Task recap

As you might remember, this task has two aims. The first aim is to measure whether participants adapt how they learn to the environment, specifically whether they learn faster (have a higher 'learning rate', see below) when the environment changes quickly and whether they learn more slowly when the environment is more stable. The second aim is to see how stress affects how you adapt to your environment.

For this, participants perform a reinforcement learning task. On each trial, they have to choose between two options (in blue and in green) on the figure below. They have to make this choice based on two attributes: how likely each option is to lead to a reward and how high this reward would be. The size of the potential reward is displayed on top of the option (i.e. the number written on top of the option). The probability of a reward is what participants have to infer through experience, i.e. learn: as you can see in the figure below, for the first 90 trials of the experiment (block 1, 'stable block') the probability of reward is 75%; for the next 90 trials (block 2, 'volatile block') the probability changed back and forth between 20% and 80%. For example, a probability of 80% meant that if participants picked option A (blue option), they would get a reward 8 out of 10 times and if they picked option B (green option), they would get a reward 2 out of 10 times. I.e. the probability described how likely the reward was to be linked with option 1 vs. option 2 on a specific trial.

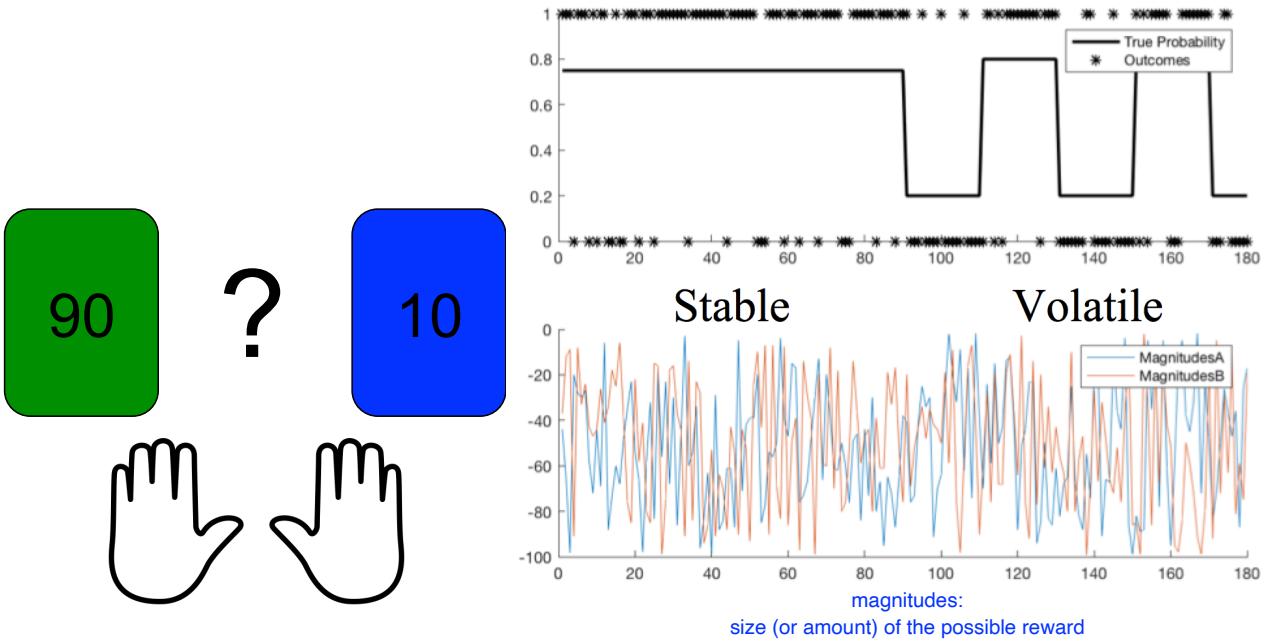


Figure 1. (Left) Task design: On each trial participants had to choose between two options based on learnt reward probabilities associated with the two colours and shown magnitudes of reward (black numbers). (Upper Right) an illustration of the true reward probability and actual outcomes of an example schedule. As one half of the schedule the true probability does not change we call it stable. The other half has many changes in the true probability and we therefore label it volatile. (Lower Right) the magnitudes for each option are drawn randomly, shown explicitly on very trial and vary independently of the learnt probabilities (bottom).

The first key manipulation of the task was that there were the two types of blocks: in the 'stable' block, contingencies did not change, i.e. they were stable. In contrast, in the 'volatile' block, contingencies reversed frequently, i.e. they were volatile. What we would predict then is that people should learn more slowly (i.e. have a lower learning rate, see next section for explanation) in the stable block and faster in the volatile block. [if anybody is interested in how you could learn optimally with a completely flexible learning rate using a Bayesian observer, sess Behrens et al., 2007 Nature Neuroscience]

The second key manipulation of the task was that participants were randomly assigned to one of two groups: in the control group, participants heard neutral sounds at random points while performing the task. In the experimental group ('stress group'), participants heard unpleasant sounds at random points during

the task. To check that this manipulation worked, i.e. made participants feel somewhat stressed, we asked them every 20 trials to rate how they felt.

Our primary behavioural measure was the participants choices in every trial (option 1 or 2) (our 'dependent measure'). The 'independent measures' is the reward magnitudes as well as the reward feedback on every trial, which the participants could use to learn about the reward probabilities of both options.

Reinforcement learning model

As we saw last week, reinforcement learning can be used to describe how you might update predictions in your mind (e.g. predictions about how likely it is that you will receive a reward if you choose a stimulus) based on experience. In this specific task, participants were trying to predict how likely they were to get a positive outcome if they choose one option or the other; and the 'experience' therefore was whether on each trial they received a positive outcome or not when picking a specific option.

We can now build a simple **computational model** describing the learning process going on in participants' brains. This specific computational model consists of a series of simple equations:

[1] $delta_{t+1} = Outcome_t - Prediction_t$

This means that on each trial the model compares the outcome (which is 1 if the participant receives a reward and zero if there is no reward) to the prediction (which is a value between 0 and 1, e.g. 0.5 in the beginning of the experiment when you don't yet have any experience and assume that reward and no reward are equally likely). This difference is then called the '(reward) prediction errors' (RPE). The lower script 't' refers to the specific trial in question, i.e. t=1 is the first trial etc.

[2] $Prediction_{t+1} = Prediction_t + alpha * delta_t$

This second equation means: after seeing an outcome (on trial t), the model updates the prediction based on the prediction error computed in [1] to form a new prediction for the next trial (trial t+1). The learning rate (often called α) describes how quickly the model learns: the higher the learning rate, the faster. In other words, higher learning rates weigh the most recent experience more heavily than low learning rates. (You can understand this by writing down the full expression for predictions on the next for e.g. a learning rate of 0 and a learning rate of 1). The learning rate is a so-called 'free parameter'. This simply means that it is "free" to be fit to the data, rather than being set to a theoretical specific value. In the next session, we will fit a reinforcement learner to real data and try, for each person, to find the learning rate that best describes their behavior.

You might wonder why there is only one and not two Predictions, as there are two distinct options. This is because only one option is rewarded in any given trial, and participants find out in every trial which option was rewarded. This is why they can just learn one Prediction/probability estimate, as the inverse of one estimate has to be true for the other option (prob1=1-prob2).

We will now code this in Matlab. Open the script 'Session2_Part1'. Change the line 'MatabCodePath' (#1) to the path where you have stored the folder 'MatlabCode' and read the rest of the comments (in green, marked with '%' in the beginning of each row) that explain what this cell of the script does.

Question: Describe what you see in the figure below (or plotted in matlab). What is the black line and what are the stars? And (looking at the relevant variables in the Matlab command window) how do they relate to the variables that we defined in the cell 'Load files and settings' from the structure 'trialVariables'.

_____

_____

Now we can examine how the predictions of the learner change when the learning rate is changed. For this, change the learning rate to 0.6 and rerun the code with the reinforcement learner and the code that made

the figure. What can you see, compared to the initial lower learning rate (e.g. .3 or .1)?

_____

_____

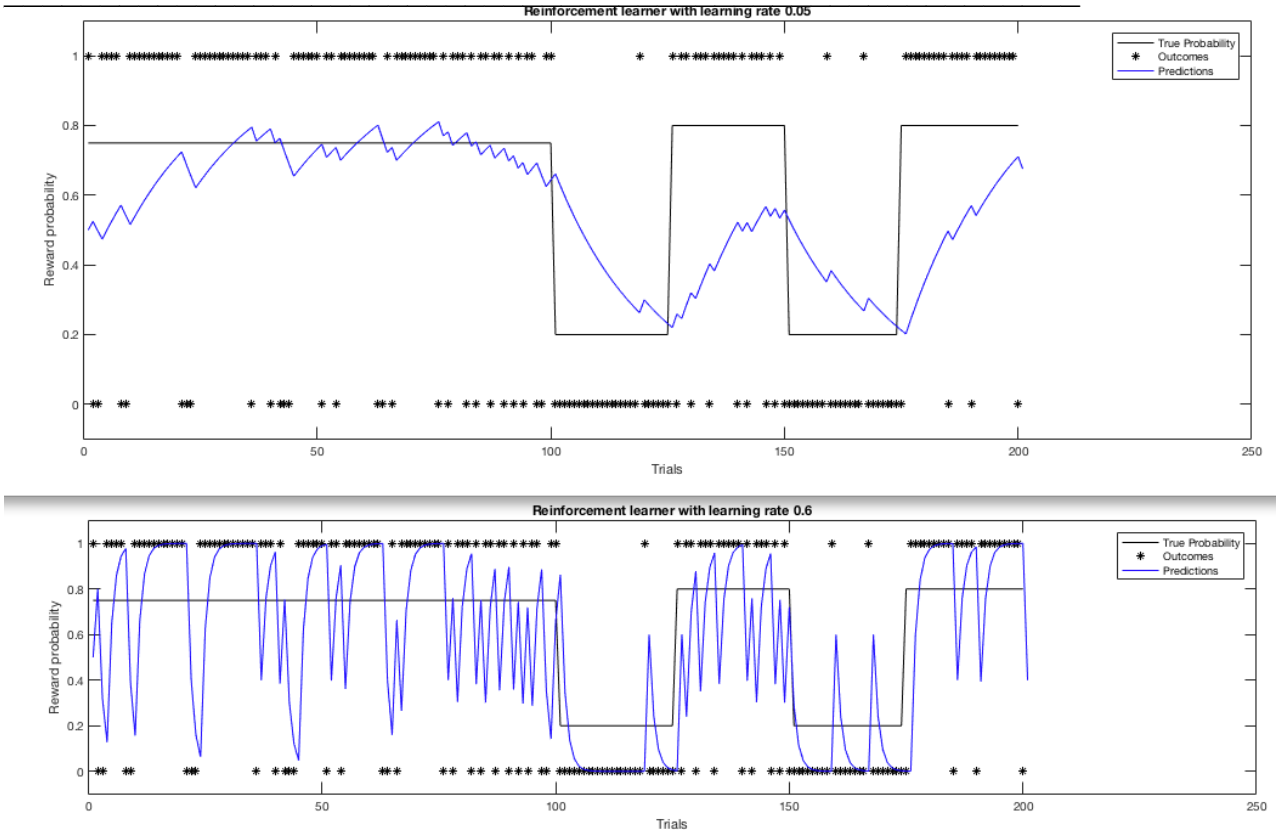


Figure 2. The predictions with a small (top) and large (bottom) learning rate.

Question: After seeing how learning changes with different learning rates, can you find two learning rates, one, which would be roughly best for the stable period and one that would optimize learning for the volatile one?

_____

_____

How can a model make decisions?

The computational learning model described above is applied to every trial of the experiment. What we now have based on equations 1-2 is a model that makes predictions about how likely a reward is to happen if the model chooses one outcome or the other. However, that alone is not enough to solve the task and we want our model to be able to describe behaviour in a way that participants might make decisions.

The next two equations describe how we can make the model actually chooses one of the two options on each trial. First, the model needs to combine the predictions of how likely an outcome is with the size of the potential outcomes. For example, even if you are almost 100% sure that you will receive a reward if you choose an option, if the reward is very small (say 2 out of 100 possible points), you might not want to pick the option. To formalize the fact that the model and our participants need to take into account both magnitude and probability, we could use different methods. While there is experimental evidence for several different ways to combine probabilities with magnitudes, for now we stick with the normative or optimal way to do so:

Utility (or value) is a single number that summarizes how good or desirable an option is, taking into account all relevant factors.Here:
Reward probability → how likely you are to get a reward if you choose an option Reward magnitude → how large that reward is

*[3] Utility = Reward probability prediction * Reward magnitude*

As you can see above we simply multiply the probability with magnitude to generate the overall utility or value of an option. An alternative instead would be to make a weighted sum of the two.

Question: What would you want to look out for when thinking about a method to combine two dimensions? What specifically would you want to consider if you were to add them together?

you must carefully consider scaling and weighting, otherwise the decision could be dominated by whichever dimension has larger numbers.
Human decision-making often uses subjective weights, so sometimes addition makes sense for modeling behavior, but you need to make it comparable.
Scale differences: Probability is usually between 0 and 1 Magnitude could be any number (e.g., 0–100 points)
Adding them directly could bias the utility toward the larger scale (e.g., magnitude dominates probability)
Participants may weight probability and magnitude differently
Multiplication naturally reflects expected value, but a weighted sum allows tuning:

Now the model has given each option a specific utility for every trial. In general the model should prefer the option with the higher utility, but how can we make it select a specific option on each trial?

Intuitively, you might want to say, **always** pick the option with the higher utility even if the difference is small. This would be a so-called 'hard-max' (for hard maximization) rule, and make our agent optimal within the constraints of the chosen learning rate. However, we know that this is not how real participants behave; they sometimes make mistakes and instead pick options with lower value. The might also not precisely know the exact utility of each option and genuinely believe the slightly worse option to be slightly better! We also know that mistakes are more likely to occur the closer the options are to each other in terms of their respective utilities. If you take all this together, i.e. that choices aren't deterministically, but only **stochastically** related to the utility of both options and that the probability of choosing an option should be proportional to how close the options are to each other, we can use the following formula (softmax).

[4] $p(choice_{OptionA}) = \frac{e^{\beta * UtilityOptionA}}{e^{\beta * UtilityOptionA} + e^{\beta * UtilityOptionB}}$ SOFTMAX

In words: the probability of choosing option A is given by a **softmax** function with the free parameter β, also called the 'inverse temperature'. The higher the inverse temperature, the 'steeper' the softmax is (see figure below). The reason it is called 'inverse temperature' is an allusion to thermodynamics: you can think of it as 'the higher the temperature (i.e. the lower the inverse temperature), the faster particles move around', i.e. the more noise there is or the more stochastic behavior becomes. Below and in the matlab script here are plots related to this.

β = inverse temperature → controls how deterministic the choice is

High β
β → almost always pick the highest utility

Low β
β → more random exploration

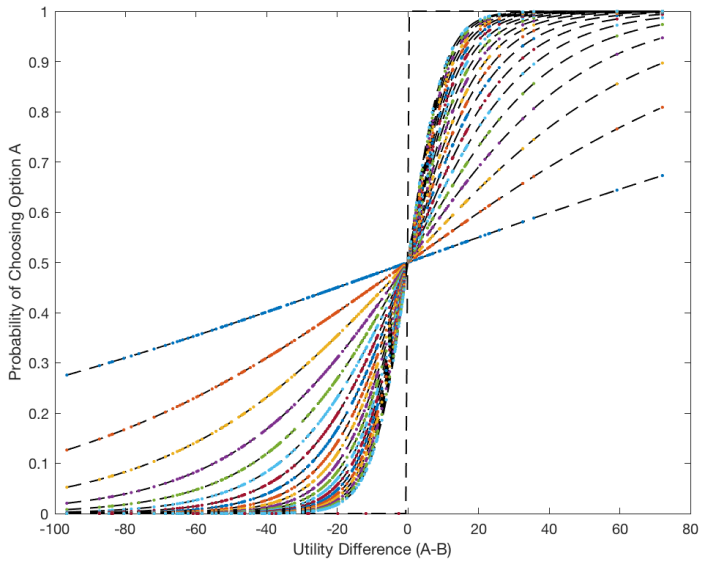


Figure: plot of choice probabilities as a function of utility differences between the two options A and B. The different coloured lines represent different inverse temperatures (sometimes referred to as betas). The lower the inverse T, the steeper the function.

Question: from the plot above it looks like the choice probability is only determined by the difference in utility between option A and B. Is that true for how we wrote the equation in our matlab script? Are there any reasons you might doubt people are only sensitive to the difference in utility between the options?

What matters is the relative difference between the two utilities.
is why your softmax plot is S-shaped along the x-axis of SoftmaxLineX (the decision variable), which represents the difference between options. So, Yes, in the way you wrote the equation in your MATLAB script, that is exactly correct.

Creating simulations from the model

The reason we write down code for computational models is to analyse data from actual participants with these models. We will look at how to do this exactly (i.e. what to code in Matlab) later. However, importantly, whenever you use a computational model to analyse your data, you should first use the very same computational model to simulate data. This involves the following steps:

1) give your computational model the same sequence of trials that you want to give to your real participants (in our case, the outcomes (reward yes or no) and the reward magnitudes)
2) pick a range of different values for your free parameters
3) look at what the model "does", i.e. what internal variables the model has; here for example the reward predictions and the utilities on each trial and the choices between options 1 and 2

Why is it so important for you run simulations rather than just fitting your model to the data?
For one you can use simulations to understand how real participants might behave in your task, whether you picked a sensible schedule and whether you completely defined the decision problem without collecting real participants first.
Secondly, you can use the simulations to validate your experiment and your analysis method. For this, you have to generate at least two models (most likely more), one in which the hypothesis you want to test is true (here: simulate a lower learning rate in the stable block and a higher learning rate in the volatile block) and one in which the hypothesis you want to test is false (here: simulate the same learning rate in the stable and the volatile block). Then, to validate your analysis method, analyse this simulated data with the same models that you used to generate your data. If your analyses work well, you should find that only in the case where you have simulated data to contain your effect of interest, your analysis will show the effect. And in the case where you made the simulation without the effect of interest, your analysis also shows that there is no effect. To learn more about this read: Palminteri, Wyart and Koechlin (2017,TiCS 'The importance of falsification in computational cognitive modeling').

Here: before testing your model on real data, you first simulate artificial behavior under different plausible parameter values to:
check whether your model behaves sensibly, and
validate that your analysis pipeline can recover known effects.

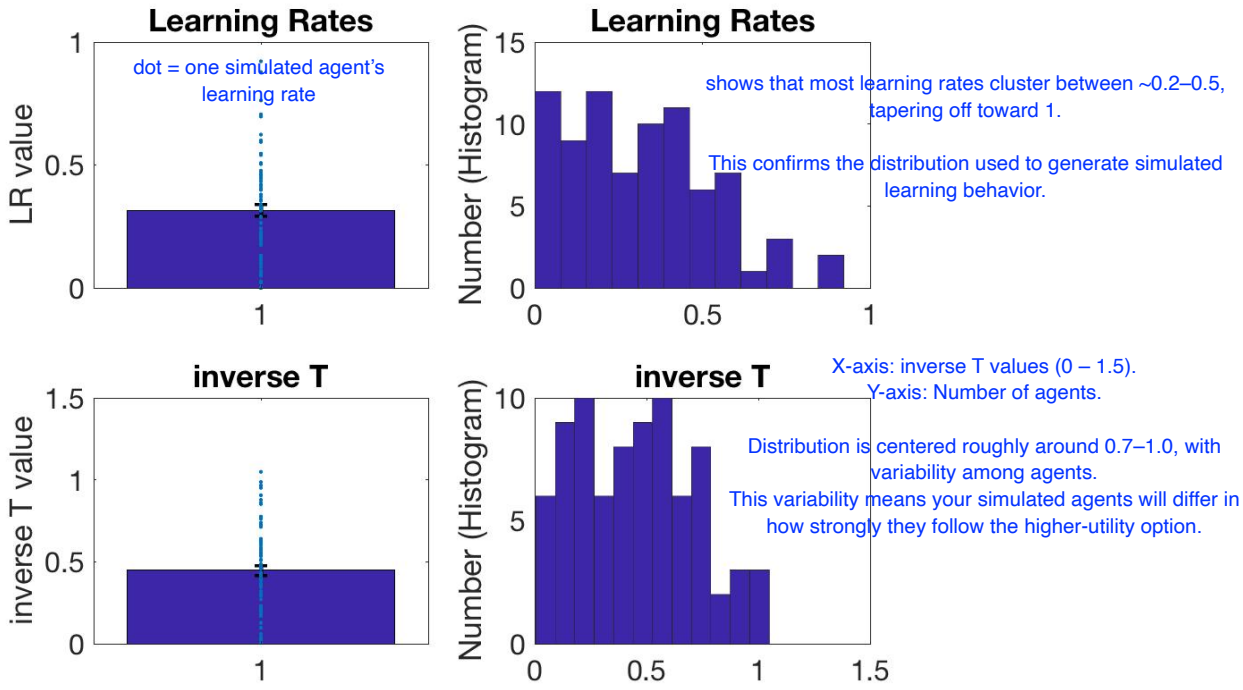


Figure: The distribution of Learning rates and inverse Temperature to generate the simulated agents. We used a normal distribution with a mean of .3 for the learning rates, bound to be above 0. On the left all samples, mean and standard error and on the right the histogram, as this is a nice way to show a distribution.

The script is running two simulations, one with the whole schedule and one with very few trials and saves this to show the usefulness of simulations later in the model fitting procedure! Later I will also show another way simulations can be used to validate analyses between models with and without your effect of interest (in our study two vs one learning rate). Below is the binned choices of the model plotted against objective i.e. true utility difference. We had to reduce the number of bins for the short simulation, as there was otherwise not enough data in each one.

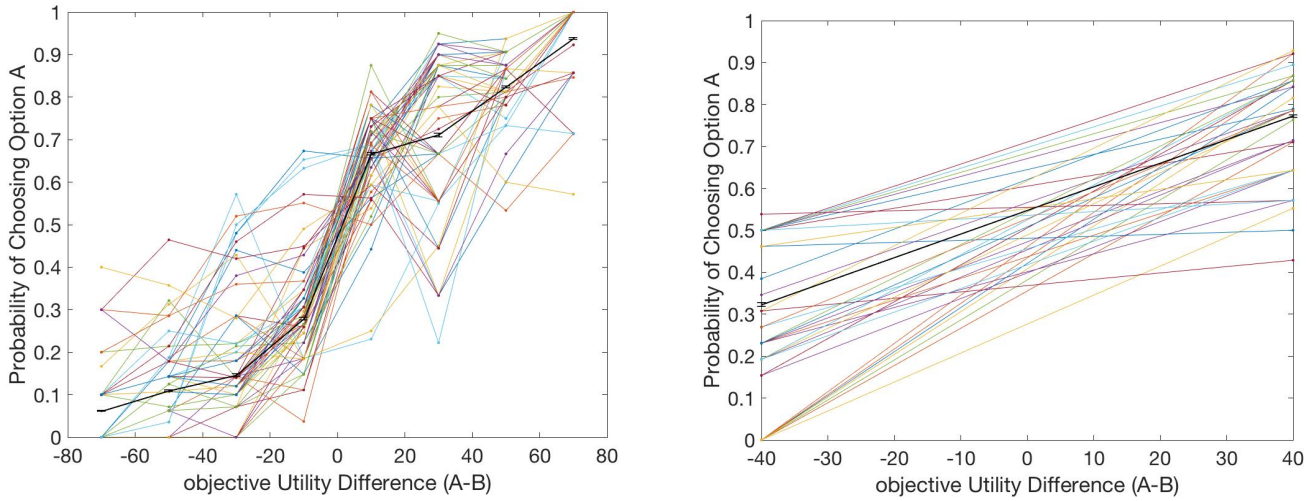


Figure: proportion of A choices in equally spaced bins of objective utility differences. The black line is the overall average across agents and the coloured lines represent individual agents. On the left the full simulation and the right only 40 trials. Due to the small number of trials we had to reduce the number of

bins to two.

Question: Why do we need to bin multiple trials for the plot of the actual choices, while the softmax earlier was individual trials?

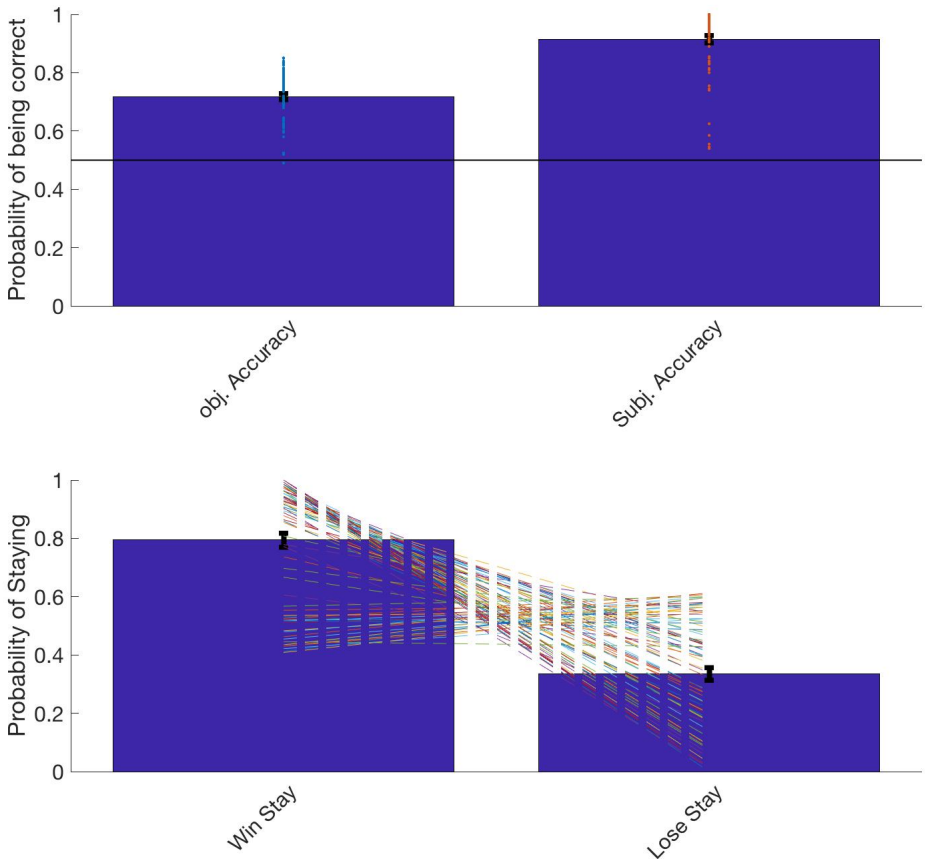
We bin multiple trials for the logistic regression plots to smooth out trial-by-trial noise and reveal overall behavioral trends, like "Win-Stay" or "Lose-Stay" tendencies. In contrast, the softmax earlier produces single-trial probabilities, so individual trial predictions can be plotted directly. Binning makes the aggregate effects of past outcomes more interpretable.

Question: Why do you think the plot above with the actual behavior of the simulation looks different than the previously shown choice probability softmax function even when binned?

The simulated behavior plot can look different from the softmax choice probabilities because the actual choices include stochastic variability—the agent sometimes chooses against the softmax probability, especially if the temperature parameter allows exploration. Even when binned, these random deviations accumulate, so the observed "Win-Stay" or "Lose-Stay" probabilities don't exactly match the smooth, idealized softmax curve.

Model "behaviour"

Looking at some "behavioural measures" of the simulation you can see the subjective [using the learners estimate of probability] and objective [using the true probability used for generating the outcomes] accuracy as well as the odds of staying with an option following reward and no reward. Particularly the reward/no-reward difference in behaviour is interesting because it is a simple model-free way to look at outcome sensitivity of behaviour. The coloured dotted lines are individual simulated participants.



Question: Why is the subjective accuracy always higher than the objective one?

Subjective accuracy is higher than objective accuracy because it is conditional on the agent's own choices—it measures how often the agent's choice was correct given what it chose, ignoring trials it didn't select. Objective accuracy, in contrast, considers all trials regardless of choice, so mistakes on less likely or exploratory choices lower the overall accuracy. Essentially, subjective accuracy reflects a "self-centered" view of performance.

Question: Are these aggregate measures sufficient for looking at our experiments? If not, why not?

No, aggregate measures are often not sufficient because they collapse over trials and ignore trial-by-trial dynamics, such as how past outcomes influence current choices or how learning evolves over time. They can mask important effects like Win-Stay/Lose-Shift biases, learning rates, or adaptation to changing reward structures, which are crucial for understanding the cognitive mechanisms in your experiments.

Question: Are you surprised that for some participants staying with an option is more likely after a lose than a win? Why? What would you conclude from the plot about the those simulated agents internal model?

Not really—if some participants are more likely to stay after a loss, it suggests they may be using a different strategy or internal model than simple Win-Stay/Lose-Shift. For those simulated agents, the plot would indicate that their internal model might expect reversals or changing reward contingencies, so they "stick" after a loss in anticipation that the option could still be rewarding next time. Essentially, it reflects a non-standard learning or exploration strategy rather than purely following immediate outcomes.

Question: Does looking at the analyses above of simulated behaviour inform or influence your confidence in analyzing the data in that way, particularly considering how we generated to simulations in the first place?

Yes—it helps build confidence because the analyses recover patterns that match the internal rules of the simulated agents, like Win-Stay/Lose-Shift tendencies or longer trial-history effects captured by the logistic regression. Seeing that the methods pick up known, controlled behavior from the simulations suggests they are sensitive and appropriate for analyzing real experimental data. At the same time, it highlights that interpretations depend on assumptions about the agents' learning strategies, so one should be cautious about over-interpreting correlations as causal mechanisms.

The Importance of playing around with a model:

Generally, I encourage you to try tinkering with matlab code and the model. It's a great way to understand your model, behavior and lots of other things. It allows you to play around with crazy ideas (e.g. what happens if the learning rate was to shrink throughout the experiment? Or how does simulated behavior look like if accumulated unsigned prediction errors increase the learning rate?). Also, importantly, the more complex your model is, the less you will be able to predict how the model behaves, either when changing parameter values or for any given set of trials, before explicitly simulating it as you don't necessarily know all complex interactions of different aspects of your model.

What about our experimental manipulation?

You might have noticed that what we have covered so far in the script and handout does not directly address our manipulation yet. This is because, before we can look at changes in learning rate, we first need to make a model that can solve the task in general. Now that we have figured out how people might learn, integrate magnitudes with probabilities and generate choices, we can test our predictions about how the learning process might be modulated (i.e. changed) by context such as volatility or stress.

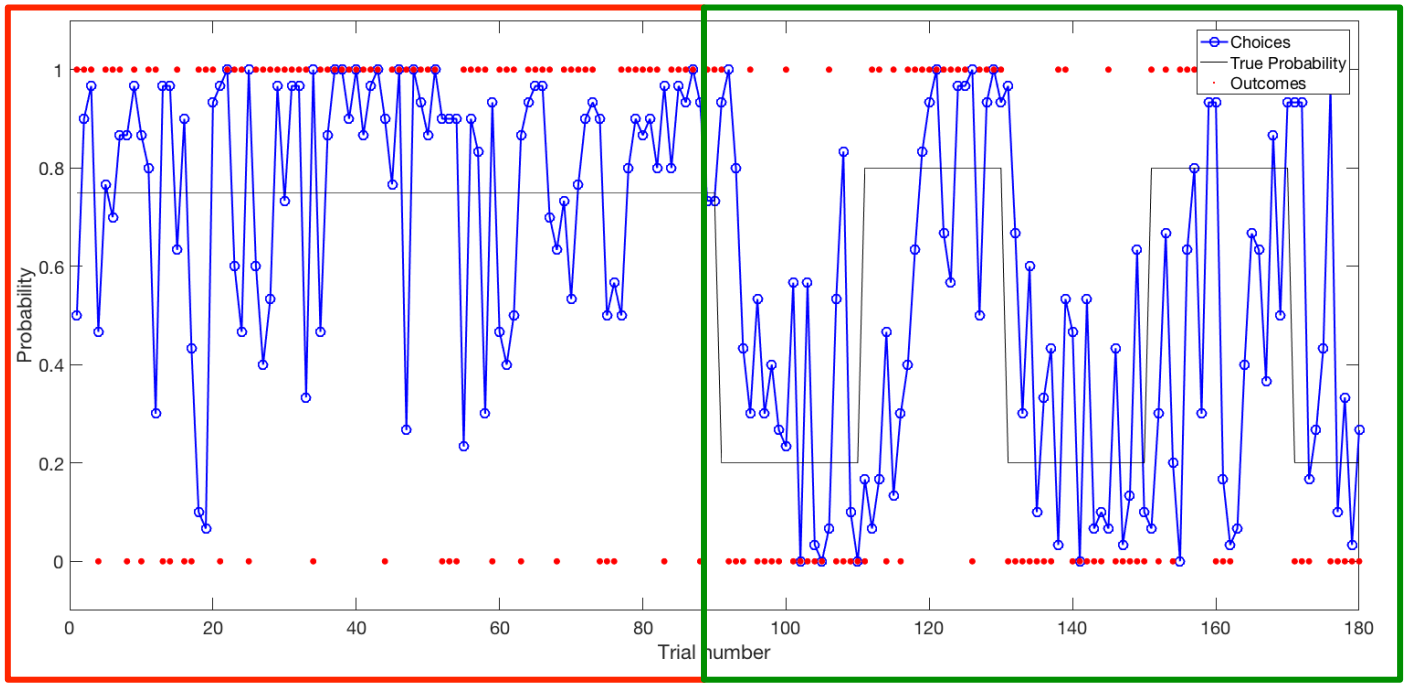


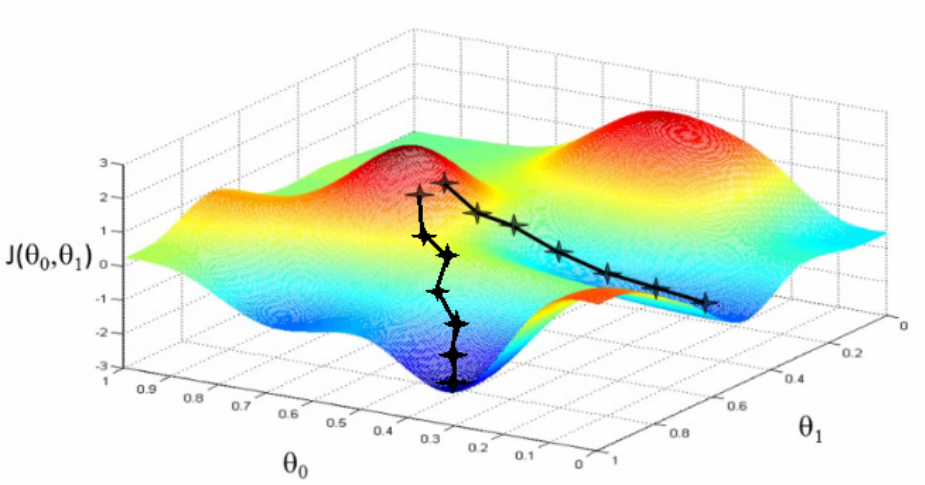
figure: Example schedule highlighting the manipulation of volatility half-way through (red stable, green volatile)

As you hopefully remember, the schedule is split into two phases, with order counterbalanced between participants. We predict that the learning rate is higher in more quickly changing environments and lower in more stable ones.
One way to test this is by fitting a model with free parameters to the data, which has separate learning rates for each phase. Then we can test whether there is a significant difference between the fitted learning rate. This is one very elegant way to test our key hypothesis!

How do you fit free parameters to data?
There are many ways to fit data and we will discuss the procedure more in the next practical. For now it is enough to know that the process is about finding the best fitting model (i.e. the model with the least fitting error or highest likelihood of the data given the model [P(D|Model)]). Sometimes it is easiest to imagine the Model parameter space as a landscape (easiest for two parameters like one learning rate alpha and one inverse temperature beta; so each parameter is an axes of the parameter space). Below you can see an example. All the fitting method tries to do is find the best model configuration, without knowing the whole landscape. In fact, the easiest procedure is computing the whole landscape and just picking the best value, but that becomes infeasible very quickly when models become more complex.

Model Fitting Procedure (fminsearch)

Minimizing negative log likelihood, i.e. finding the model with the highest probability given the data



ground truth = known parameters

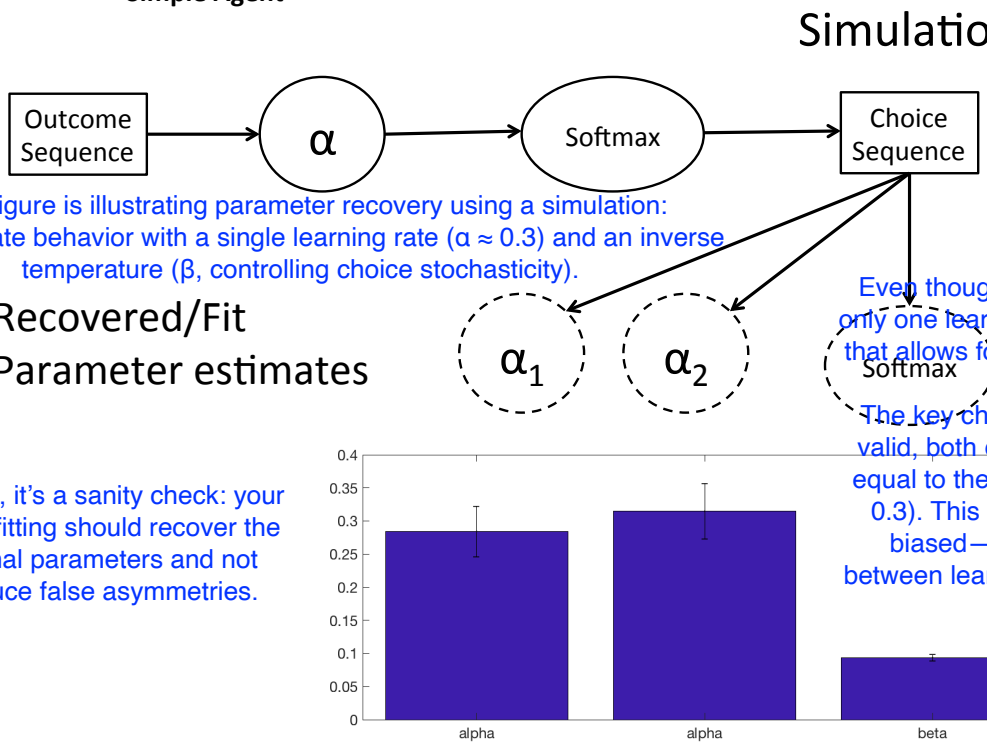
How to know that your parameter comparison is valid?

When it comes to model fitting, the best way to test whether your statistical test is fair/valid is running simulations. This is done by running a simulation with a known ground truth (fancy way of saying you know what model or parameters and parameter values generated the data). In one version there is only one learning rate and in another there are two. Then you can simply analyze the simulated behavior and see whether the fitting procedure A) recovers the parameter estimates you used for data in the first place and B) that a model with only one learning rate when fit with two possible rates does not fit two different values, while a simulation with two does. This is important because it shows that your schedule is not biased in some way to "mimic" to learning rates, when only one exists and that you are sensitive enough with sampling to find two learning rates when two exist. In other words it can show sensitivity and selectivity. See below for illustration of the procedure.

Using two learning rates means that the simulated agent learns differently from positive outcomes (rewards) versus negative outcomes (punishments or no rewards):

Parameter Recovery using simulations

Simple Agent



This figure is illustrating parameter recovery using a simulation: You simulate behavior with a single learning rate (α ≈ 0.3) and an inverse temperature (β, controlling choice stochasticity).

Even though the data were generated with only one learning rate, you can still fit a model that allows for two learning rates (α^+ for wins, α^- for losses).

The key check is: if the fitting procedure is valid, both α^+ and α^- should end up roughly equal to the true α used in the simulation (≈ 0.3). This shows your fitting method isn't biased—it doesn't invent a difference between learning from wins and losses when there isn't one.

In short, it's a sanity check: your model fitting should recover the original parameters and not produce false asymmetries.

Figure: An illustration of a simulation using one learning rate and inverse temperature (here called beta). When simulating with one alpha, it is still possible to fit two. However, both alpha's should hopefully give you the same answer, ideally the number that you used to generate the data in the first place. In this example it seems to work well, as the real alpha was roughly .3.

Parameter Recovery using simulations

Complex Agent

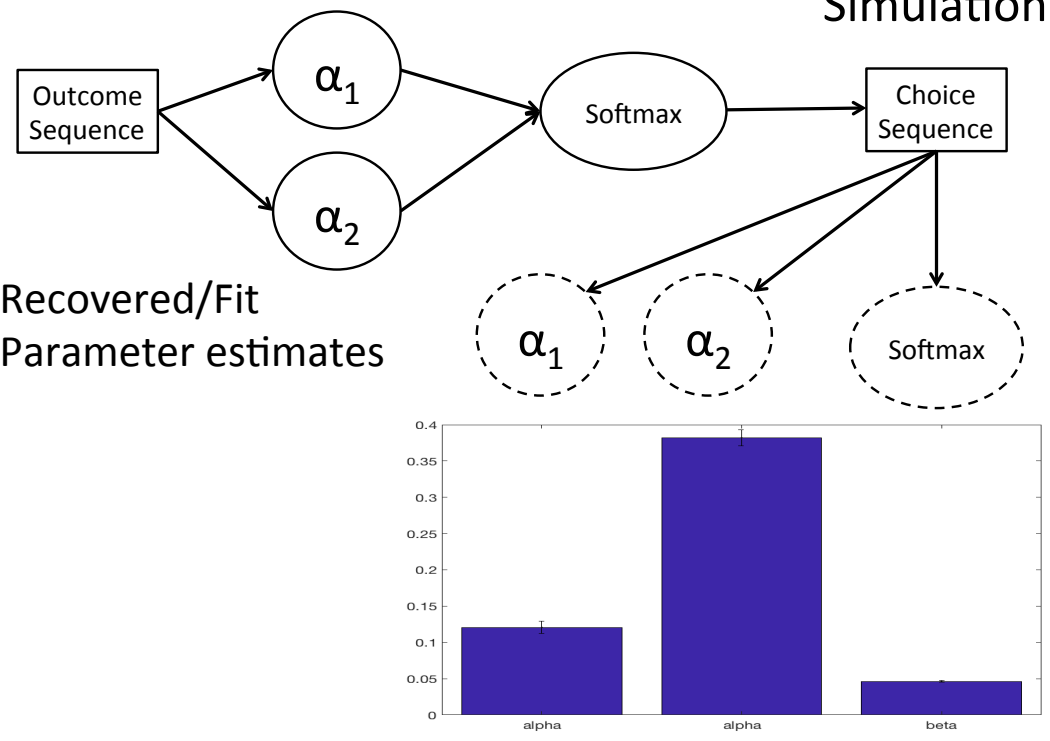


figure: An illustration of a simulation using two learning rates and one inverse temperature (here called beta). When simulating with two alphas, you should hopefully recover two distinct alphas, ideally the numbers that you used to generate the data in the first place. In this example it seems to work well, as the real alphas were roughly .1 and .4.

Summary

Of course, there are many other ways to tackle the question of whether participants learn differently. We could have tried simpler binning analyses (like in the aggregate measurements plot), but those aren't always very sensitive or specific. Alternatively, we could have build a regression model but that can sometimes be more complicated than simply fitting a model of your learning process explicitly [See optional section below to learn how a regression works]. The point of model fitting isn't that it is the only way to look at your data, but that it is a very principled way to deal with complex data, allowing the inference of the underlying mechanism behind such behaviours. It often also allows for very simple comparisons of parameter estimates. Both the potential for more ecological/real-lifeness of the paradigm and simplicity of comparison is why this approach is becoming increasingly popular within psychiatry (See Scholl and Klein-Fluegge 2017 for a review on this. Also generally a great review to explain the use of models in parts of cognitive neuroscience).

Question: When do you think building a model of the task is appropriate? What do you need to know before building a cognitive model? When could a complex model be inappropriate in a task?

when you want to formalize hypotheses about how participants make decisions, identify latent processes (like learning rates, exploration, or biases), or predict behavior under different conditions. Before building a model, you need to know the structure of the task, what variables can influence decisions, and what behavioral data you can collect. A complex model could be inappropriate if the task is too simple, has limited data, or if the model's assumptions are not justified by the task design, because it could lead to overfitting or unreliable parameter estimates.

Question: Do you think there is any value in sitting down and writing a cognitive model as well as simulating behaviours prior to collecting behavioural data? Name at least two reasons why it is useful.

1. allows to test whether your experimental design can actually distinguish between different theoretical hypotheses or parameter values.

2. It helps you check for potential biases or limitations in your task, ensuring that the data you collect will be informative for the questions you want to answer.

3. simulating behavior helps you practice the analysis pipeline before real data collection.

By including reward magnitude as a separate regressor, you "control for" its effect, isolating the unique contribution of past reward history on choice.

In other words, controlling for magnitude ensures that your β weights for the primary regressor reflect its effect after removing the influence of reward size, giving a cleaner interpretation of how that factor alone drives behavior.

Optional Section: Logistic Multiple Regressions

Regression Analysis

All linear multiple regressions rely on a variation of this simple formula below:

$$Y \approx \beta_0 + \beta_1 * Regressor_1 + \beta_2 * Regressor_1 + \beta_3 * Regressor_1 + \beta_4 * Regressor_1$$

In other words, regressions try to find a linear decomposition of the data given the regressors, using the β weights. Given that not all data is normally distributed, some regression models take additional link or transfer functions of the Regressors, before predicting the data. Additionally, many regression tools like glmfit in matlab allow the user to specify the distribution of the data. Using both the correct distribution for the data as well as link function corresponding to the distribution and regressor, is essential for appropriate regression results. If in doubt about the distribution of your data, just plot it using a histrogram.

Multiple regressions are one of the most commonly used methods for many experiments, particularly in decision neuroscience. They are used both for neural and behavioural data and are great for taking into account/controlling for multiple not completely independent/uncorrelated factors (see figure below for an example taking account of additional magnitude information or not. You can see that the learning estimates change by controlling for magnitude information). When using Regressions together with model based analyses, additional issues of interpretation can arise which we are going to explore below. One solution of those problems is running simulations. Furthermore, whenever running a regression it is important to look at the correlational plot of your design matrix (designmatrix=all regressors of a multiple regression) to see whether there are any problems of too large co-linearity or even rank-deficiency in your analysis. For this it is important to not only look at averaged correlations but specifically at correlations in every individual participants.

regression using past rewards (t-1…t-10) as regressors. You can see that the effect of the most recent trial (t-1) is largest and negative, indicating a strong influence on current choices. Controlling for magnitude ensures that this effect is truly about previous choice history, not reward size.

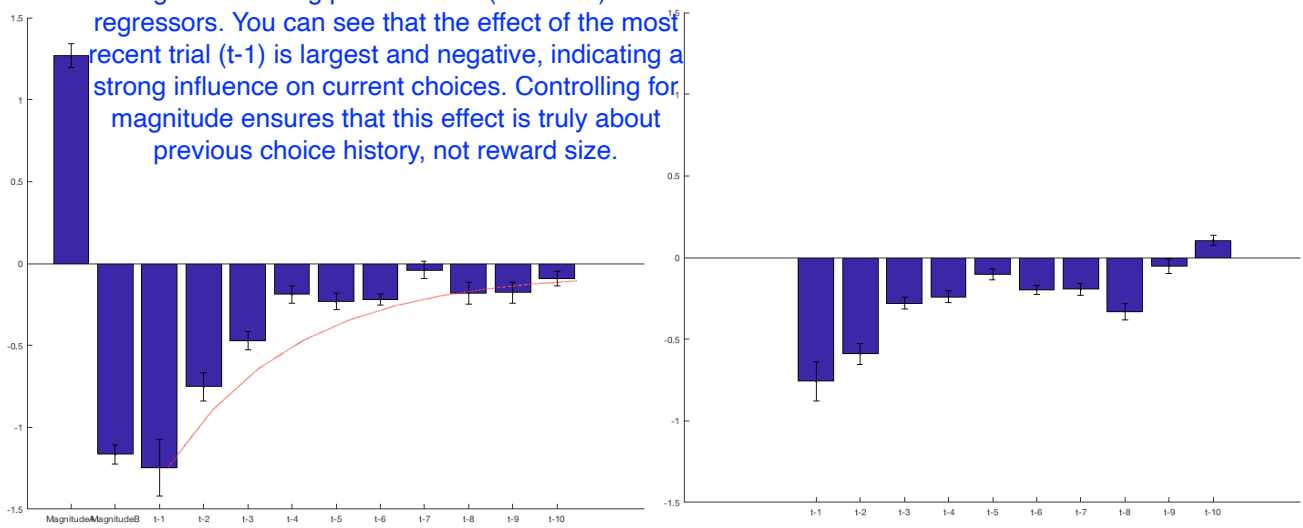


Figure: Including Magnitude information improves fit of the reward outcomes as well. You can also here now see in a completely data driven way that the exponential function of the RL model looks appropriate (see overlain red line)

Even more optional questions:

Q1) We are running a binomial logistic regression on Choice Data. Why are we doing this? What would other link functions do and why are most of those inappropriate for binary choice data. What would change for e.g. scalar rating or reaction time data? How could you make sure you are using the correct link function for your specific data type?

Q2) What can multiple regressions give you that simpler breakdowns of data according to one dimensions don't offer?

Q3) Given the red curve and regression weights plotted above, what Do you think happens for different learning rates?

Q4) There are also multiple ways how to enter factors i.e. parameterize the regression analysis. Why could changing parametrizations be useful? For example, you could enter every previous outcome as a separate regressor or alternatively, make one regressor with the current prediction of a reinforcement learner instead. You can't do both at the same time, as this would be rank-deficient (one would be a linear combination of the other).

Q5) What does the constant ($beta_0$) in a regression mean?

Q6) What happens if you don't normalize, i.e. z-score your regressors?