- plot +
         transition_reveal(along = data$twochoiceRL[[subj]]$Trial)

    return(animate.plot)
  } # end animation plot

} # end plotting
```

Listing 3. R code to estimate two-choice behavior.

Usage:

```
# Saving simulated data for estimating purpose
data <- simulate_twochoiceRL(trials_unique = TRUE)

# Function to estimate two-choice behavior with default values
estimate_twochoiceRL(seed_value = 528,
                     data = NULL,
                     method = "mle",
                     nRes = 5,
   tr_rad = c(1,5),
                     prior_mean = c(NULL,NULL),
                     prior_sd = c(NULL,NULL),
                     plot = FALSE,
                     progress_bar = TRUE)
```

Source code:

```
1  estimate_twochoiceRL <- function(seed_value = 528,
2                                   data = NULL,
3                                   method = "mle",
4                                   nRes = 5,
5                                   tr_rad = c(1,5),
6                                   prior_mean = c(NULL,NULL),
7                                   prior_sd = c(NULL,NULL),
8                                   plot = FALSE,
9                                   progress_bar = TRUE) { # start estimation
10
11   # ensures replication of result - the rnorm() function
12   # used in the next few lines will result in the same
13   # sequence of random numbers for the specified seed
14   set.seed(seed_value)
15
16   # if specified estimation method is MLE, then run MLE
17   if (method == "mle") {
18
19     # randomly generate initial guesses for parameters
20     init_x1 <- rnorm(data$subjects,0,5)
21     init_x2 <- rnorm(data$subjects,0,5)
22
23     # perform MLE
24     mle_data <- mle_twochoiceRL(
25       data = data,                        # simulated task data
26       param = list(init_x1,init_x2),      # initial guesses for parameters
27       fn = "mle.objectiveFunction",       # likelihood function being minimized
28       opt = "TRM",                        # trust-region optimization method
29       radius = c(tr_rad[1],tr_rad[2]),    # initial and max allowed radius
30       nRes = nRes,                        # random restarts
31       progress_bar = progress_bar)        # track completion time of estimation
32
33     # save mle results
34     x1_hat <- mle_data[[2]] # estimates of x1
```

```
    x2_hat <- mle_data[[3]] # estimates of x2

    # store true and estimated parameters
    x1 <- data$x1
    x2 <- data$x2

    df <- data.frame(x1,
                     x2,
                     x1_hat,
                     x2_hat)

    # mle function output
    results <- list(value = df,
                    bias_x1 = bias(x1,x1_hat),
                    bias_x2 = bias(x2,x2_hat),
                    rmse_x1 = rmse(x1,x1_hat),
                    rmse_x2 = rmse(x2,x2_hat),
                    corr_x1 = cor(x1,x1_hat),
                    corr_x2 = cor(x2,x2_hat))

    # if user doesn't want to see the plot, then only return results
    if (plot == FALSE){
      return(results)
    }
    else {

      # generate plot 1
      p1 <- ggplot(df,
                   aes(x=x1,y=x1_hat)) +
        geom_point() +
        geom_smooth(method = "lm", se = FALSE) +
        coord_cartesian(xlim=c(min(x1,x1_hat),
                               max(x1,x1_hat)),
                        ylim=c(min(x1,x1_hat),
                               max(x1,x1_hat))) +
         labs(x = expression("x"[1]),
              y = expression(hat(x)[1])) +
        theme(#axis.title.y = element_blank(),
              #axis.title.x = element_blank(),
              #axis.text = element_blank(),
              panel.background = element_rect(),
              panel.grid.major = element_line(size=1),
              panel.grid.minor = element_line(size=1),
              axis.ticks = element_line(colour="black",
                          size = 1.5),
              panel.border = element_rect(colour = "black",
                          fill = NA,
                          size = 2.5),
              legend.text = element_blank(),
              legend.position = "none",
              aspect.ratio = 1) +
        annotate("label",
```

```
                label = paste("bias =", round(bias(x1,x1_hat),3), "\n",
                              "rmse =", round(rmse(x1,x1_hat),3),"\n",
                              "r =", round(cor(x1,x1_hat,
                                   method = "pearson"),3)),
                x=max(x1), # adjust position based on plot
                y=min(x1_hat)+1, # adjust position based on plot
                size=4) +

      # add reference line
      geom_abline(intercept = 0, slope = 1,
                linetype = "dashed",
                size = 1,
                color = "red")

      # generate plot 2
      p2 <- ggplot(df,
                   aes(x=x2,y=x2_hat)) +
        geom_point() +
        geom_smooth(method = "lm", se = FALSE) +
        coord_cartesian(xlim=c(min(x2,x2_hat),
                               max(x2,x2_hat)),
                        ylim=c(min(x2,x2_hat),
                               max(x2,x2_hat))) +
        labs(x = expression("x"[2]),
             y = expression(hat(x)[2])) +
        theme(#axis.title.y = element_blank(),
              #axis.title.x = element_blank(),
              #axis.text = element_blank(),
              panel.background = element_rect(),
              panel.grid.major = element_line(size=1),
              panel.grid.minor = element_line(size=1),
              axis.ticks = element_line(colour="black",
                             size = 1.5),
              panel.border = element_rect(colour = "black",
                            fill = NA,
                            size = 2.5),
              legend.text = element_blank(),
              legend.position = "none",
              aspect.ratio = 1) +
        annotate("label",
                 label = paste("bias =", round(bias(x2, x2_hat),3),"\n",
                               "rmse =", round(rmse(x2, x2_hat),3),"\n",
                               "r =", round(cor(x2, x2_hat,
                               method = "pearson"),3)),
                 x=max(x2), # adjust position based on plot
                 y=min(x2_hat)+1, # adjust position based on plot
                 size=4) +

      # add reference line
      geom_abline(intercept = 0, slope = 1,
            linetype = "dashed",
            size = 1,
```

```
            color = "red")

      # combine plots
      plot <- ggarrange(p1,p2,
                ncol = 2, nrow = 1,
                common.legend = TRUE,
                legend = 'bottom')

      # return MLE estimation values and plot
      return(list(results, p1, p2, plot))
    }

  } else if (method == "map"){ # if MAP specified, run MAP

    # randomly generate initial guesses for parameters
    init_x1 <- rnorm(data$subjects,0,5)
    init_x2 <- rnorm(data$subjects,0,5)

    # use default priors if no prior information is specified
    if (is.null(prior_mean) == TRUE && is.null(prior_sd) == TRUE){
      message("MAP requires specification of initial priors;
            defaulting to prior_mean = c(0,0) and prior_sd = c(5,5)")

      # initialize priors for parameters
      m1 = 0
      s1 = 5
      m2 = 0
      s2 = 5
    } else {
      # initialize priors for parameters
      m1 = prior_mean[1]
      s1 = prior_sd[1]
      m2 = prior_mean[2]
      s2 = prior_sd[2]
    }

    map_data <- map_twochoiceRL(
      data = data,                      # simulated task data
      param = list(init_x1,init_x2),    # initial guesses for parameters
      prior_mean = c(m1,m2),            # initial prior means
      prior_sd = c(s1,s2),              # initial prior standard deviation
      fn = "map.objectiveFunction",     # likelihood function being minimized
      opt = "TRM",                      # trust-region optimization method
      radius = c(tr_rad[1],tr_rad[2]), # initial and max allowed radius
      nRes = nRes,                      # random restarts
      progress_bar = progress_bar)      # track completion time of estimation

    # save map results
    x1_hat <- map_data[[2]] # estimates of x1
    x2_hat <- map_data[[3]] # estimates of x2

    # store true and estimated parameters
```

```
    x1 <- data$x1
    x2 <- data$x2

    df <- data.frame(x1,
                     x2,
                     x1_hat,
                     x2_hat)

    # map function output
    results <- list(value   = df,
                    bias_x1 = bias(x1,x1_hat),
                    bias_x2 = bias(x2,x2_hat),
                    rmse_x1 = rmse(x1,x1_hat),
                    rmse_x2 = rmse(x2,x2_hat),
                    corr_x1 = cor(x1,x1_hat),
                    corr_x2 = cor(x2,x2_hat)
    )

    if (plot == FALSE){
      return(results)
    }
    else {
      # generate plot 1
      p1 <- ggplot(df,
                   aes(x=x1,y=x1_hat)) +
        geom_point() +
        geom_smooth(method = "lm", se = FALSE) +
        coord_cartesian(xlim=c(min(x1,x1_hat),
                               max(x1,x1_hat)),
                        ylim=c(min(x1,x1_hat),
                               max(x1,x1_hat))) +
        labs(x = expression("x"[1]),
             y = expression(hat(x)[1])) +
        theme(#axis.title.y = element_blank(),
              #axis.title.x = element_blank(),
              #axis.text = element_blank(),
              panel.background = element_rect(),
              panel.grid.major = element_line(size=1),
              panel.grid.minor = element_line(size=1),
              axis.ticks = element_line(colour="black", size = 1.5),
              panel.border = element_rect(colour = "black",
                                          fill = NA,
                                          size = 2.5),
              legend.text = element_blank(),
              legend.position = "none",
              aspect.ratio = 1) +
        annotate("label",
                 label = paste("bias =", round(bias(x1,x1_hat),3), "\n",
                               "rmse =", round(rmse(x1,x1_hat),3),"\n",
                               "r =", round(cor(x1, x1_hat,
                               method = "pearson"),3)),
                 x=max(x1), # adjust position based on plot
```

```
                y=min(x1_hat)+1, # adjust position based on plot
                size=4) +
    # add reference line
    geom_abline(intercept = 0, slope = 1,
                linetype = "dashed",
                size = 1,
                color = "red")

# generate plot 2
p2 <- ggplot(df,
             aes(x=x2,y=x2_hat)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  coord_cartesian(xlim=c(min(x2,x2_hat),
                         max(x2,x2_hat)),
                  ylim=c(min(x2,x2_hat),
                         max(x2,x2_hat))) +
  labs(x = expression("x"[2]),
       y = expression(hat(x)[2])) +
  theme(#axis.title.y = element_blank(),
        #axis.title.x = element_blank(),
        #axis.text = element_blank(),
        panel.background = element_rect(),
        panel.grid.major = element_line(size=1),
        panel.grid.minor = element_line(size=1),
        axis.ticks = element_line(colour="black", size = 1.5),
        panel.border = element_rect(colour = "black",
                                    fill = NA,
                                    size = 2.5),
        legend.text = element_blank(),
        legend.position = "none",
        aspect.ratio = 1) +
  annotate("label",
           label = paste("bias =", round(bias(x2, x2_hat),3),"\n",
                         "rmse =", round(rmse(x2, x2_hat),3),"\n",
                         "r =", round(cor(x2, x2_hat,
                              method = "pearson"),3)),
           x=max(x2),
           y=min(x2_hat)+1,
           size=4) +
# add reference line
geom_abline(intercept = 0, slope = 1,
      linetype = "dashed",
      size = 1,
      color = "red")

# combine plots
plot <- ggarrange(p1,p2,
          ncol = 2, nrow = 1,
          common.legend = TRUE,
          legend = 'bottom')
```

```
295
296       # return MAP estimation values and plot
297       return(list(results, p1, p2, plot))
298     }
299
300   } else if (method == "eml") {
301     d <-numeric() # store difference of log-likelihood b/w iterations
302     diff = 10000  # set large likelihood value difference for convergence
303     prev = Inf    # initialize previous log-likelihood value
304     iter = 1      # initial iteration of E-M step
305
306     # initialize variable to store posterior hyperparameters
307     posterior_hyperparameters <- list()
308
309     # Perform E-M step with Laplace Approximation
310     while(diff > 0.001){ # start E-M procedure
311
312       # generate initial guesses for iterations
313       if (iter == 1){ # first iteration, generate guesses randomly
314         init_x1 <- rnorm(data$subjects,0,5)
315         init_x2 <- rnorm(data$subjects,0,5)
316
317         # use default priors if no prior information is specified
318         if (is.null(prior_mean) == TRUE && is.null(prior_sd) == TRUE){
319
320           message("EML requires specification of initial prior;
321                   defaulting to prior_mean = c(0,0) and prior_sd = c(5,5)")
322
323           # initialize priors for parameters
324           m1 = 0
325           s1 = 5
326           m2 = 0
327           s2 = 5
328         } else {
329           # initialize priors for parameters
330           m1 = prior_mean[1]
331           s1 = prior_sd[1]
332           m2 = prior_mean[2]
333           s2 = prior_sd[2]
334         }
335       } else { # successive iterations, use previous 'iterations MAP
336         init_x1 <- eml_data[[2]]
337         init_x2 <- eml_data[[3]]
338         m1 <- m1_laplace
339         s1 <- s1_laplace
340         m2 <- m2_laplace
341         s2 <- s2_laplace
342       }
343
344       # Run E-step
345       eml_data <- eml_twochoiceRL(
346         data = data,                    # simulated task data
```

```
347         param = list(init_x1,init_x2), # initial guesses for parameters
348         prior_mean = c(m1,m2),         # initial prior means
349         prior_sd = c(s1,s2),           # initial prior standard deviations
350         fn = "eml.objectiveFunction",  # likelihood function being minimized
351         opt = "TRM",                   # trust-region optimization method
352         radius = c(tr_rad[1],tr_rad[2]), # initial and max allowed radius
353         nRes = nRes,                   # random restarts
354         iter = iter,                   # iteration number
355         eml_data = eml_data,           # previous 'iterations output
356         progress_bar = progress_bar)   # track completion time of estimation
357
358       # calculate difference in log-likelihood for convergence
359       diff <- abs(eml_data[[1]] - prev)
360       d[iter] <- diff
361       prev <- eml_data[[1]]
362
363       # print output in R console
364       print(paste("iter", iter, ":",
365                   "LL =", round(eml_data[[1]],3),",",
366                   "diff =", round(diff,3),",",
367                   "(m1,m2,s1,s2) =", "(",round(m1,3),",",
368                                          round(m2,3),",",
369                                          round(s1,3),",",
370                                          round(s2,3),")"))
371       print("-----------------------------------------------------------")
372
373       # store posterior hyperparameters
374       posterior_hyperparameters[[iter]] <- c(iter,m1,m2,s1,s2)
375
376       # Run M-step with Laplace approximation: update hyperparameters
377       m1_laplace <- mean(eml_data[[2]])
378       s1_laplace <- sqrt(sum((eml_data[[2]]^(2))+
379                              (eml_data[[4]])-(2*eml_data[[2]]*m1)+
380                              (rep(m1^(2),data$subjects)))/(data$subjects-1))
381
382       m2_laplace <- mean(eml_data[[3]])
383       s2_laplace <- sqrt(sum((eml_data[[3]]^(2))+
384                              (eml_data[[5]])-(2*eml_data[[3]]*m2)+
385                              (rep(m2^(2),data$subjects)))/(data$subjects-1))
386
387       # iterate
388       iter = iter+1
389
390     } # end E-M procedure
391
392     # save eml results
393     x1_hat <- eml_data[[2]] # estimates of x1
394     x2_hat <- eml_data[[3]] # estimates of x2
395
396     # store true and estimated parameters
397     x1 <- data$x1
398     x2 <- data$x2
```

```
399
400     df <- data.frame(x1,
401                      x2,
402                      x1_hat,
403                      x2_hat)
404
405     posterior_hyperparam_vals <- as.data.frame(
406                                  matrix(
407                                   unlist(posterior_hyperparameters),
408                                   length(posterior_hyperparameters),5,
409                                   byrow = TRUE
410                                  ))
411     colnames(posterior_hyperparam_vals) <- c("iter","m1","m2","s1","s2")
412
413     # eml function output
414     results <- list(value = df,
415                     bias_x1 = bias(x1,x1_hat),
416                     bias_x2 = bias(x2,x2_hat),
417                     rmse_x1 = rmse(x1,x1_hat),
418                     rmse_x2 = rmse(x2,x2_hat),
419                     corr_x1 = cor(x1,x1_hat),
420                     corr_x2 = cor(x2,x2_hat),
421                     posterior_vals = posterior_hyperparam_vals)
422
423     # generate Gaussian density from posterior hyperparameters
424     post_hyper_param <- results$posterior_vals
425
426     # x1
427     post_hyper_param_x1 <- data.frame(
428        mean = post_hyper_param$m1,
429        stdev = post_hyper_param$s1,
430        iter = paste0("Iter_",sprintf("%02.0f",1:nrow(post_hyper_param))),
431        stringsAsFactors = F)
432     # x2
433     post_hyper_param_x2 <- data.frame(
434       mean = post_hyper_param$m2,
435       stdev = post_hyper_param$s2,
436       iter = paste0("Iter_",sprintf("%02.0f",1:nrow(post_hyper_param))),
437       stringsAsFactors = F)
438
439     # points at which to evaluate the Gaussian densities
440     x1_eval <- seq(-10,10, by = 0.01) # adjust to cover the range
441     x2_eval <- seq(-10,10, by = 0.01) # adjust to cover the range
442
443     # compute Gaussian densities based on means and standard deviations
444     pdf_x1 <- mapply(dnorm,
445                      mean = post_hyper_param_x1$mean,
446                      sd = post_hyper_param_x1$stdev,
447                      MoreArgs = list(x = x1_eval),
448                      SIMPLIFY = FALSE)
449
450     pdf_x2 <- mapply(dnorm,
```

```
                    mean = post_hyper_param_x2$mean,
                    sd = post_hyper_param_x2$stdev,
                    MoreArgs = list(x = x2_eval),
                    SIMPLIFY = FALSE)

  # add group names
  names(pdf_x1) <- post_hyper_param_x1$iter
  names(pdf_x2) <- post_hyper_param_x2$iter

  # convert list to dataframe
  pdf_x1 <- do.call(cbind.data.frame, pdf_x1)
  pdf_x1$x1_eval <- x1_eval

  pdf_x2 <- do.call(cbind.data.frame, pdf_x2)
  pdf_x2$x2_eval <- x2_eval

  # convert dataframe to long format
  x1_long <- gather(pdf_x1, iter, density, -x1_eval)
  x2_long <- gather(pdf_x2, iter, density, -x2_eval)

  if (plot == FALSE){
    return(results)
  }
  else {

    # generate plot 1
    p1 <- ggplot(df,
                 aes(x=x1,y=x1_hat)) +
      geom_point() +
      geom_smooth(method = "lm", se = FALSE) +
      coord_cartesian(xlim=c(min(x1,x1_hat),
                             max(x1,x1_hat)),
                      ylim=c(min(x1,x1_hat),
                             max(x1,x1_hat))) +
      labs(x = expression("x"[1]),
           y = expression(hat(x)[1])) +
      theme(#axis.title.y = element_blank(),
            #axis.title.x = element_blank(),
            #axis.text = element_blank(),
            panel.background = element_rect(),
            panel.grid.major = element_line(size=1),
            panel.grid.minor = element_line(size=1),
            axis.ticks = element_line(colour="black", size = 1.5),
            panel.border = element_rect(colour = "black",
                                        fill = NA,
                                        size = 2.5),
            legend.text = element_blank(),
            legend.position = "none",
            aspect.ratio = 1) +
       annotate("label",
                label = paste("bias =", round(bias(x1,x1_hat),3), "\n",
                              "rmse =", round(rmse(x1,x1_hat),3),"\n",
```

```
                              "r =", round(cor(x1,x1_hat,
                                               method = "pearson"),3)),
                x=max(x1)-1,
                y=min(x1_hat)+0.5,
                size=4) +
    # add reference line
    geom_abline(intercept = 0, slope = 1,
                linetype = "dashed",
                size = 1,
                color = "red")

  # generate plot 2
  p2 <- ggplot(df,
               aes(x=x2,y=x2_hat)) +
    geom_point() +
    geom_smooth(method = "lm", se = FALSE) +
    coord_cartesian(xlim=c(min(x2,x2_hat),
                           max(x2,x2_hat)),
                    ylim=c(min(x2,x2_hat),
                           max(x2,x2_hat))) +
    labs(x = expression("x"[2]),
         y = expression(hat(x)[2])) +
    theme(#axis.title.y = element_blank(),
          #axis.title.x = element_blank(),
          #axis.text = element_blank(),
          panel.background = element_rect(),
          panel.grid.major = element_line(size=1),
          panel.grid.minor = element_line(size=1),
          axis.ticks = element_line(colour="black", size = 1.5),
          panel.border = element_rect(colour = "black",
                                      fill = NA,
                                      size = 2.5),
          legend.text = element_blank(),
          legend.position = "none",
          aspect.ratio = 1) +
     annotate("label",
              label = paste("bias =", round(bias(x2, x2_hat),3),"\n",
                            "rmse =", round(rmse(x2, x2_hat),3),"\n",
                            "r =", round(cor(x2, x2_hat,
                                             method = "pearson"),3)),
              x=max(x2)-1,
              y=min(x2_hat)+0.5,
              size=4) +
    # add reference line
    geom_abline(intercept = 0, slope = 1,
                linetype = "dashed",
                size = 1,
                color = "red")

  # combine plots
  plot <- ggarrange(p1,p2,
                    ncol = 2, nrow = 1,
```

```
                      common.legend = TRUE,
                      legend = 'bottom')

# posterior hyperparameter joy plot (x1)
ggjoy_x1 <- ggplot(x1_long,
                   aes(x = x1_eval, y = factor(iter),
                       height = density, fill = factor(iter))) +
  geom_density_ridges(stat="identity",
                      alpha = 0.5, color = "white") +
  theme(axis.title.y = element_blank(),
        axis.title.x = element_blank(),
        axis.text = element_blank(),
        panel.background = element_rect(),
        panel.grid.major = element_line(size=1),
        panel.grid.minor = element_line(size=1),
        axis.ticks = element_line(colour="black", size = 1.5),
        panel.border = element_rect(colour = "black",
                                    fill = NA, size = 1.5),
        #legend.text = element_blank(),
        legend.position = "none",
        aspect.ratio = 1) +
        labs(x = "x1", y = "Iteration")

# posterior hyperparameter joy plot (x2)
ggjoy_x2 <- ggplot(x2_long,
                   aes(x = x2_eval, y = factor(iter),
                       height = density, fill = factor(iter))) +
  geom_density_ridges(stat="identity",
                      alpha = 0.5, color = "white") +
  theme(axis.title.y = element_blank(),
        axis.title.x = element_blank(),
        axis.text = element_blank(),
        panel.background = element_rect(),
        panel.grid.major = element_line(size=1),
        panel.grid.minor = element_line(size=1),
        axis.ticks = element_line(colour="black", size = 1.5),
        panel.border = element_rect(colour = "black",
                                    fill = NA, size = 1.5),
        #legend.text = element_blank(),
        legend.position = "none",
        aspect.ratio = 1) +
        labs(x = "x2", y = "Iteration")

ggjoy_x1x2 <- ggarrange(ggjoy_x1,ggjoy_x2, ncol = 2, nrow = 1)

# animated joy plot (x1)
ggjoy_x1_anim <- ggjoy_x1 +
  transition_states(factor(iter), transition_length = 1,
                    state_length = 1) + shadow_mark()

# animated joy plot (x2)
ggjoy_x2_anim <- ggjoy_x2 +
```

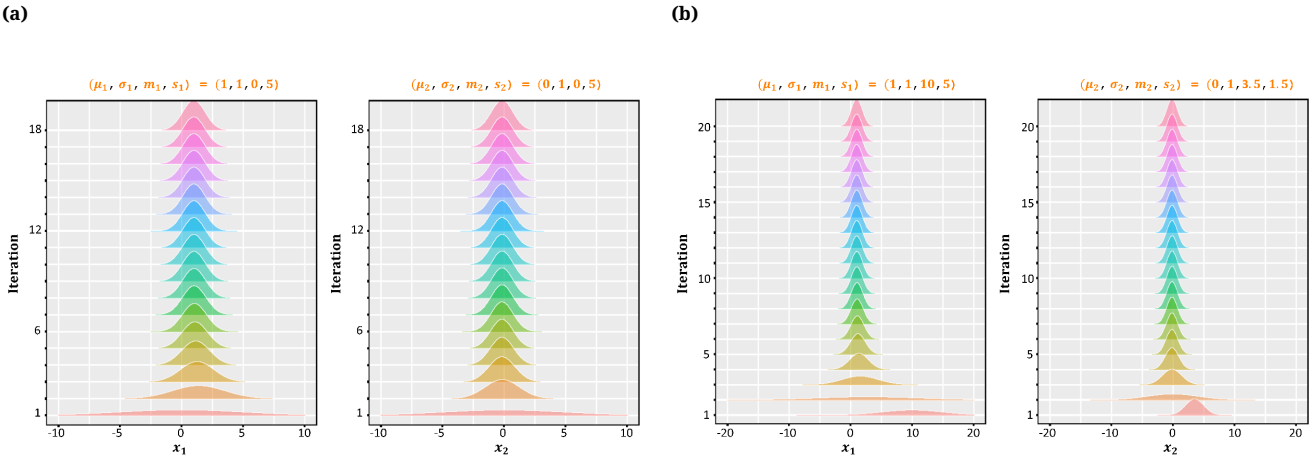
```
        transition_states(factor(iter), transition_length = 1,
                          state_length = 1) + shadow_mark()

      # return all results and plots
      return(list(results,
                  p1, p2, plot,
                  ggjoy_x1, ggjoy_x2, ggjoy_x1x2,
                  ggjoy_x1_anim , ggjoy_x2_anim))
    } # end else statement for plotting EML results

  } # end else if statement for EML

} # end estimation
```

Appendix B: Laplace-approximated posterior hyperparameters

We provide a graphical illustration of the estimated posterior hyperparameters obtained on each EML-iteration. Two possible results are shown in Figure 6: **(a)** where we simulate behavior using parameters sampled from a normal distribution with $(\mu_1, \sigma_1) = (1, 1)$ and $(\mu_2, \sigma_2) = (0, 1)$ for x_1 and x_2, respectively, and estimate our parameters – which generated our simulated behavior – with priors sampled from a normal distribution with $(m_{(1,2)}, s_{(1,2)}) = (0, 5)$ for both parameters, while **(b)** we simulate behavior using parameters sampled from the same normal distribution for each parameter as in (a), but now estimate our parameters with relatively wide priors sampled from a normal distribution with $(m_1, s_1) = (10, 5)$ and $(m_2, s_2) = (3.5, 1.5)$ for x_1 and x_2, respectively. In both cases, we see convergence of the posterior hyperparameters to the distribution we had used to simulate behavior, highlighting the performance of the EML method in closely recovering our behavioral parameters.

```
# Simulate data
data_sim <- simulate_twochoiceRL(mean_x = c(μ1,μ2),
                                 stdev_x = c(σ1,σ2) )

# Estimate data
estimate_twochoiceRL(data = data_sim, method = "eml",
                     prior_mean = c(m1,m2),
                     prior_sd = c(s1,s2),
                     plot = TRUE)
```

Figure 6 ■ Laplace-approximated posterior hyperparameters per EML iteration.



Appendix C: Softmax to turn numbers into probabilities

The softmax function is known for modeling an individual's choice among a set of choices. In the classic two-armed bandit task, we have some expectation (or evidence), $\vec{Q} = \{Q_A, Q_B\}$, that an individual will choose a particular machine: $\vec{M} = \{M_A, M_B\}$ with some probability. Larger values for $\vec{Q}$ indicate higher expectation (or more evidence) that a particular chosen machine will maximize reward. So, we need to convert these expectations into choice probabilities for each machine.

In laymen terms, the softmax can simply be thought of as a way to convert numbers into probabilities. It divides each element of $\vec{Q}$ by the sum of all its elements:

$$P(M) = \frac{e^{\beta \vec{Q}}}{\sum e^{\beta \vec{Q}}} \tag{C.1}$$

where $\vec{Q}$ represents a row vector, $\left[Q_A, Q_B\right]$, containing the expected values for machine A and machine B. Therefore, $\beta\vec{Q}$ represents the scaled vector, $\left[\beta Q_A, \beta Q_B\right]$, where β acts to scale the impact of the choice probability of a machine, P(M).

It may not be intuitive now, but if we code the softmax just as (C.1) we will run into issues of numerical instability. We know that β ranges from [0, ∞) so there is a possibility for β to be large and increasingly scale $\vec{Q}$ which would result in a very large exponential value ($e^{10000} = \infty$), and we want values between [0,1] as per the definition of probability.

Fortunately, the softmax identity trick (Goodfellow, Bengio & Courville, 2016) - subtract the maximum value from each element - will avoid this numerical instability:

$$\begin{aligned}
P(M) &= \frac{e^{\vec{x}}}{\sum e^{\vec{x}}} && (\text{let } \vec{x} = \beta\vec{Q}) \\
&= \frac{e^{\vec{x} - max(\vec{x})}}{\sum e^{\vec{x} - max(\vec{x})}} && (\text{softmax identity}) \\
&= \frac{\frac{e^{\vec{x}}}{e^{max(\vec{x})}}}{\frac{\sum e^{\vec{x}}}{e^{max(\vec{x})}}} && (e^{a-b} = \tfrac{e^a}{e^b}) \\
&= \frac{e^{\vec{x}}}{e^{max(\vec{x})}} * \frac{e^{max(\vec{x})}}{\sum e^{\vec{x}}} && (\text{invert and multiply}) \\
&= \frac{e^{\vec{x}}}{\sum e^{\vec{x}}} && (\text{simplify})
\end{aligned}$$

This demonstrates that we can preserve the original softmax function while overcoming numerical instability.

Appendix D: Log-likelihood, Gradient and Hessian derivations

Introduction. Let us revisit our model with two parameters - inverse temperature, β, and learning rate α - that we aim to estimate. We know that the domain of these model parameters are [0,∞] and [0,1], respectively. However, given an unconstrained optimization approach (i.e., trust region), we need to perform our estimation on unconstrained parameters. Thus, our motivation behind reparameterization of parameters using x_1 and x_2:

$$\beta = e^{x_1} \tag{D.1}$$

$$\alpha = \frac{1}{1 + e^{-x_2}} \tag{D.2}$$

This reparameterization allows our parameters - x_1 and x_2 - to have an unconstrained domain of [-∞, ∞] while preserving the appropriate domains of β and α. We now introduce the softmax choice function and Q-learning rule formula for choosing either arm A or arm B of a slot machine:

$$P(A) = \frac{e^{Q_1}}{e^{Q_1} + e^{Q_2}} \tag{D.3}$$

$$Q_{A_t} = Q_{A_{t-1}} + \alpha(r_{t-1} - Q_{A_{t-1}}) \tag{D.4}$$

$$P(B) = \frac{e^{Q_2}}{e^{Q_1} + e^{Q_2}} \tag{D.5}$$

$$Q_{B_t} = Q_{B_{t-1}} + \alpha(r_{t-1} - Q_{B_{t-1}}) \tag{D.6}$$

where β and α have been reparameterized and $Q_1 = \beta Q_A$ and $Q_2 = \beta Q_B$. Equations (D.3) and (D.4) represent the softmax choice function and Q-learning rule for a participants' choice of arm A, respectively. Likewise, equations (D.5) and (D.6) represent the choice of arm B. For the purpose of explaining the remaining derivations, we will perform the derivations on the basis of a particular choice by a participant (i.e., the probability of choosing arm A). This means we will focus on equations (D.3) and (D.4). Let's begin by algebraically converting equation (D.4) into one indexed by 1 (for choosing arm A):

$$\begin{aligned} Q_{A_t} &= Q_{A_{t-1}} + \alpha(r_{t-1} - Q_{A_{t-1}}) && \text{(from equation 4)} \\ \beta Q_{A_t} &= \beta Q_{A_{t-1}} + \alpha(\beta r_{t-1} - \beta Q_{A_{t-1}}) && \text{(multiply by } \beta\text{)} \\ Q_{1_t} &= Q_{1_{t-1}} + \alpha(\beta r_{t-1} - Q_{1_{t-1}}) && (Q_1 = \beta Q_A) \end{aligned}$$

Hence,

$$Q_{1_t} = Q_{1_{t-1}} + \alpha(\beta r_{t-1} - Q_{1_{t-1}}) \tag{D.7}$$

Log-likelihood. Now that we have setup the general model we can begin performing our derivations. We will first start with the derivation of the log-likelilhood (LL) of equation (D.3):

$$\begin{aligned} P(A) &= \frac{e^{Q_1}}{e^{Q_1} + e^{Q_2}} && \text{(from equation 3)} \\ \log(P(A)) &= \log(\frac{e^{Q_1}}{e^{Q_1} + e^{Q_2}}) && \text{(}log\text{ both sides)} \\ &= \log(e^{Q_1}) - \log(e^{Q_1} + e^{Q_2}) && \text{(}log\text{ quotient rule)} \\ &= Q_1 - \log(e^{Q_1} + e^{Q_2}) && (\log(e^x) = x) \end{aligned}$$

Thus,

$$LL = \log(P(A)) = Q_1 - \log(e^{Q_1} + e^{Q_2}) \tag{D.8}$$

Gradient. Given our log-likelihood, we are ready to compute the gradient of the LL with respect to each of our parameters - x_1 and x_2 - as symbolized by $\frac{\partial LL}{\partial x_1}$ and $\frac{\partial LL}{\partial x_2}$, respectively :

Starting with $\frac{\partial LL}{\partial x_1}$:

$$\begin{aligned} LL &= Q_1 - \log(e^{Q_1} + e^{Q_2}) && \text{(from equation 8)} \\ \frac{\partial LL}{\partial x_1} &= \frac{\partial Q_1}{\partial x_1} - \left[\frac{1}{e^{Q_1} + e^{Q_2}}\right]\left[\frac{\partial Q_1}{\partial x_1}e^{Q_1} + \frac{\partial Q_2}{\partial x_1}e^{Q_2}\right] && \text{(partial derivative w.r.t } x_1) \\ &= \frac{\partial Q_1}{\partial x_1} - \left[P(A)\frac{\partial Q_1}{\partial x_1} + P(B)\frac{\partial Q_2}{\partial x_1}\right] && \text{(from equation 3 and 5)} \end{aligned}$$

from which we get

$$\frac{\partial LL}{\partial x_1} = \frac{\partial Q_1}{\partial x_1} - \left[P(A)\frac{\partial Q_1}{\partial x_1} + P(B)\frac{\partial Q_2}{\partial x_1}\right] \tag{D.9}$$

Second, we need to solve for $\frac{\partial Q_1}{\partial x_1}$ in equation (D.9). To accomplish this we will refer to equation (D.7) and differentiate with respect to x_1:

Solving $\frac{\partial Q_1}{\partial x_1}$:

$$\begin{aligned} Q_{1_t} &= Q_{1_{t-1}} + \alpha(\beta r_{t-1} - Q_{1_{t-1}}) && \text{(from equation 7)} \\ \frac{\partial Q_{1_t}}{\partial x_1} &= \frac{\partial Q_{1_{t-1}}}{\partial x_1} + \alpha r_{t-1}\frac{\partial \beta}{\partial x_1} - \alpha\frac{\partial Q_{1_{t-1}}}{\partial x_1} && \text{(partial derivative w.r.t } x_1) \\ &= (1-\alpha)\frac{\partial Q_{1_{t-1}}}{\partial x_1} + \alpha r_{t-1}\frac{\partial \beta}{\partial x_1} && \text{(simplify using common factor)} \end{aligned}$$

Thus,

$$\frac{\partial Q_{1_t}}{\partial x_1} = (1-\alpha)\frac{\partial Q_{1_{t-1}}}{\partial x_1} + \alpha r_{t-1}\frac{\partial \beta}{\partial x_1} \tag{D.10}$$

The last step needed to evaluate equation (D.9) is to evaluate the term, $\frac{\partial \beta}{\partial x_1}$, from equation (D.10):

Getting $\frac{\partial \beta}{\partial x_1}$:

$$\begin{aligned} \beta &= e^{x_1} && \text{(from equation 1)} \\ \frac{\partial \beta}{\partial x_1} &= \frac{\partial e^{x_1}}{\partial x_1} && \text{(partial derivative w.r.t } x_1) \\ &= e^{x_1} && (\tfrac{\partial}{\partial x}e^x = e^x) \\ &= \beta && \text{(from equation 1)} \end{aligned}$$

Consequently,

$$\frac{\partial \beta}{\partial x_1} = \beta \tag{D.11}$$

Finally, we expand equation (D.9) using equations (D.10) and (D.11) to obtain the gradient of the LL with respect to x_1.

We now perform a similar derivation procedure for the gradient of the LL with respect to the other parameter, x_2:

Starting with $\frac{\partial LL}{\partial x_2}$:

$$\begin{aligned} LL &= Q_1 - \log(e^{Q_1} + e^{Q_2}) && \text{(from equation 8)} \\ \frac{\partial LL}{\partial x_2} &= \frac{\partial Q_1}{\partial x_2} - \left[\frac{1}{e^{Q_1} + e^{Q_2}}\right]\left[\frac{\partial Q_1}{\partial x_2}e^{Q_1} + \frac{\partial Q_2}{\partial x_2}e^{Q_2}\right] && \text{(partial derivative w.r.t } x_2) \\ &= \frac{\partial Q_1}{\partial x_2} - \left[P(A)\frac{\partial Q_1}{\partial x_2} + P(B)\frac{\partial Q_2}{\partial x_2}\right] && \text{(from equation 3 and 5)} \end{aligned}$$

we get

$$\frac{\partial LL}{\partial x_2} = \frac{\partial Q_1}{\partial x_2} - \left[P(A)\frac{\partial Q_1}{\partial x_2} + P(B)\frac{\partial Q_2}{\partial x_2}\right] \tag{D.12}$$

To solve for $\frac{\partial Q_1}{\partial x_2}$ in equation (D.12), we will refer to equation (D.7) and differentiate with respect to x_2:

Solving $\frac{\partial Q_1}{\partial x_2}$:

$$\begin{aligned}
Q_{1_t} &= Q_{1_{t-1}} + \alpha(\beta r_{t-1} - Q_{1_{t-1}}) && \text{(from equation 7)}\\
\frac{\partial Q_{1_t}}{\partial x_2} &= \frac{\partial Q_{1_{t-1}}}{\partial x_2} + \alpha\left[-\frac{\partial Q_{1_{t-1}}}{\partial x_2}\right] + (\beta r_{t-1} - Q_{1_{t-1}})\left[\frac{\partial \alpha}{\partial x_2}\right] && \text{(partial derivative w.r.t } x_2)\\
&= (1-\alpha)\frac{\partial Q_{1_{t-1}}}{\partial x_2} + (\beta r_{t-1} - Q_{1_{t-1}})\frac{\partial \alpha}{\partial x_2} && \text{(simplify using common factor)}
\end{aligned}$$

we obtain

$$\frac{\partial Q_{1_t}}{\partial x_2} = (1-\alpha)\frac{\partial Q_{1_{t-1}}}{\partial x_2} + (\beta r_{t-1} - Q_{1_{t-1}})\frac{\partial \alpha}{\partial x_2} \tag{D.13}$$

The last step needed to evaluate equation (D.12) is to evaluate the term, $\frac{\partial \alpha}{\partial x_2}$, from equation (D.13):

Finally, $\frac{\partial \alpha}{\partial x_2}$:

$$\begin{aligned}
\alpha &= \frac{1}{1+e^{-x_2}} && \text{(from equation 2)}\\
&= (1+e^{-x_2})^{-1} && \text{(rewritten form)}\\
\frac{\partial \alpha}{\partial x_2} &= -(1+e^{-x_2})^{-2}(-e^{-x_2}) && \text{(partial derivative w.r.t } x_2)\\
&= (1+e^{-x_2})^{-2}(e^{-x_2}) && \text{(divide by -1)}\\
&= (\alpha)^2(e^{-x_2}) && \text{(since } \alpha = (1+e^{-x_2})^{-1})\\
&= (\alpha)^2\left[\frac{1-\alpha}{\alpha}\right] && \text{(solve for } e^{-x_2} \text{ in equation 2)}\\
&= \alpha(1-\alpha) && \text{(simplify)}
\end{aligned}$$

Consequently,

$$\frac{\partial \alpha}{\partial x_2} = \alpha(1-\alpha) \tag{D.14}$$

Finally, we expand equation (D.12) using equations (D.13) and (D.14) to obtain the gradient of the LL with respect to x_2.

Hessian. Now that we have completed our gradient computation, we are ready to compute the Hessian of the LL with respect to each of our parameters - x_1 and x_2 - as symbolized by $\partial^2 LL/\partial x_1^2$, $\partial^2 LL/(\partial x_1 \partial x_2)$, $\partial^2 LL/(\partial x_2 \partial x_1)$ and $\partial^2 LL/\partial x_2^2$. Let us first look at our computed gradients where we replace $P(A)$ and $P(B)$ with e^{LL} and 1-e^{LL}, respectively, from equations (D.9) and (D.12) given that we focus on particpants' choice of choosing arm A:

$$\frac{\partial LL}{\partial x_1} = \frac{\partial Q_1}{\partial x_1} - \left[e^{LL}\frac{\partial Q_1}{\partial x_1} + (1-e^{LL})\frac{\partial Q_2}{\partial x_1}\right] \tag{D.15}$$

$$\frac{\partial LL}{\partial x_2} = \frac{\partial Q_1}{\partial x_2} - \left[e^{LL}\frac{\partial Q_1}{\partial x_2} + (1-e^{LL})\frac{\partial Q_2}{\partial x_2}\right] \tag{D.16}$$

Note that we can replace the probabilities with e^{LL} because $LL = \log(P(A))$ so if we take the exponential from both sides we get $P(A) = e^{LL}$ and since $P(B) = 1 - P(A)$ we get $P(B) = 1 - e^{LL}$.

Using equations (D.15) and (D.16), we can compute the Hessians. Let us begin with the Hessian of LL w.r.t x_1:

Solve for $\frac{\partial^2 LL}{\partial x_1^2}$:

$$\begin{aligned}
\frac{\partial LL}{\partial x_1} &= \frac{\partial Q_1}{\partial x_1} - \left[e^{LL}\frac{\partial Q_1}{\partial x_1} + (1-e^{LL})\frac{\partial Q_2}{\partial x_1}\right] && \text{(from equation 15)}\\
\frac{\partial^2 LL}{\partial x_1^2} &= \frac{\partial^2 Q_1}{\partial x_1^2} - \left[e^{LL}\frac{\partial^2 Q_1}{\partial x_1^2} + \frac{\partial Q_1}{\partial x_1}\left(\frac{\partial LL}{\partial x_1}e^{LL}\right)\right]\\
&\quad - \left[(1-e^{LL})\frac{\partial^2 Q_2}{\partial x_1^2} + \frac{\partial Q_2}{\partial x_1}\left(-\frac{\partial LL}{\partial x_1}e^{LL}\right)\right] && \text{(second partial derivative w.r.t } x_1\text{)}\\
&= \frac{\partial^2 Q_1}{\partial x_1^2} - \left[e^{LL}\frac{\partial^2 Q_1}{\partial x_1^2} + (1-e^{LL})\frac{\partial^2 Q_2}{\partial x_1^2}\right]\\
&\quad - \frac{\partial LL}{\partial x_1}e^{LL}\left(\frac{\partial Q_1}{\partial x_1} - \frac{\partial Q_2}{\partial x_1}\right) && \text{(expand and simplify)}
\end{aligned}$$

resulting in

$$\frac{\partial^2 LL}{\partial x_1^2} = \frac{\partial^2 Q_1}{\partial x_1^2} - \left[e^{LL}\frac{\partial^2 Q_1}{\partial x_1^2} + (1-e^{LL})\frac{\partial^2 Q_2}{\partial x_1^2}\right] - \frac{\partial LL}{\partial x_1}e^{LL}\left(\frac{\partial Q_1}{\partial x_1} - \frac{\partial Q_2}{\partial x_1}\right) \tag{D.17}$$

Now, we need to solve for $\frac{\partial^2 Q_1}{\partial x_1^2}$ in equation (D.17). To accomplish this we will refer to equation (D.10) and differentiate with respect to x_1:

Solve for $\frac{\partial^2 Q_1}{\partial x_1^2}$:

$$\begin{aligned}
\frac{\partial Q_{1_t}}{\partial x_1} &= (1-\alpha)\frac{\partial Q_{1_{t-1}}}{\partial x_1} + \alpha r_{t-1}\frac{\partial \beta}{\partial x_1} && \text{(from equation 10)}\\
\frac{\partial^2 Q_{1_t}}{\partial x_1^2} &= (1-\alpha)\frac{\partial^2 Q_{1_{t-1}}}{\partial x_1^2} + \alpha r_{t-1}\beta && \text{(second partial derivative w.r.t } x_1\text{)}
\end{aligned}$$

resuling in

$$\frac{\partial^2 Q_{1_t}}{\partial x_1^2} = (1-\alpha)\frac{\partial^2 Q_{1_{t-1}}}{\partial x_1^2} + \alpha r_{t-1}\beta \tag{D.18}$$

Finally, we expand equation (D.17) using equation (D.22) to obtain the Hessian of the LL with respect to x_1.

We will now derive the remaining Hessian components - $\frac{\partial^2 LL}{\partial x_1 \partial x_2}$, $\frac{\partial^2 LL}{\partial x_2 \partial x_1}$ and $\frac{\partial^2 LL}{\partial x_2^2}$.

Solve for $\frac{\partial^2 LL}{\partial x_1 \partial x_2}$:

$$\begin{aligned}
\frac{\partial^2 LL}{\partial x_1 \partial x_2} &= \frac{\partial^2 Q_1}{\partial x_1 \partial x_2} - \left[e^{LL}\frac{\partial^2 Q_1}{\partial x_1 \partial x_2} + \frac{\partial Q_1}{\partial x_1}\left(\frac{\partial LL}{\partial x_2}e^{LL}\right)\right]\\
&\quad - \left[(1-e^{LL})\frac{\partial^2 Q_2}{\partial x_1 \partial x_2} + \frac{\partial Q_2}{\partial x_1}\left(-\frac{\partial LL}{\partial x_2}e^{LL}\right)\right] && \text{(partial derivative of equation 15 w.r.t } x_2\text{)}
\end{aligned}$$

resulting in

$$\frac{\partial^2 LL}{\partial x_1 \partial x_2} = \frac{\partial^2 Q_1}{\partial x_1 \partial x_2} - \left[e^{LL}\frac{\partial^2 Q_1}{\partial x_1 \partial x_2} + (1-e^{LL})\left(\frac{\partial^2 Q_2}{\partial x_1 \partial x_2}\right)\right] - \frac{\partial LL}{\partial x_2}e^{LL}\left(\frac{\partial Q_1}{\partial x_1} - \frac{\partial Q_2}{\partial x_1}\right) \tag{D.19}$$

Now, we need to solve for $\frac{\partial^2 Q_1}{\partial x_1 \partial x_2}$ in equation (D.19). To accomplish this we will refer to equation (D.10) and differentiate with respect to x_2:

Solve for $\frac{\partial^2 Q_1}{\partial x_1 \partial x_2}$:

$$\frac{\partial Q_{1_t}}{\partial x_1} = (1-\alpha)\frac{\partial Q_{1_{t-1}}}{\partial x_1} + \alpha r_{t-1}\frac{\partial \beta}{\partial x_1} \qquad \text{(from equation 10)}$$

$$\frac{\partial^2 Q_1}{\partial x_1 \partial x_2} = (1-\alpha)\frac{\partial^2 Q_1}{\partial x_1 \partial x_2} - \frac{\partial \alpha}{\partial x_2}\left(\frac{\partial Q_{1_{t-1}}}{\partial x_1}\right) + \frac{\partial \alpha}{\partial x_2} r_{t-1}\beta \qquad \text{(second partial derivative w.r.t } x_2\text{)}$$

$$= (1-\alpha)\frac{\partial^2 Q_1}{\partial x_1 \partial x_2} - (\alpha(1-\alpha))\frac{\partial Q_{1_{t-1}}}{\partial x_1} + (\alpha(1-\alpha))r_{t-1}\beta \qquad (\tfrac{\partial \alpha}{\partial x_2} = \alpha(1\text{-}\alpha))$$

resulting in

$$\frac{\partial^2 Q_1}{\partial x_1 \partial x_2} = (1-\alpha)\frac{\partial^2 Q_1}{\partial x_1 \partial x_2} - (\alpha(1-\alpha))\left[\frac{\partial Q_{1_{t-1}}}{\partial x_1} - r_{t-1}\beta\right] \qquad \text{(D.20)}$$

Finally, we expand equation (D.19) using equation (D.20) to obtain the Hessian of the LL of x_1 with respect to x_2.

Solve for $\frac{\partial^2 LL}{\partial x_2^2}$:

$$\frac{\partial LL}{\partial x_2} = \frac{\partial Q_1}{\partial x_2} - \left[e^{LL}\frac{\partial Q_1}{\partial x_2} + (1-e^{LL})\frac{\partial Q_2}{\partial x_2}\right] \qquad \text{(from equation 16)}$$

$$\frac{\partial^2 LL}{\partial x_2^2} = \frac{\partial^2 Q_1}{\partial x_2^2} - \left[e^{LL}\frac{\partial^2 Q_1}{\partial x_2^2} + \frac{\partial Q_1}{\partial x_2}\left(\frac{\partial LL}{\partial x_2}e^{LL}\right)\right] - \left[(1-e^{LL})\frac{\partial^2 Q_2}{\partial x_2^2} + \frac{\partial Q_2}{\partial x_2}\left(-\frac{\partial LL}{\partial x_2}e^{LL}\right)\right] \qquad \text{(second partial derivative w.r.t } x_2\text{)}$$

$$= \frac{\partial^2 Q_1}{\partial x_2^2} - \left[e^{LL}\frac{\partial^2 Q_1}{\partial x_2^2} + (1-e^{LL})\frac{\partial^2 Q_2}{\partial x_2^2}\right] - \frac{\partial LL}{\partial x_2}e^{LL}\left(\frac{\partial Q_1}{\partial x_2} - \frac{\partial Q_2}{\partial x_2}\right) \qquad \text{(expand and simplify)}$$

resulting in

$$\frac{\partial^2 LL}{\partial x_2^2} = \frac{\partial^2 Q_1}{\partial x_2^2} - \left[e^{LL}\frac{\partial^2 Q_1}{\partial x_2^2} + (1-e^{LL})\frac{\partial^2 Q_2}{\partial x_2^2}\right] - \frac{\partial LL}{\partial x_2}e^{LL}\left(\frac{\partial Q_1}{\partial x_2} - \frac{\partial Q_2}{\partial x_2}\right) \qquad \text{(D.21)}$$

Now, we need to solve for $\frac{\partial^2 Q_1}{\partial x_2^2}$ in equation (D.21). To accomplish this we will refer to equation (D.10) and differentiate with respect to x_2:

Solve for $\frac{\partial^2 Q_1}{\partial x_2^2}$:

$$\frac{\partial Q_{1_t}}{\partial x_2} = (1-\alpha)\frac{\partial Q_{1_{t-1}}}{\partial x_2} + (\beta r_{t-1} - Q_{1_{t-1}})\frac{\partial \alpha}{\partial x_2} \qquad \text{(from equation 13)}$$

$$\frac{\partial^2 Q_{1_t}}{\partial x_2^2} = (1-\alpha)\frac{\partial^2 Q_{1_{t-1}}}{\partial x_2^2} + \left(-\frac{\partial \alpha}{\partial x_2}\right)\frac{\partial Q_{1_{t-1}}}{\partial x_2} + (\beta r_{t-1} - Q_{1_{t-1}})\frac{\partial^2 \alpha}{\partial x_2^2} + \left(-\frac{\partial Q_{1_{t-1}}}{\partial x_2}\right)\frac{\partial \alpha}{\partial x_2} \qquad \text{(second partial derivative w.r.t } x_2\text{)}$$

$$= (1-\alpha)\frac{\partial^2 Q_{1_{t-1}}}{\partial x_2^2} - 2(\alpha(1-\alpha))\frac{\partial Q_{1_{t-1}}}{\partial x_2} + (1-2\alpha)\beta r_{t-1} - Q_{1_{t-1}} \qquad \text{(expand and simplify)}$$

resulting in

$$\frac{\partial^2 Q_{1_t}}{\partial x_2^2} = (1-\alpha)\frac{\partial^2 Q_{1_{t-1}}}{\partial x_2^2} - 2(\alpha(1-\alpha))\frac{\partial Q_{1_{t-1}}}{\partial x_2} + (1-2\alpha)\beta r_{t-1} - Q_{1_{t-1}} \qquad \text{(D.22)}$$

Finally, we expand equation (D.21) using equation (D.22) to obtain the Hessian of the LL with respect to x_2.

Solve for $\frac{\partial^2 LL}{\partial x_2 \partial x_1}$:

$$\begin{aligned}\frac{\partial^2 LL}{\partial x_2 \partial x_1} &= \frac{\partial^2 Q_1}{\partial x_2 \partial x_1} - \left[e^{LL}\frac{\partial^2 Q_1}{\partial x_2 \partial x_1} + \frac{\partial Q_1}{\partial x_2}\left(\frac{\partial LL}{\partial x_1}e^{LL}\right)\right] \\ &\quad - \left[(1-e^{LL})\frac{\partial^2 Q_2}{\partial x_2 \partial x_1} + \frac{\partial Q_2}{\partial x_2}\left(-\frac{\partial LL}{\partial x_1}e^{LL}\right)\right] \qquad \text{(partial derivative of equation 15 w.r.t } x_1\text{)}\end{aligned}$$

resutling in

$$\frac{\partial^2 LL}{\partial x_2 \partial x_1} = \frac{\partial^2 Q_1}{\partial x_2 \partial x_1} - \left[e^{LL}\frac{\partial^2 Q_1}{\partial x_2 \partial x_1} + (1-e^{LL})\left(\frac{\partial^2 Q_2}{\partial x_2 \partial x_1}\right)\right] - \frac{\partial LL}{\partial x_1}e^{LL}\left(\frac{\partial Q_1}{\partial x_2} - \frac{\partial Q_2}{\partial x_2}\right) \tag{D.23}$$

To solve for $\frac{\partial^2 Q_1}{\partial x_2 \partial x_1}$ in equation (D.23), we will refer to equation (D.10) and differentiate with respect to x_2:

Step $\frac{\partial^2 Q_1}{\partial x_2 \partial x_1}$:

$$\begin{aligned}\frac{\partial Q_{1_t}}{\partial x_2} &= (1-\alpha)\frac{\partial Q_{1_{t-1}}}{\partial x_2} + (\beta r_{t-1} - Q_{1_{t-1}})\frac{\partial \alpha}{\partial x_2} && \text{(from equation 13)} \\ \frac{\partial^2 Q_{1_t}}{\partial x_2 \partial x_1} &= (1-\alpha)\frac{\partial^2 Q_{1_{t-1}}}{\partial x_2 \partial x_1} + (\beta r_{t-1} - Q_{1_{t-1}})\frac{\partial^2 \alpha}{\partial x_2 \partial x_1} \\ &\quad + (\frac{\partial \beta}{\partial x_1}r_{t-1} - \frac{\partial Q_{1_{t-1}}}{\partial x_1})\frac{\partial \alpha}{\partial x_2} && \text{(second partial derivative w.r.t } x_1\text{)} \\ &= (1-\alpha)\frac{\partial^2 Q_{1_{t-1}}}{\partial x_2 \partial x_1} + (\alpha(1-\alpha))\left[\beta r_{t-1} - \frac{\partial Q_{1_{t-1}}}{\partial x_1}\right] && \text{(expand and simplify)}\end{aligned}$$

resulting in

$$\frac{\partial^2 Q_1}{\partial x_2 \partial x_1} = (1-\alpha)\frac{\partial^2 Q_{1_{t-1}}}{\partial x_2 \partial x_1} + (\alpha(1-\alpha))\left[\beta r_{t-1} - \frac{\partial Q_{1_{t-1}}}{\partial x_1}\right] \tag{D.24}$$

Finally, we expand equation (D.23) using equation (D.24) to obtain the Hessian of the LL of x_2 with respect to x_1.

Open practices

The *Open Material* badge was earned because supplementary material(s) are available on github.com/psuthaharan/twochoiceRL.

Citation

Suthaharan, P., Corlett, P. R., & Ang, Y.-S. (2021). Computational modeling of behavioral tasks: An illustration on a classic reinforcement learning paradigm. *The Quantitative Methods for Psychology*, *17*(2), 105–140. doi:10.20982/tqmp.17.2.p105

Received: 24/06/2020 ~ Accepted: 04/06/2021