A Beginner's Beginners' Guide to Modelling Learning and Decision-making Under (A LOT OF) Uncertainty

Eddie Xiao

9th Feb

Table of contents

- **<u>Model simulations: creating virtual participants</u>**
- Regularisation of learning rates under volatility and stochasticity

- **<u>Model fitting: represent, evaluate, optimise</u>**
- Friday night fun with Goris et al. (2020)
- Exploring armed bandits

Problem: n-armed bandit

✅ = win £10
❌ = no win

✅✅✅❌✅✅❌✅✅✅✅✅✅❌✅

Expected value: £8
Probability: 0.8

✅❌✅❌❌✅❌✅❌✅❌✅❌✅✅

Expected value: £5
Probability: 0.5

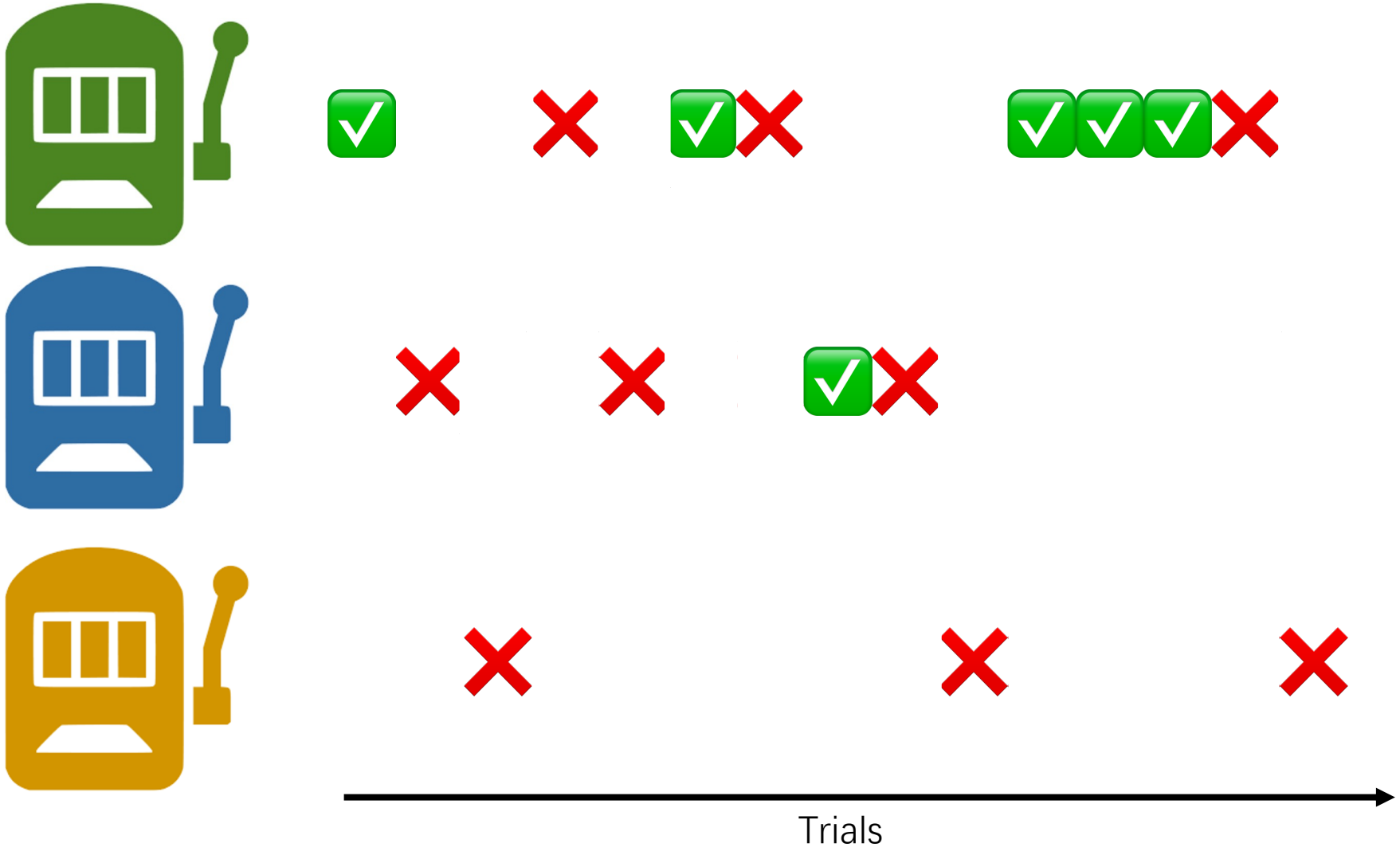
✅❌❌❌✅❌❌❌✅❌❌✅❌❌❌

Expected value: £2
Probability: 0.2

Problem:

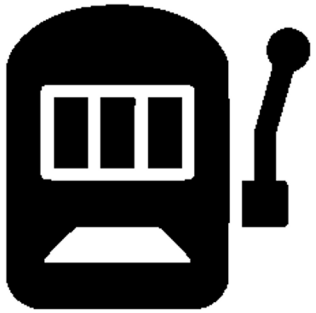
1) Learn about value / probability
2) Explore-exploit trade-off

✅ = win £10
❌ = no win



Simpler case: one-armed bandit

✅ = win £10
❌ = no win



✅✅✅❌✅✅❌✅✅✅✅✅✅❌✅

Expected value: £8
Probability: 0.8

❌❌❌✅❌❌✅❌❌❌❌❌❌✅❌

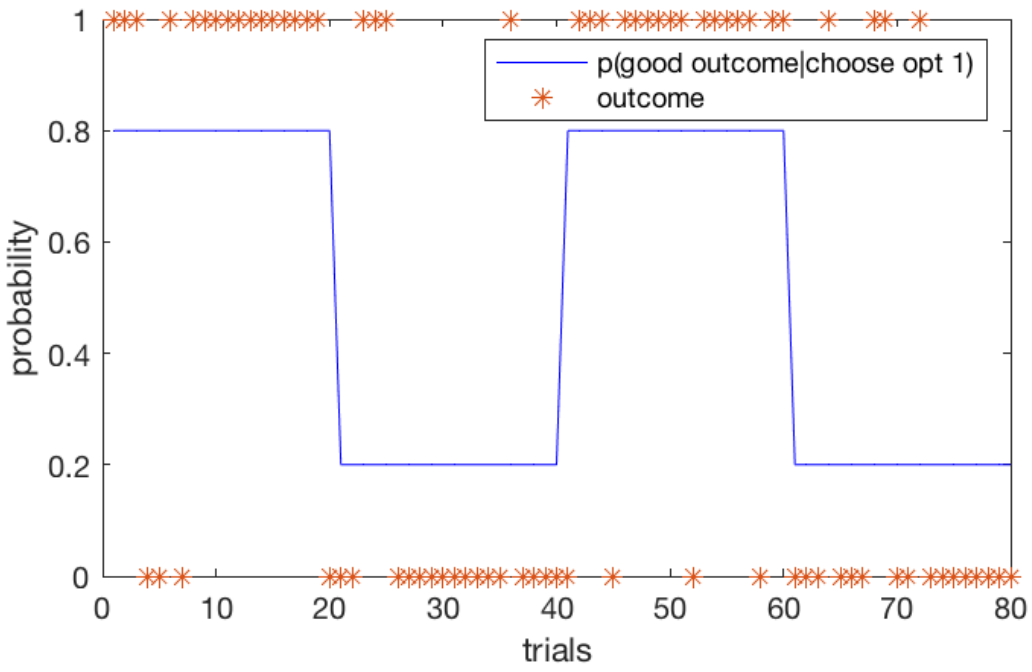
Expected value: £2
Probability: 1-0.8 = 0.2

Trials

MATLAB: creating one-armed bandits

Function: **`gen_misce_task.m`**

Your input: **[20, 0.8; 20, 0.2; 20, 0.8; 20, 0.2]**




```
function task = gen_miniblock(ntrials,contingency)
% produce trial outcomes based on contingency specified
%% outline the probabilistic associations & proportion of successes
p = repmat(contingency,ntrials,1);%probability of outcome
n1s =contingency*ntrials;%number of success available from the option
%% designate whether each trial is a win or loss, for option one
%create a string of binary outcomes with the specified proportion
rawtrials = [zeros(ntrials-n1s,1);ones(n1s,1)];
%permute the sequence of 0 and 1
outcome = rawtrials(randperm(length(rawtrials)));
```

```
%% iterate across miniblocks
for i = 1:size(seq,1)
    m = gen_miniblock(seq(i,1),seq(i,2));%generate a miniblock
    %merge the miniblocks together
    if i == 1
        p(1:trialm(i,1),1) = m.p;%add probability of outcome
        outcome(1:trialm(i,1),:) = m.outcome;%add outcome sequence
    else
        blockini = trialm(i-1,1)+1;%note down starting trial of this miniblock
        p(blockini:trialm(i,1),1) = m.p;%add p
        outcome(blockini:trialm(i,1),:) = m.outcome;%add outcome
    end
end
```

Learning model: Rescorla-Wagner Learning

$$V_{t+1} = V_t + \alpha * \delta_t$$

Update the expected value on each trial
=> for ***each*** CS / ***each*** action / ***each*** bandit

$$\delta_t = o_t - V_t$$

Prediction error as difference in ***value***

$$V_{t+1} = (1\text{-}\ \alpha) * V_t + \alpha * o_t$$

Learning rate ≈ influence of outcome

(More algebra)

$$V_{t+1} = (1-\alpha)^t * V_1 + \sum_{i=1}^{t} \alpha * \boxed{(1-\alpha)^{t-i} * o_i}$$

Response model: Softmax function

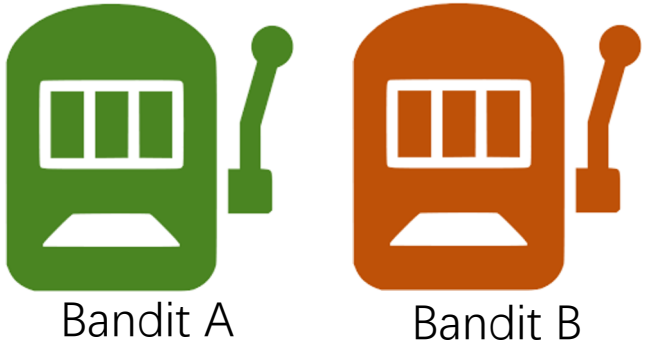
Problem: how do we use the expected values to guide action (choices)?

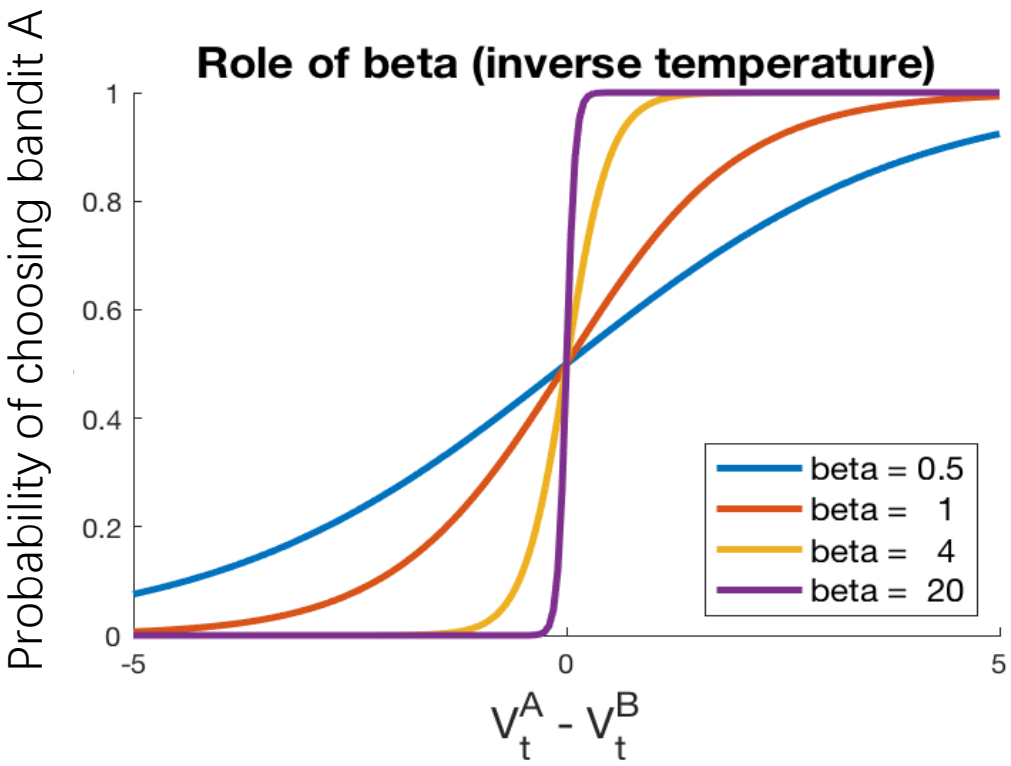
We have n bandits to choose from

$$P(\text{choose bandit k on trial t}) = \frac{e^{\beta v_t^k}}{\sum_{j=1}^{n} e^{\beta v_t^j}}$$

β: inverse temperature

Simple case: Softmax with only two options





$$P(\text{choose bandit A on trial t}) = \frac{e^{\beta v_t^A}}{e^{\beta v_t^A} + e^{\beta v_t^B}}$$

MATLAB: Putting the model together

RW1lr_plsim.m

```
%% choice/response model: decision based on knowledge from previous trials
EV_opt1 = v(t,1);%column 1 is about option 1
EV_opt2 = v(t,2);% value for option 2 across each trial
%compare between two options, and output choice probability
pchoice_opt1 = (1+exp(-beta*(EV_opt1-EV_opt2))).^-1;
pchoice_opt2 = 1-pchoice_opt1;%either choose opt1, or opt2
pchoice(t,:) = [pchoice_opt1,pchoice_opt2];%record the choice probs

%choice behaviour... credits: written by Hanneke den Ouden 2015 <h.denouden@gmail.com>
% Do a weighted coinflip to make a choice: choose stim 1 if random
% number is in the [0 p(1)] interval, and 2 otherwise
if rand(1) < pchoice(t,1) %bigger the pchoice, more likely to choose "1"
    choice(t) = 1;
else
    choice(t) = 2;
end
```

$$P(\text{choose bandit 1 on trial t}) = \frac{1}{1+e^{-\beta(v_t^A - v_t^B)}}$$

choice <= P(choose bandit 1 on trial t)

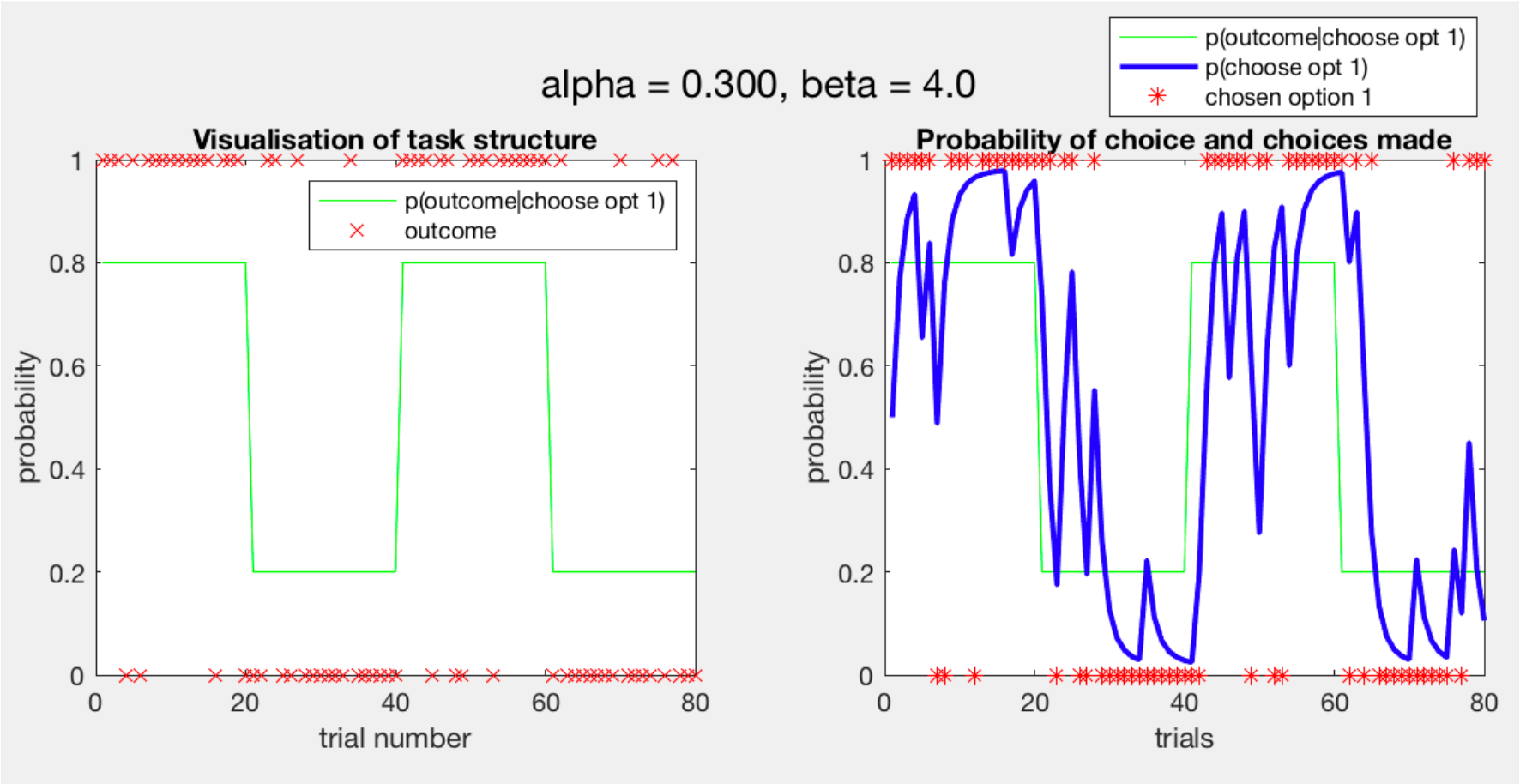
```
%% learning
%PE: observe both options, update both options
PE(t,:) = outcomes(t,:) - v(t,:);%for first trial, note change from baseline

%update expectation on future trial
v(t+1,:) = v(t,:) + alpha*PE(t,:);
```

$$\delta_t = o_t - V_t$$

$$V_{t+1} = V_t + \alpha * \delta_t$$

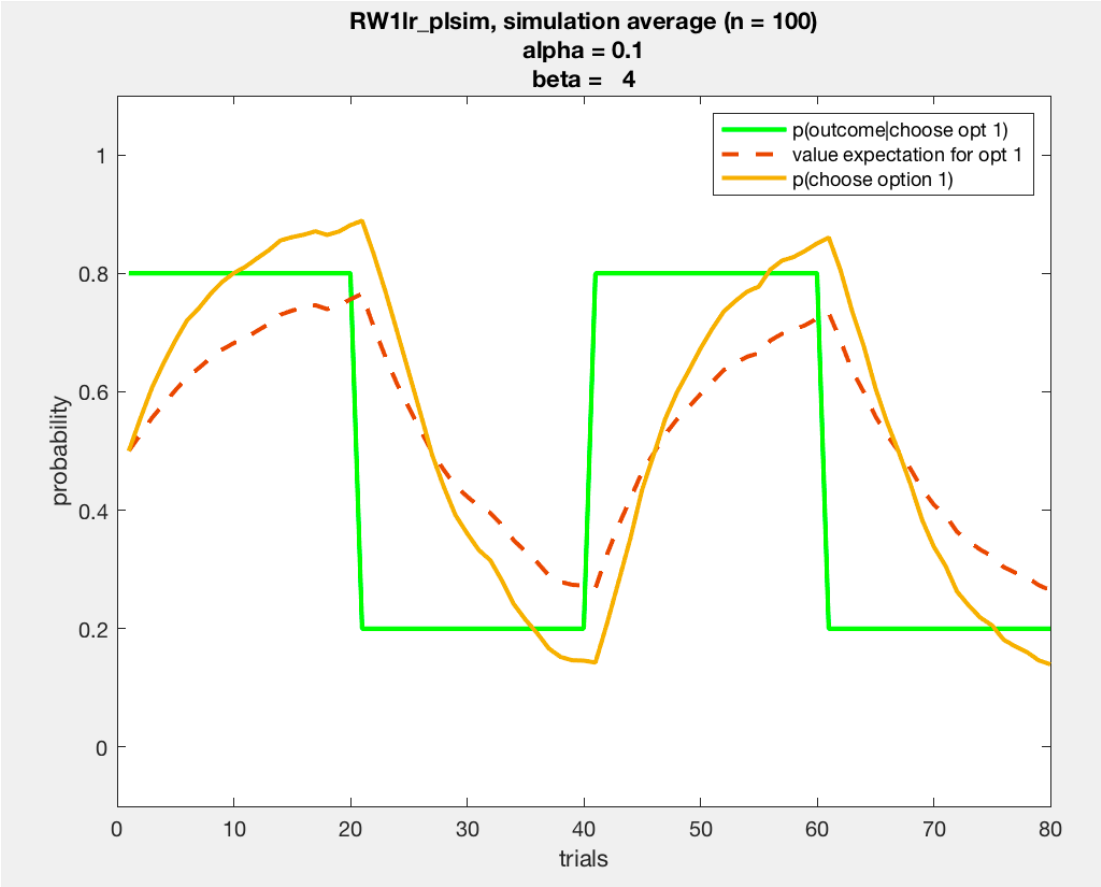
MATLAB: Visualise the virtual participant

Command: `RW1lr_plsim(task1, [0.3, 4], 1)`

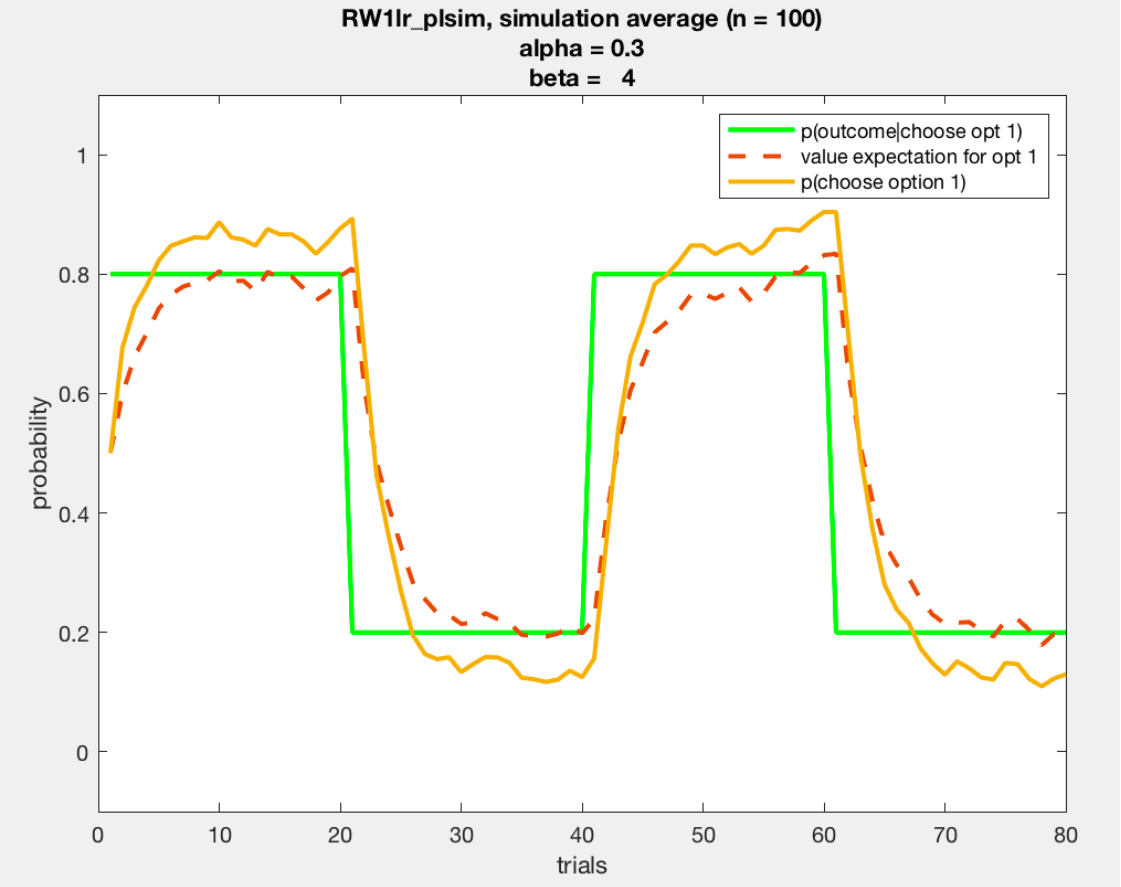


MATLAB: Visualise the virtual participant

```
sim_average_PL(@RW1lr_plsim,[0.1,4],…
[20,0.8;20,0.2;20,0.8;20,0.2],100,1)
```

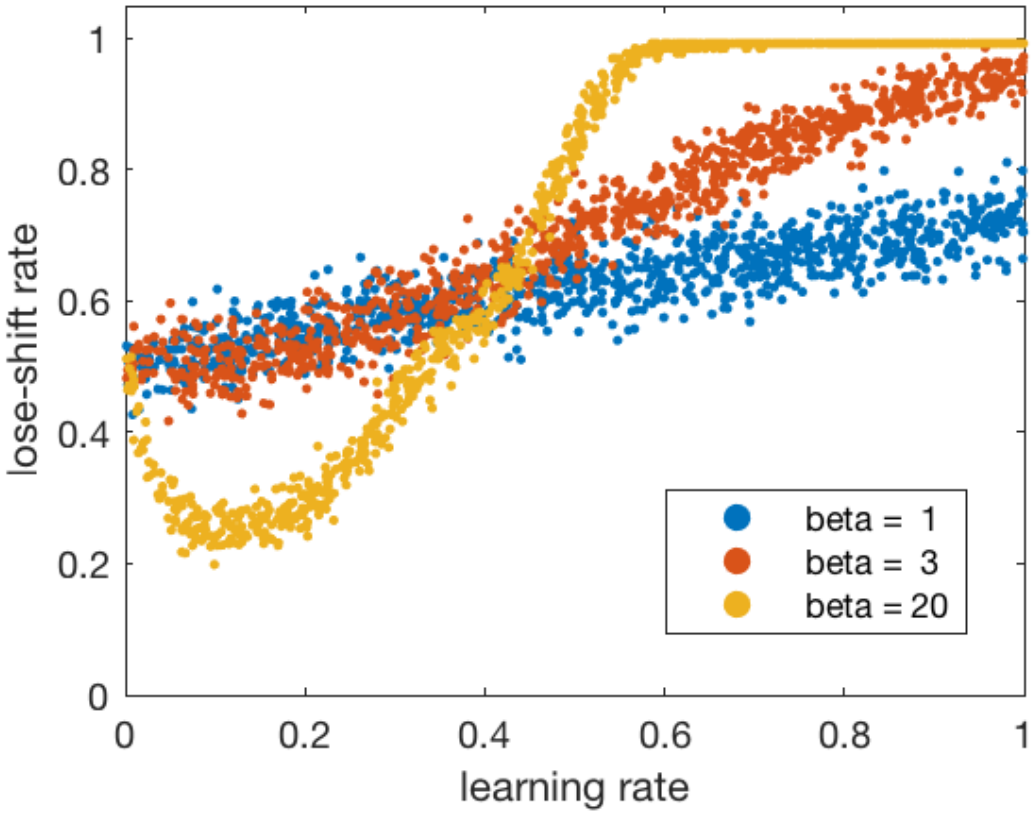


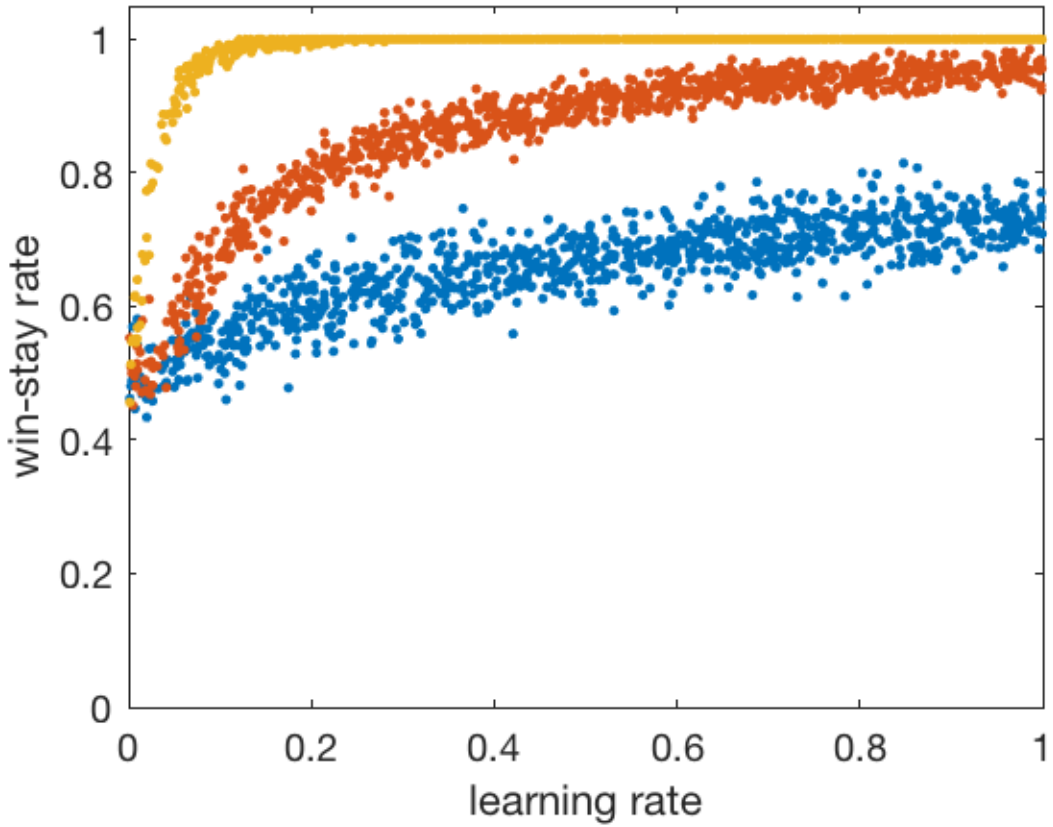

```
sim_average_PL(@RW1lr_plsim,[0.3,4],…
[20,0.8;20,0.2;20,0.8;20,0.2],100,1)
```



MATLAB: win-stay, lose-shift, and earnings

Note: there may be some "reproducibility crisis" for these plots





`PL_metric_corr.m`

`wsls_calc(simulation_output)`

Interim Summary

- Model simulations help understand properties of a model

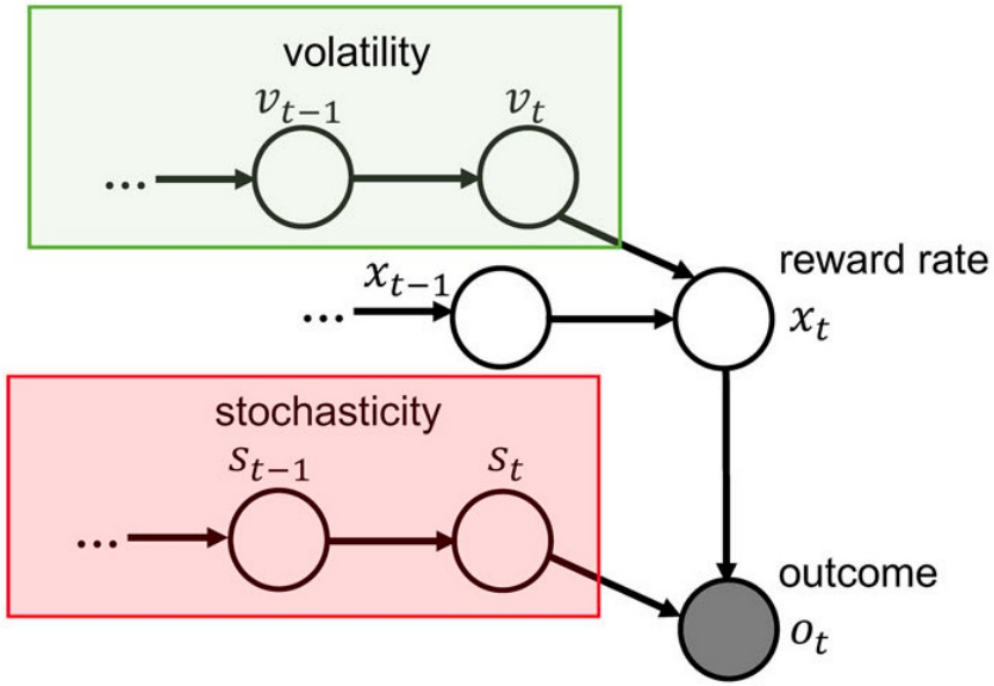
- Components: Learning model & Response model

$$V_{t+1} = (1\text{-}\ \alpha) * V_t + \alpha * o_t \quad \rightarrow \quad \alpha_t = ?$$

$$P(\text{choose bandit k on trial t}) = \frac{e^{\beta v_t^k}}{\sum_{j=1}^{n} e^{\beta v_t^j}}$$

How best to change the learning rate?

- Goal: understand the causes/sources of changes



$$\delta_t = o_t - m_t$$

$$\boxed{\alpha_t = \frac{w_t + v_t}{w_t + v_t + s_t}}$$

$$m_{t+1} = m_t + \alpha_t \delta_t$$

$$w_{t+1} = (1 - \alpha_t)(w_t + v_t)$$

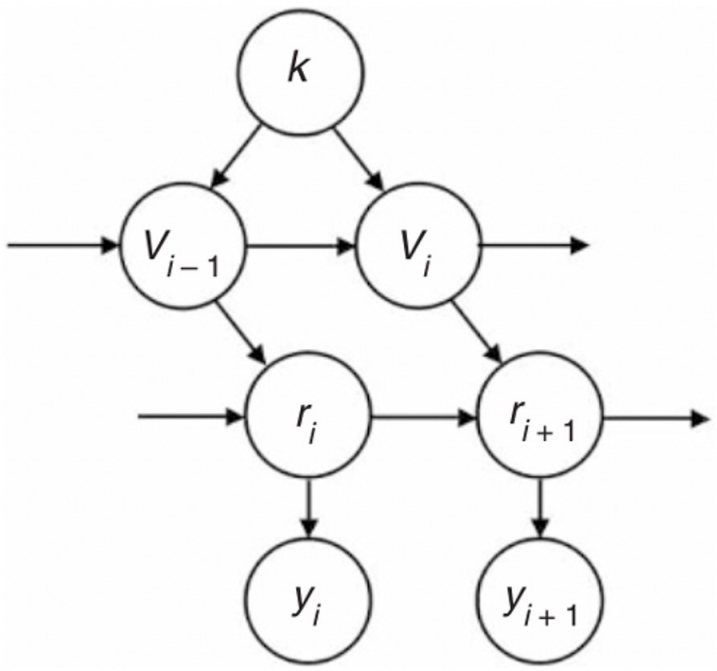
$v_t = ?$

$s_t = ?$

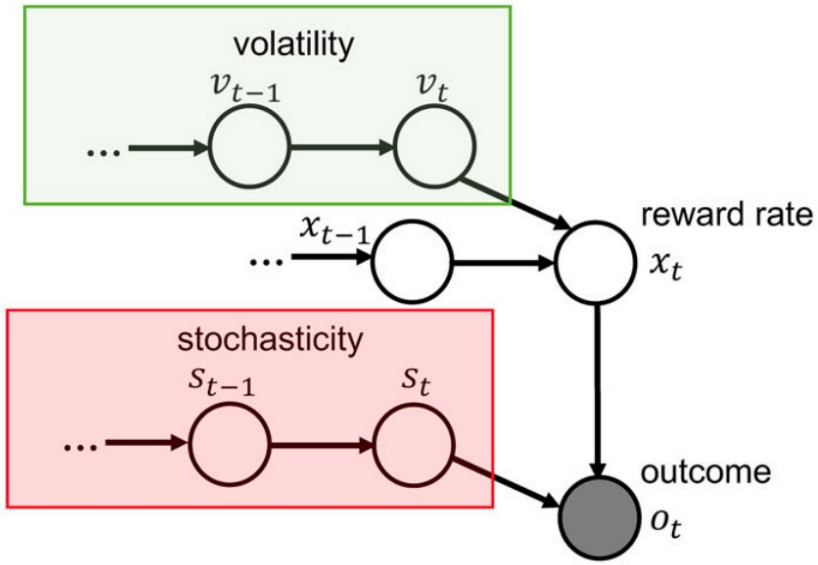
(Piray and Daw, 2021)

The gist for joint estimation of v and s

(Bayesian Observer)



Behrens et al. 2007



Piray and Daw, 2021

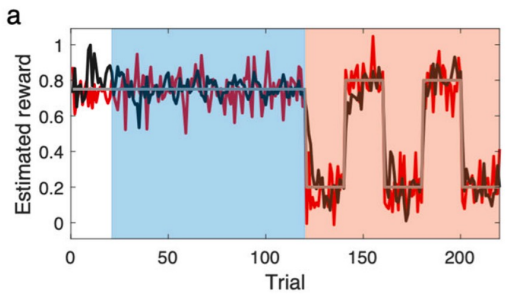
$$\frac{1}{v_t} = \frac{\varepsilon_t v_{t-1}}{1-\lambda_v}$$

$$\frac{1}{s_t} = \frac{\varepsilon_t s_{t-1}}{1-\lambda_s}$$

λ : update rates

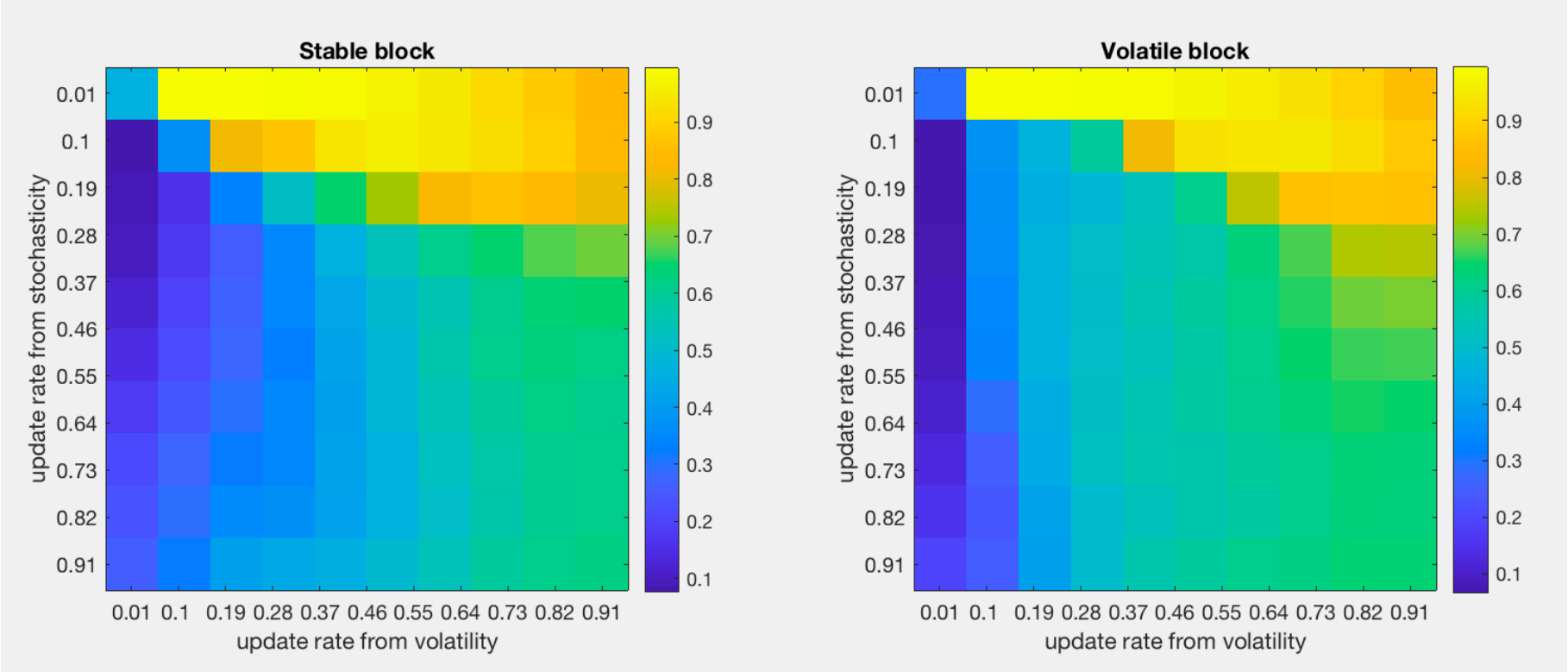
v0 & s0: initial beliefs

Searching across the parameter space…

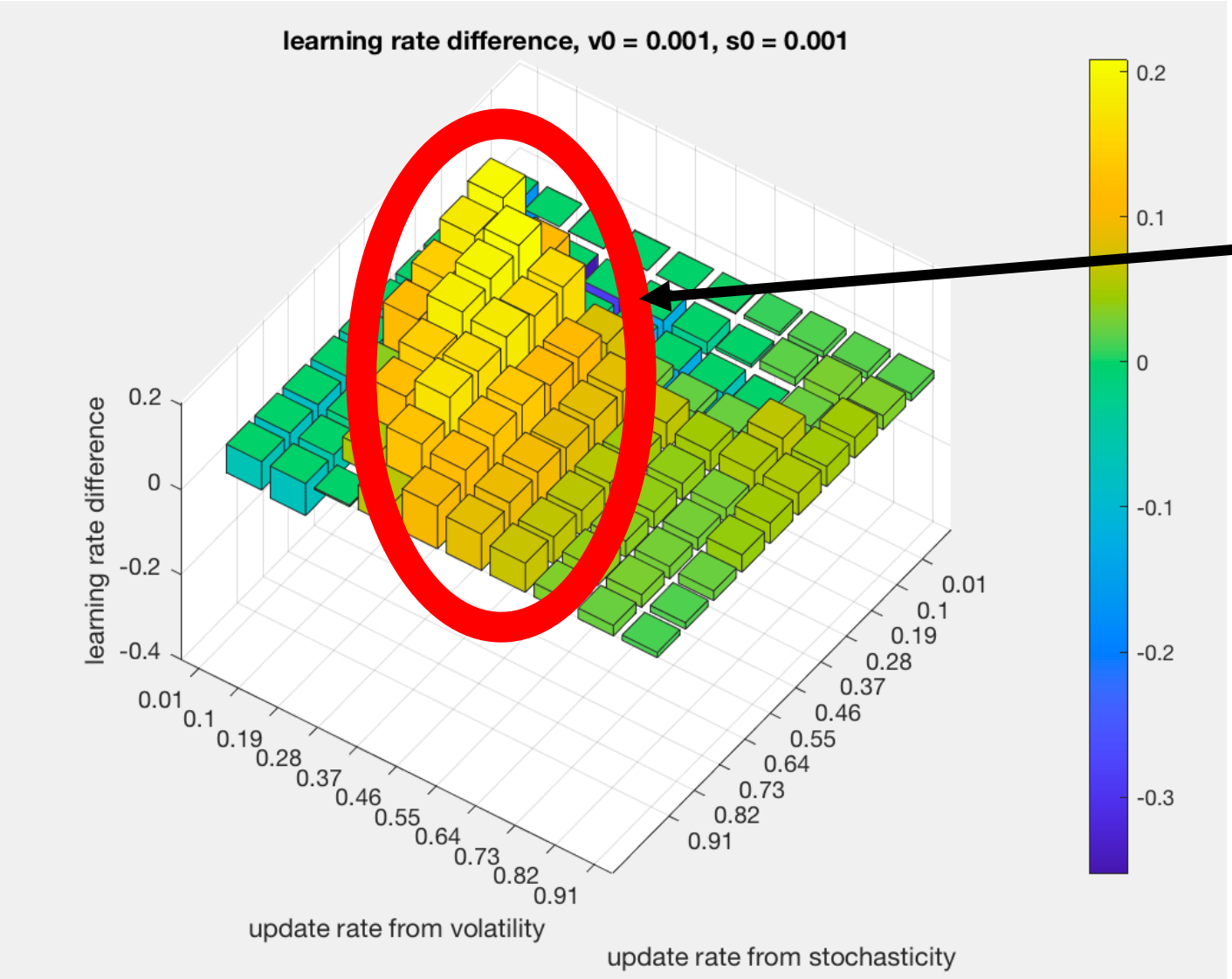


Identical task outcomes, with 100 repeated runs of each parameter combination

Initial beliefs about volatility and stochasticity set to 0.001



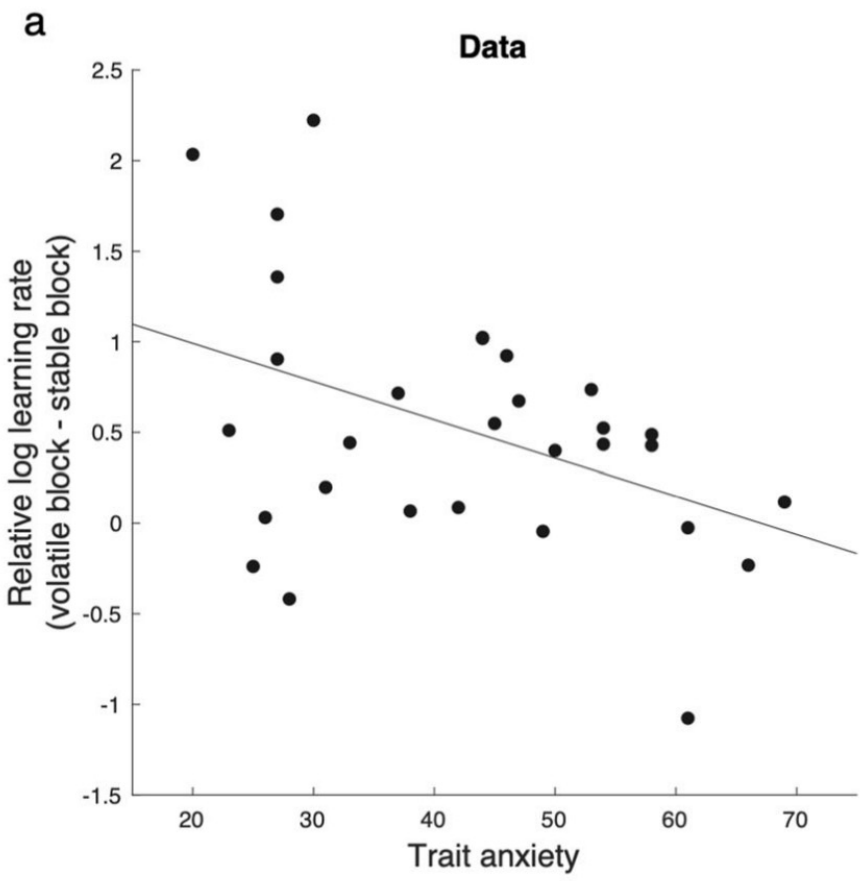
Searching across the parameter space…



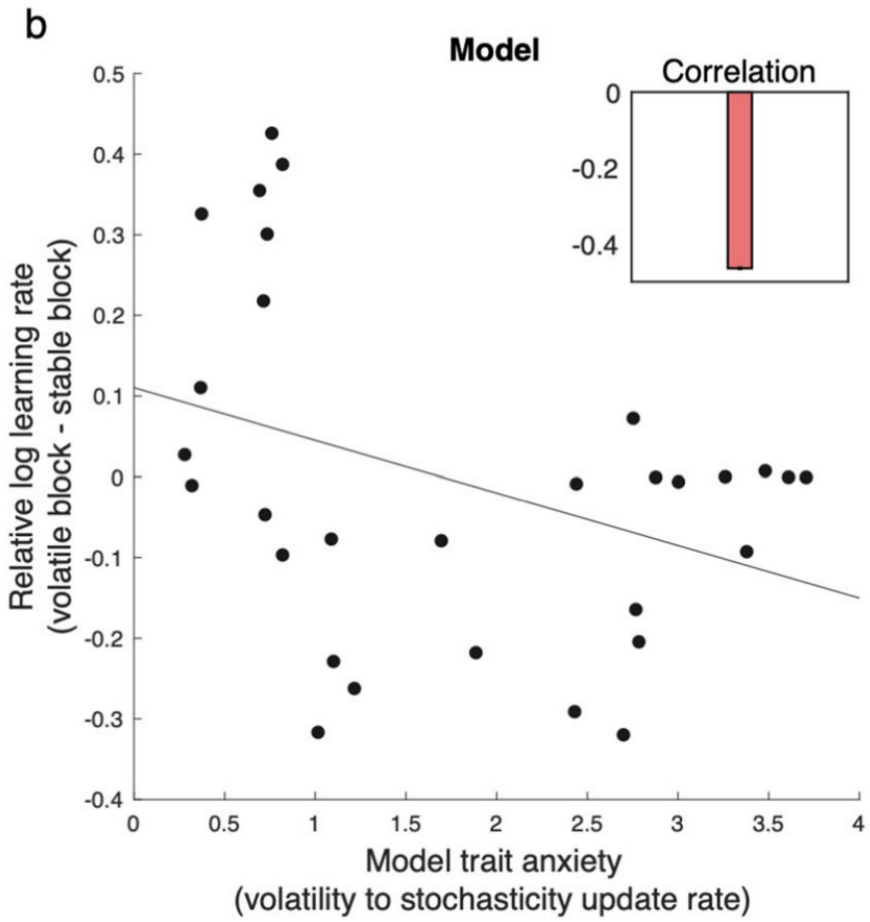
Volatility-to-stochasticity update rate ratio ≈0.5

Good for flexible learning?

(Trait) Anxiety & Uncertainty: Impaired Adaptation to Volatility (IATV)?

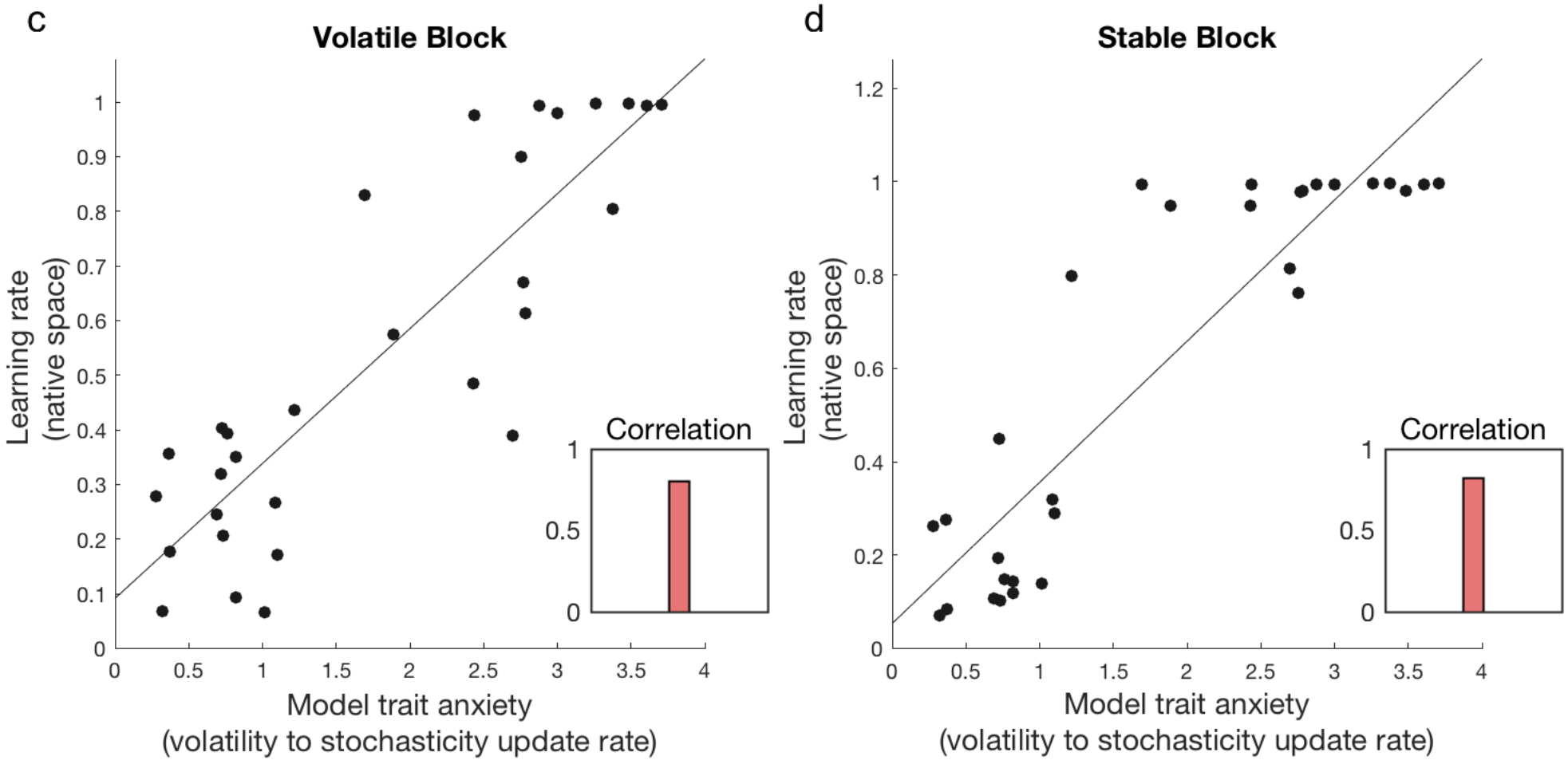


Browning et al. (2015)



Piray & Daw (2021)

(Trait) Anxiety & Uncertainty: Generally elevated learning rate?



Interim Summary

- Tenet: learn more from genuine changes in the mean, learn less when deviations were due to noise

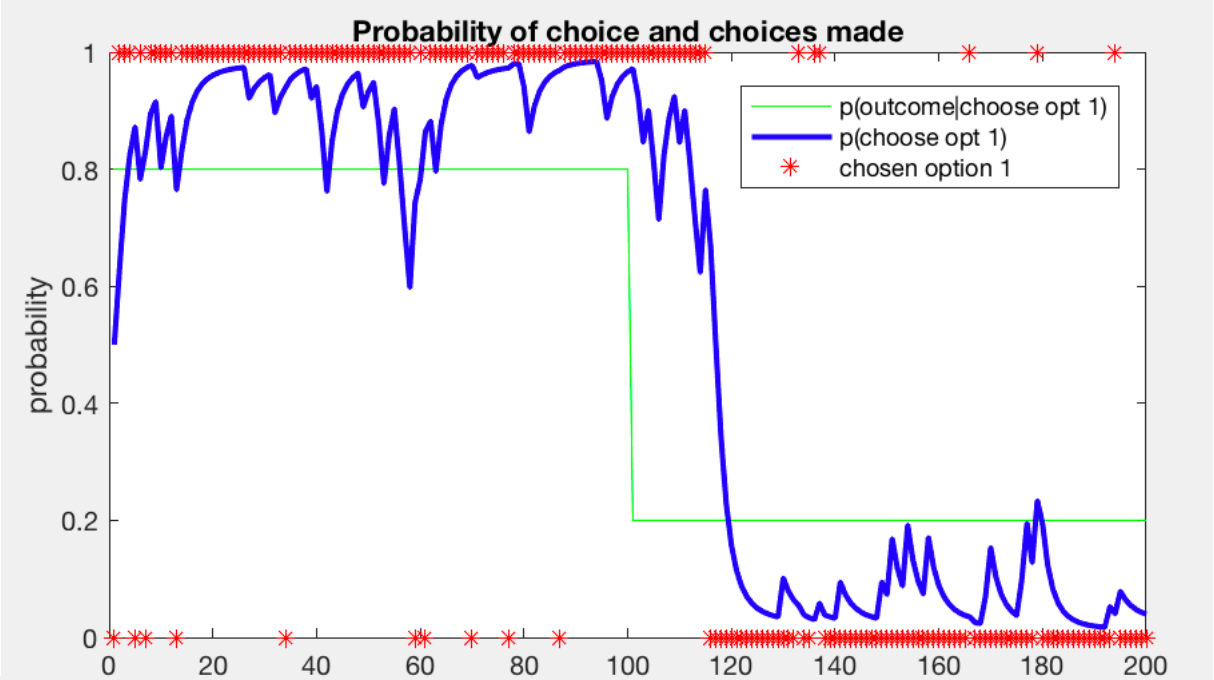
- Problem: inflexible learning rate, or generally high learning rate?

The Inverse Problem: Modelling Fitting

RW + Softmax

alpha = 0.2

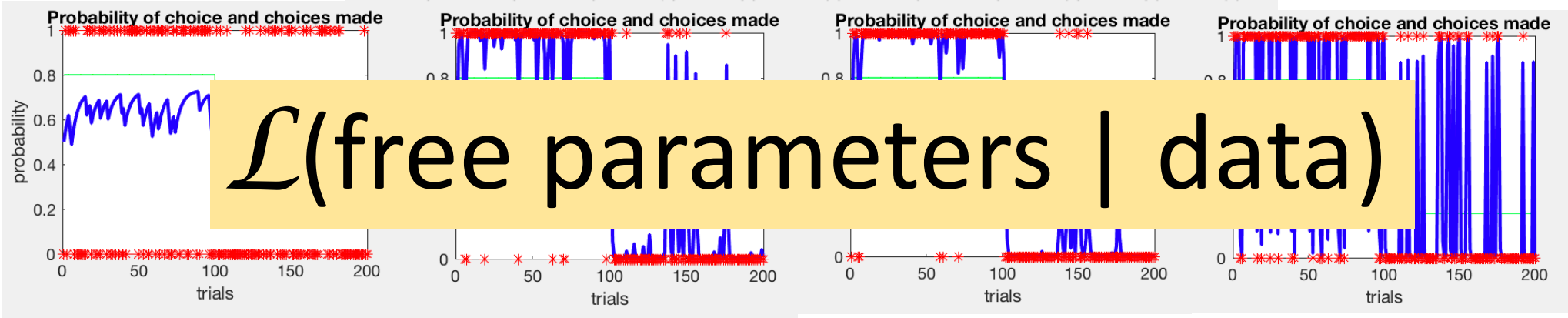
beta = 6



RW + Softmax?

alpha = ?

beta = ?



$\mathcal{L}$(free parameters | data)

Model fitting in MATLAB

Expected value => Probability (choose A, or choose B?)

Choice => probability

Probability => choice

Receive outcome => update expected value

```
%% response model
% probability of action 1
p1 = 1./(1+exp(-beta*(v(t,1)-v(t,2))));
p1_store(t) = p1;
% probability of action 2
p2 = 1-p1;
% store probability of the chosen action
a = actions(t); % action on this trial (1 or 2)
if a==1
    p(t) = p1;
elseif a==2
    p(t) = p2;
end
%% learning
%update, regardless of outcome

delta  = outcome(t) - v(t,a);
v_trial = v_trial + (alpha*delta);
```

```
%% choice/response model: decision based on knowledge from previous trials
EV_opt1 = v(t,1);%column 1 is about option 1
EV_opt2 = v(t,2);% value for option 2 across each trial
%compare between two options, and output choice probability
pchoice_opt1 = (1+exp(-beta*(EV_opt1-EV_opt2))).^-1;
pchoice_opt2 = 1-pchoice_opt1;%either choose opt1, or opt2
pchoice(t,:) = [pchoice_opt1,pchoice_opt2];%record the choice probs

%choice behaviour... credits: written by Hanneke den Ouden 2015 <h.denouden@gmail.com>
% Do a weighted coinflip to make a choice: choose stim 1 if random
% number is in the [0 p(1)] interval, and 2 otherwise
if rand(1) < pchoice(t,1) %bigger the pchoice, more likely to choose "1"
    choice(t) = 1;
else
    choice(t) = 2;
end

%% learning
%PE: observe both options, update both options
PE(t,:) = outcomes(t,:) - v(t,:);%for first trial, note change from baseline
%update expectation for next trial
v(t+1,:) = v(t,:) + alpha*PE(t,:);
```

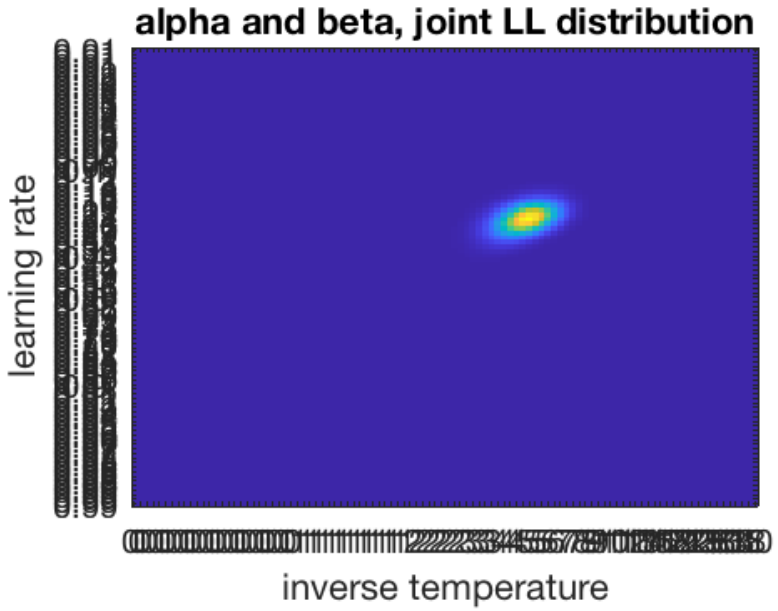
Essential code for fitting
(What happens within each trial)
`lik_RW1lr_PL.m`

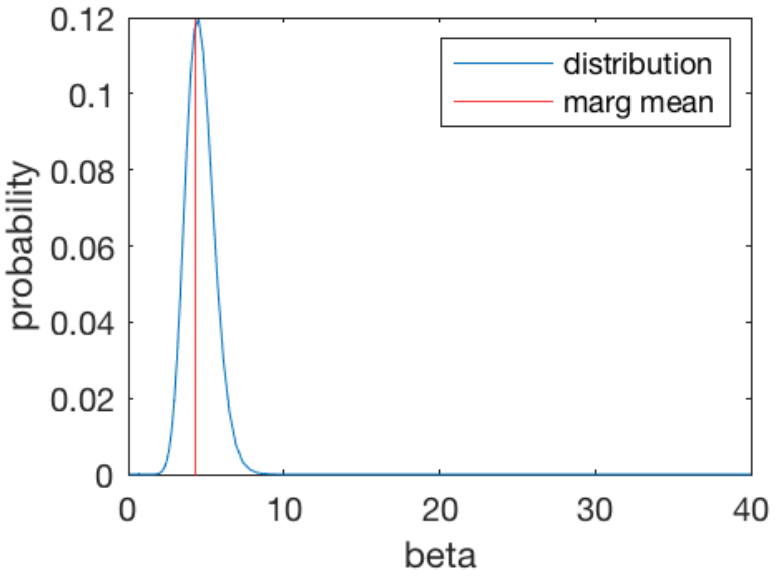
Code for simulating each trial

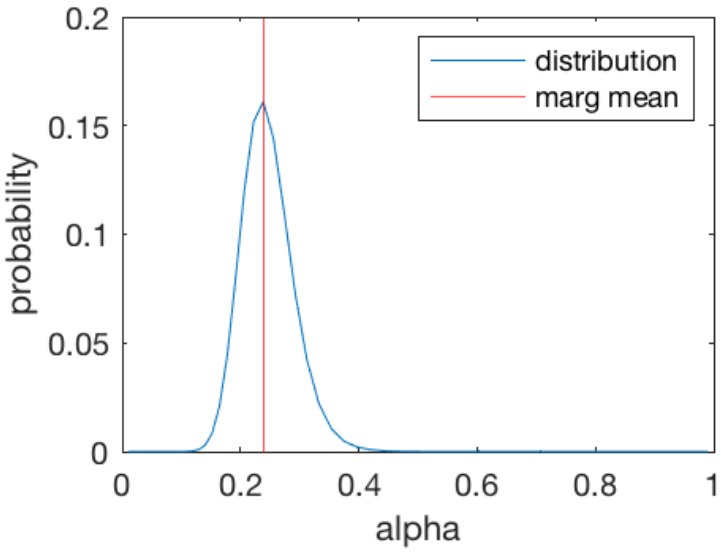
Constructing the likelihood function

- Likelihood: multiply the probabilities obtained on each trial (in practice, add the log of probabilities together)
- `negLLfun = @(x) lik_RW1lr_PL(x,actions,outcomes);`

input variable ↑ ↑ Data → fixed parameters

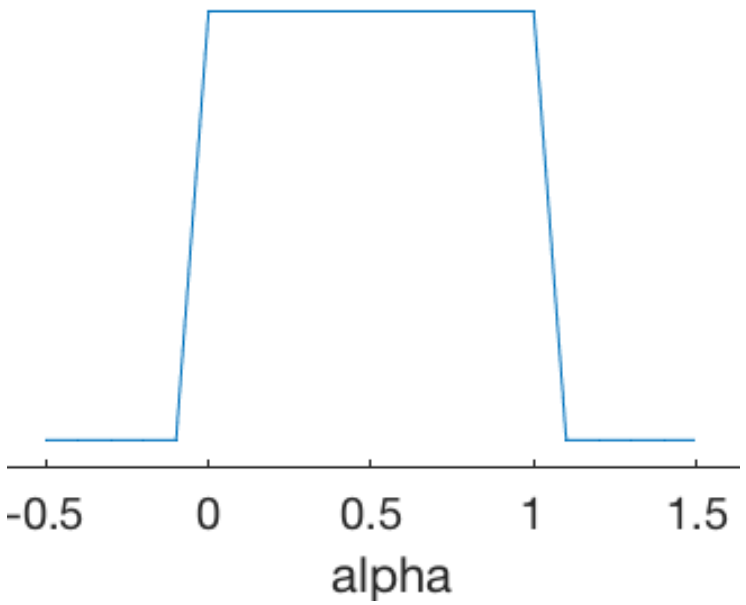


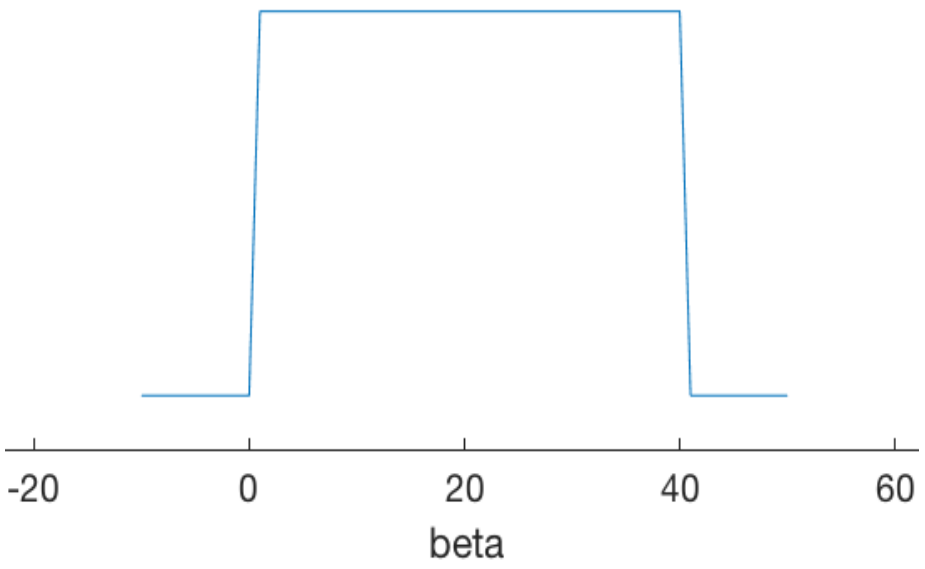




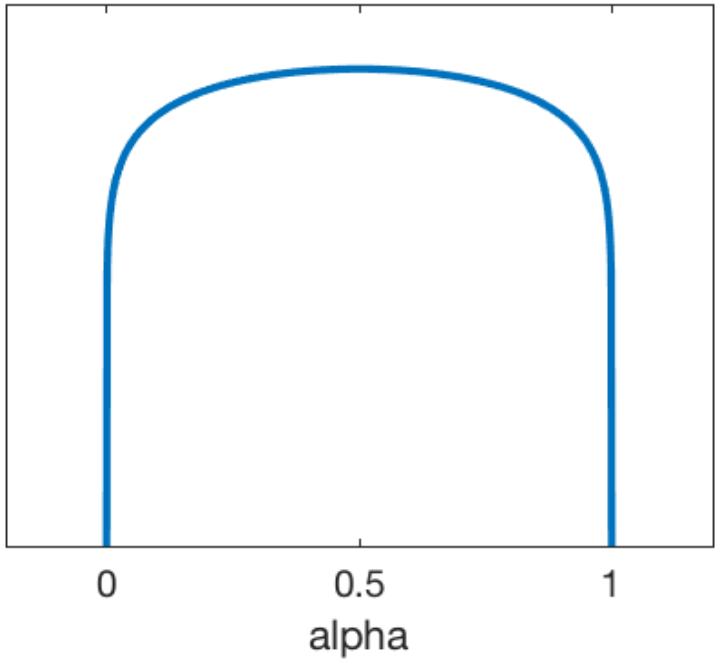
Inclusion of priors

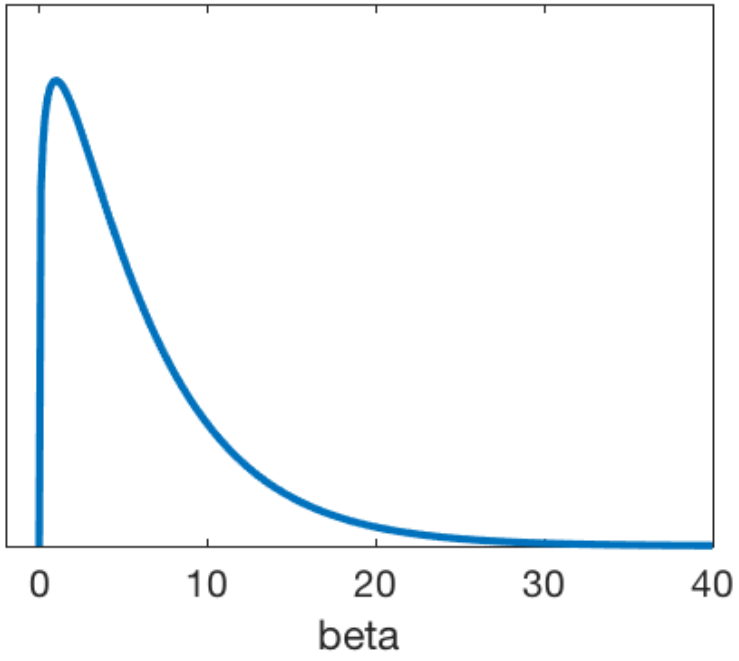
```
alphabins = 100;
betabins = 100;
%created in logit space or log etc.
a_range = logit(0.01):(logit(0.99) - logit(0.01))/(alphabins-1):logit(0.99);
b_range = log(0.1):(log(40)-log(0.1))/(betabins-1):log(40);
```





More informative priors



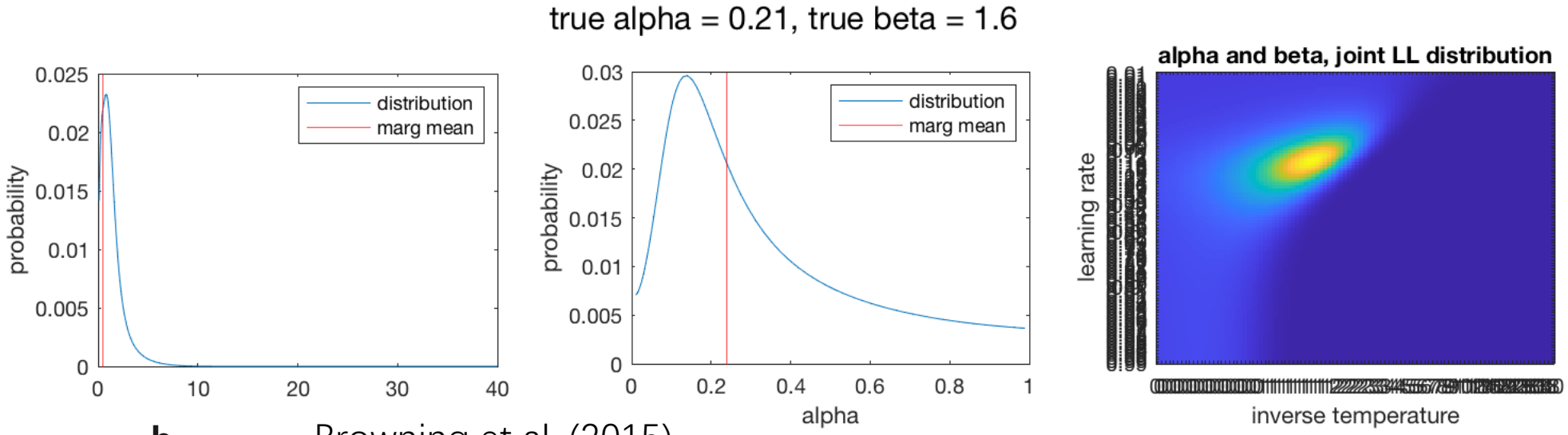


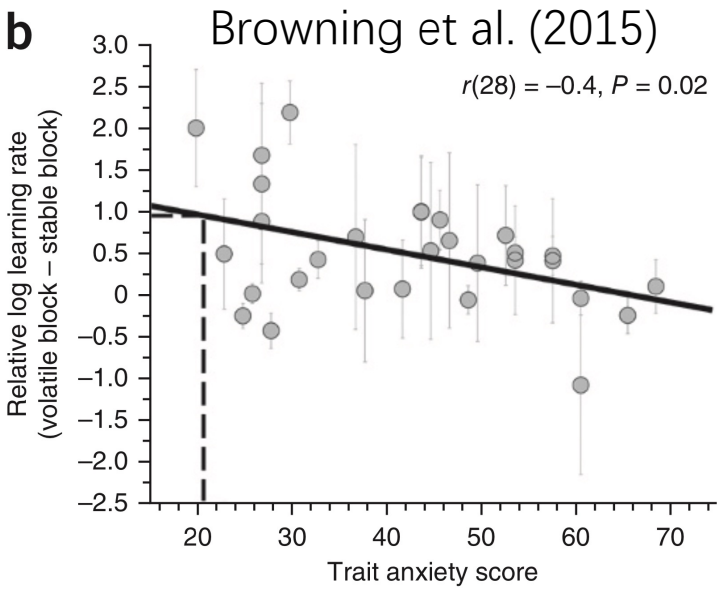
Easy implementation in MATLAB:

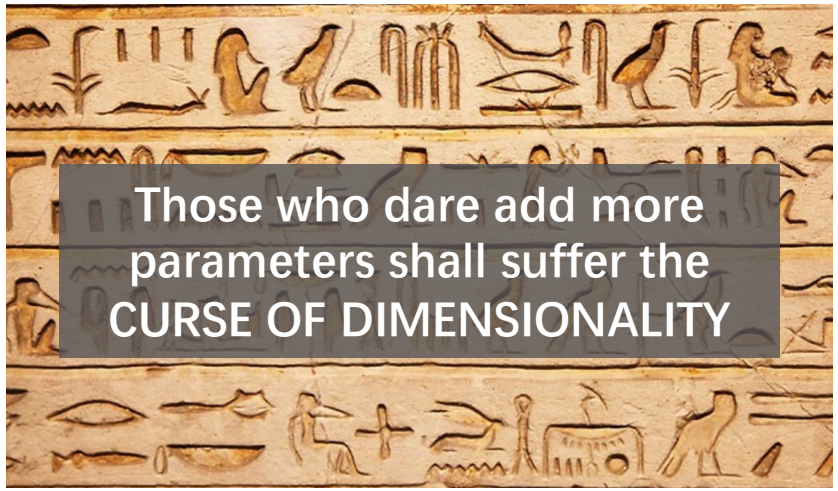
```
loglik= loglik+log(pdf('beta', alpha, 1.1,1.1));
loglik= loglik+log(pdf('gam', beta, 1.2, 5));
```

Nussenbaum et al. (2021)

Maximum likelihood versus mean posterior?

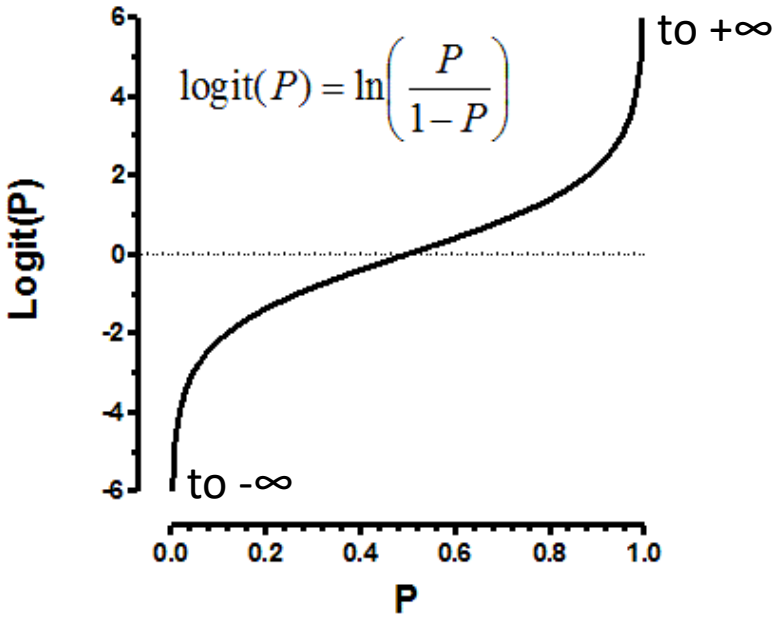


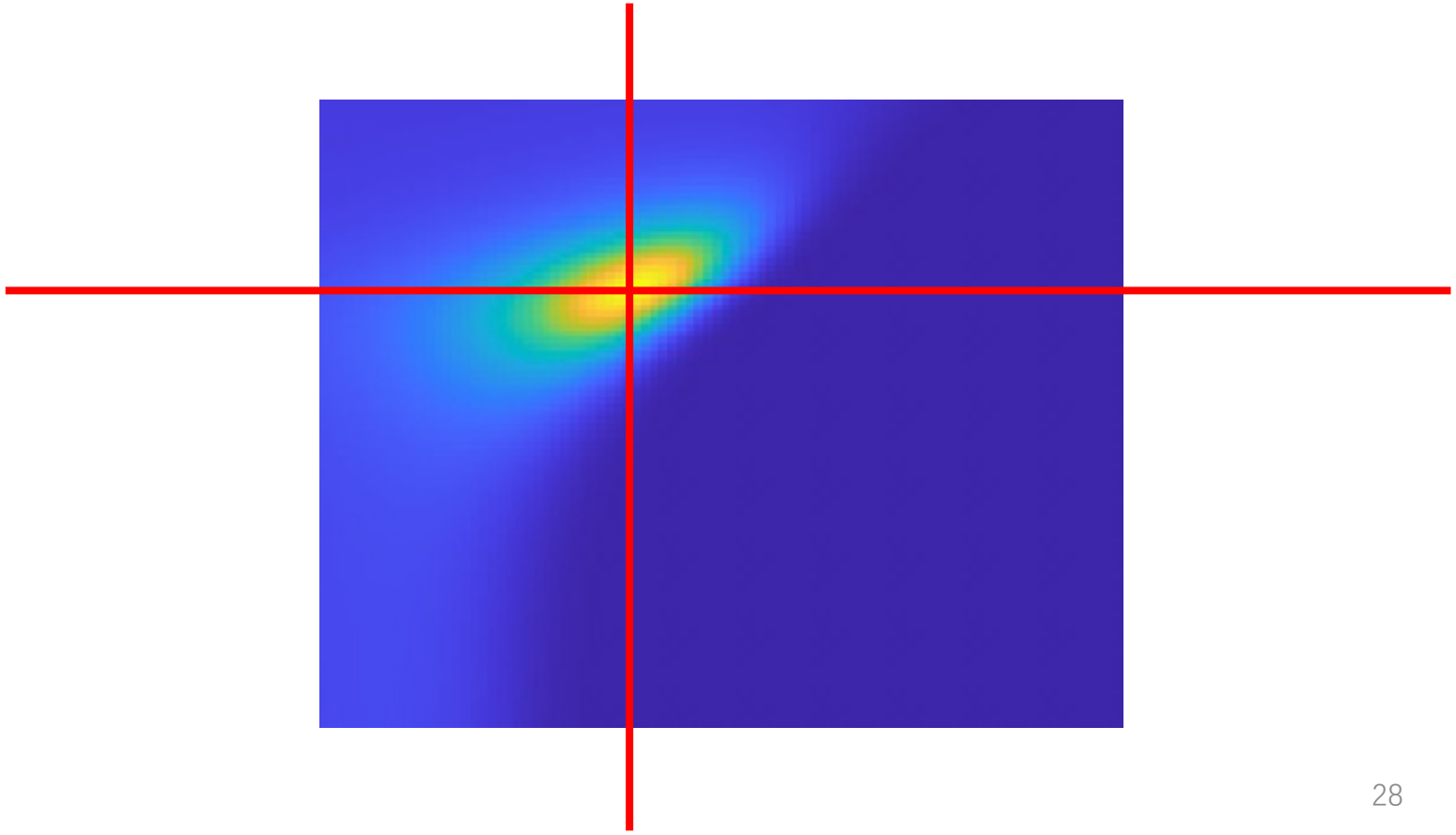




Optimisation algorithm for MLE (and MAP)

```
negLLfun = @(x) lik_RW1lr_PL(x,actions,outcomes);
params = fmincon(negLLfun,initial,[],[],[],[],lowerbound, upperbound);
alpha = params(1);
beta = params(2);
```





Parameter recovery (to motivate debugging)

- Run (many) simulations with realistic parameter values

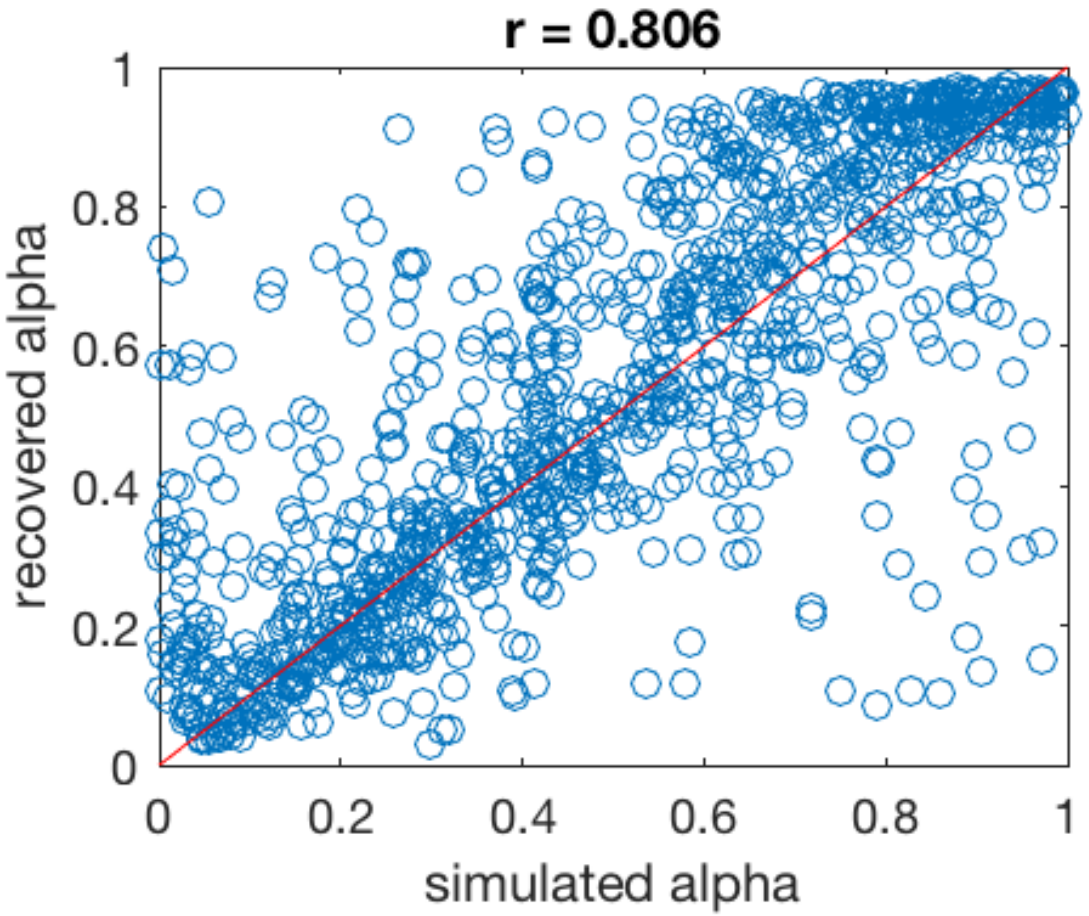
```
alpha_range = unifrnd(1e-4,1,[1,nsimus]);
beta_range = unifrnd(1e-4,20,[1,nsimus]);
```

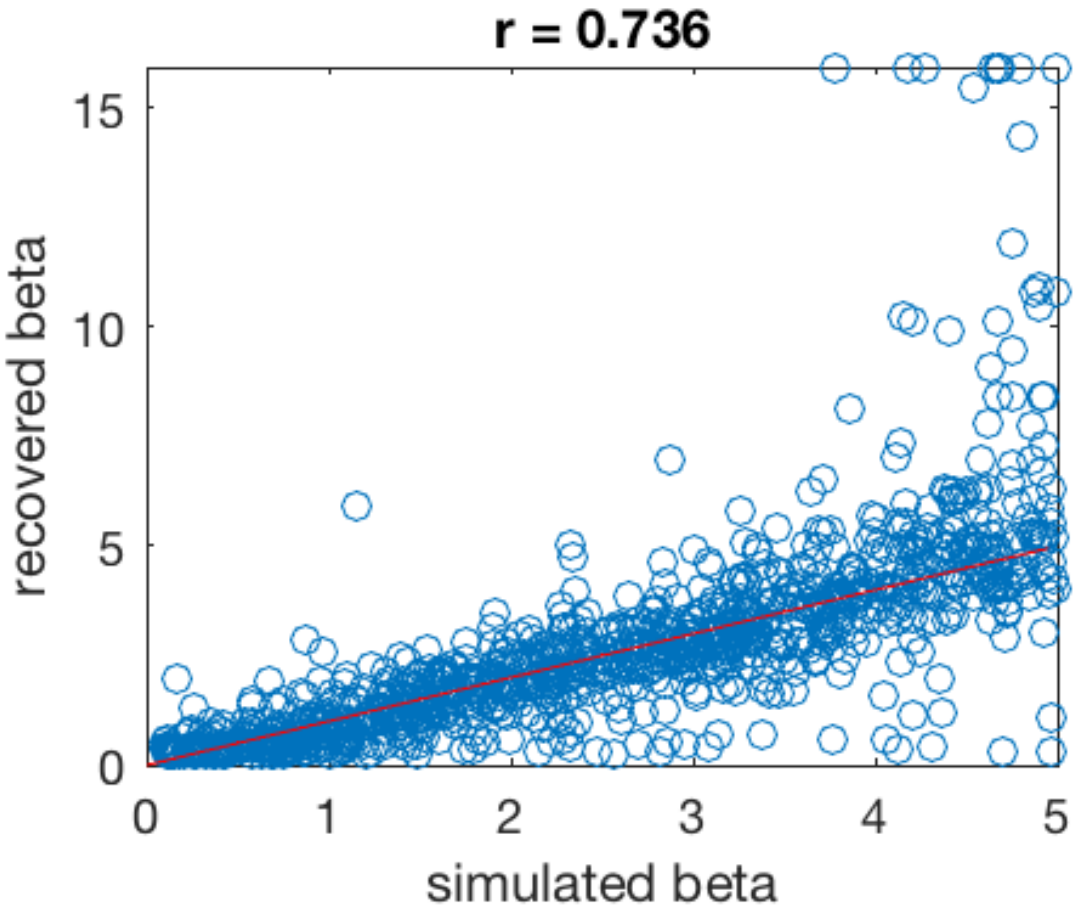
- Estimate parameters for the simulated data (known ground truth)

- Goal: r > 0.7, with little correlation amongst different parameters

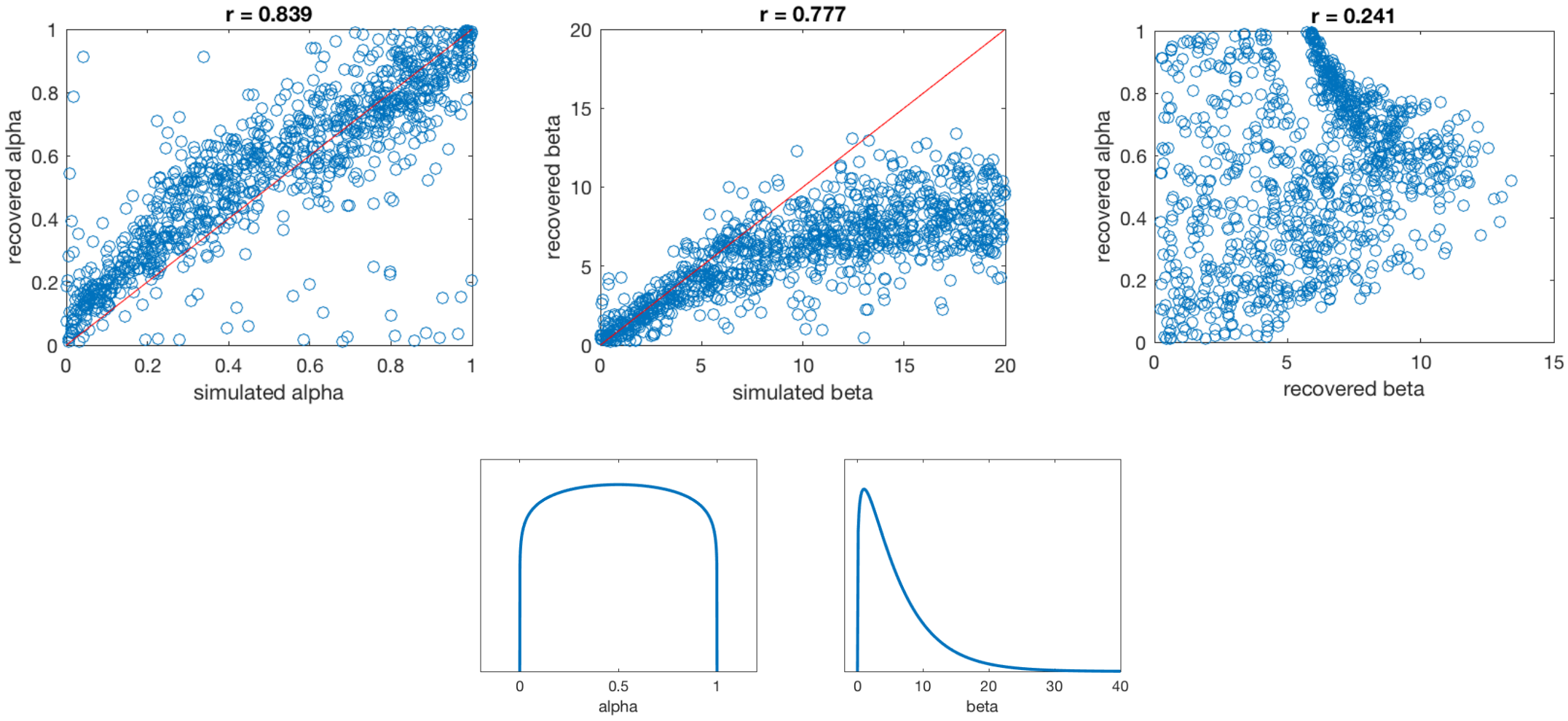
Wilson & Collins (2019)

Parameter recovery with mean posterior





Parameter recovery: priors can be helpful

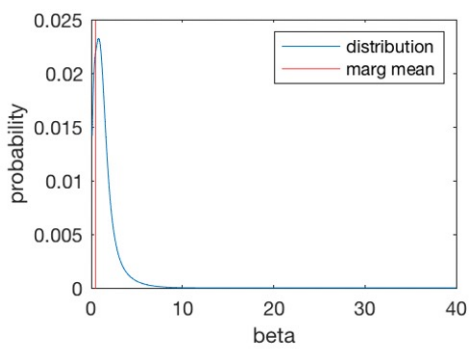


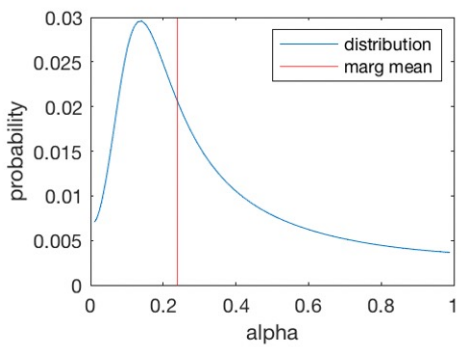
Interim Summary: Three Big Decisions

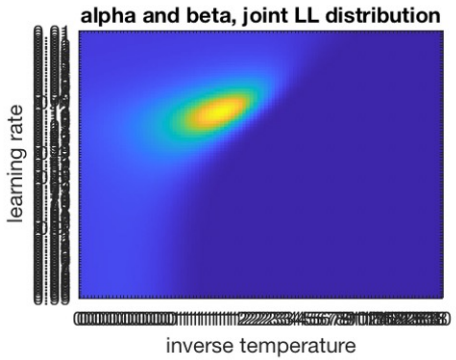
- Model representation (the equations / the generative process)

$$V_{t+1} = (1\text{-}\,\alpha) * V_t + \alpha * o_t \qquad P(\text{choose bandit k}) = \frac{e^{\beta v_t^k}}{\sum_{j=1}^{n} e^{\beta v_t^j}}$$

- Evaluation method (likelihood or posterior / accuracy in prediction)

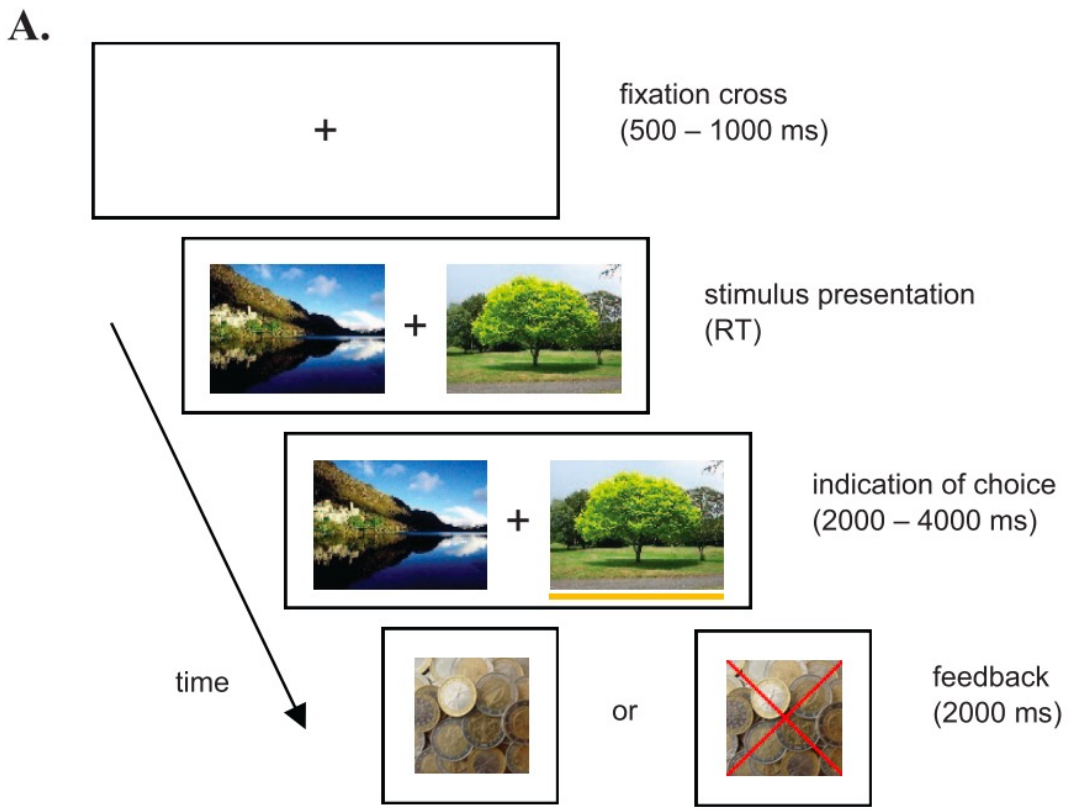


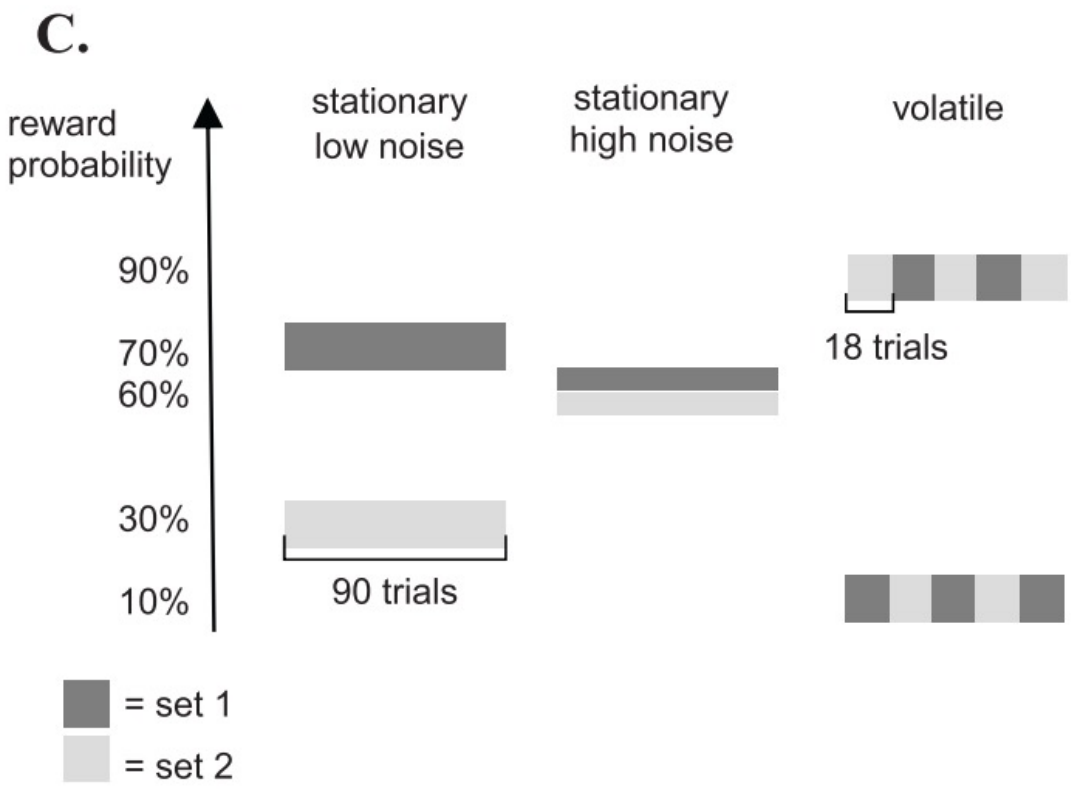




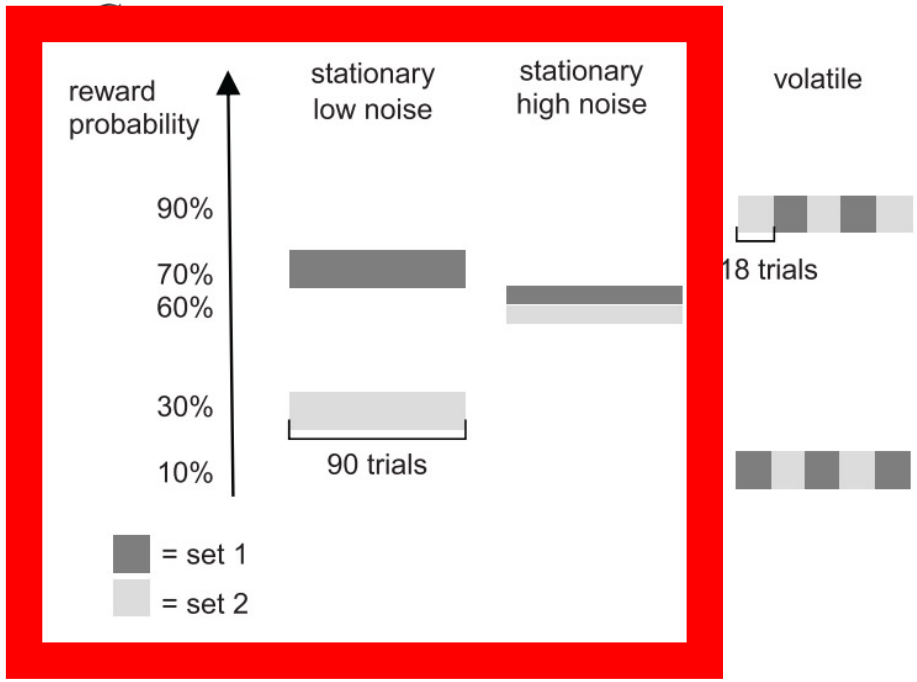
- Inference / Optimisation (MAP/ mean posterior / numerical methods)

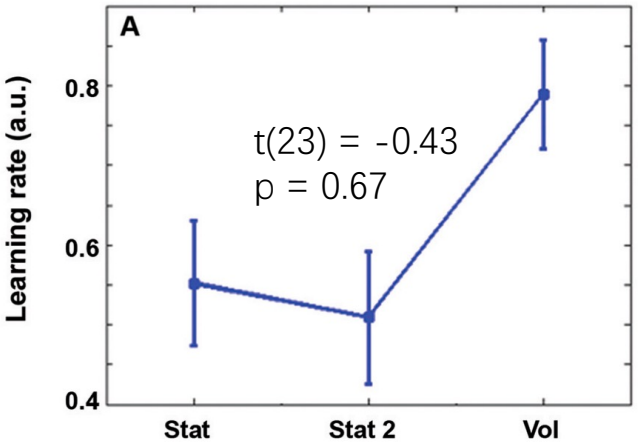
Goris et al. (2020) – High & Low Noise



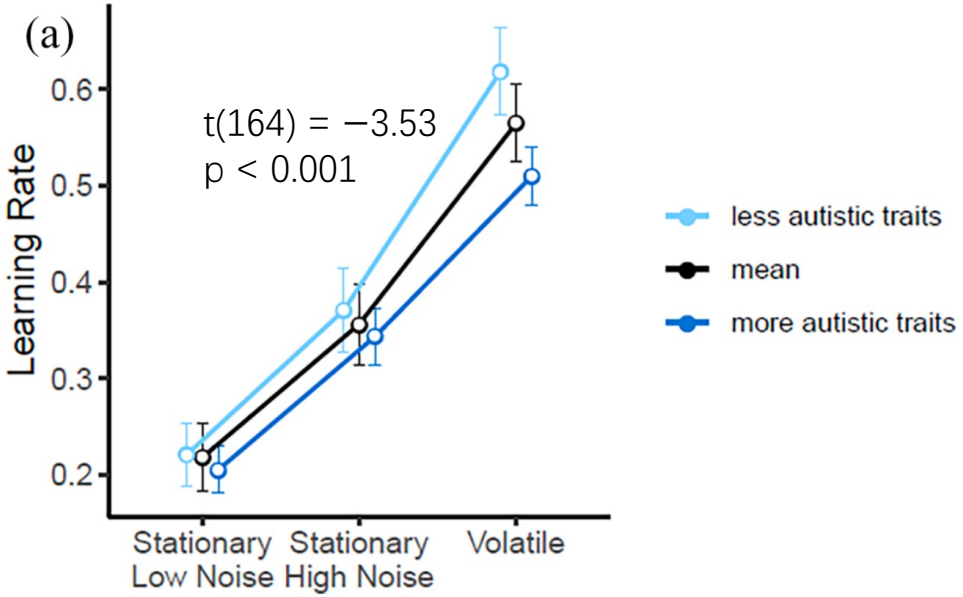


Effect of noise on learning rates?





Silvetti et al. (2013)



Goris et al. (2020)

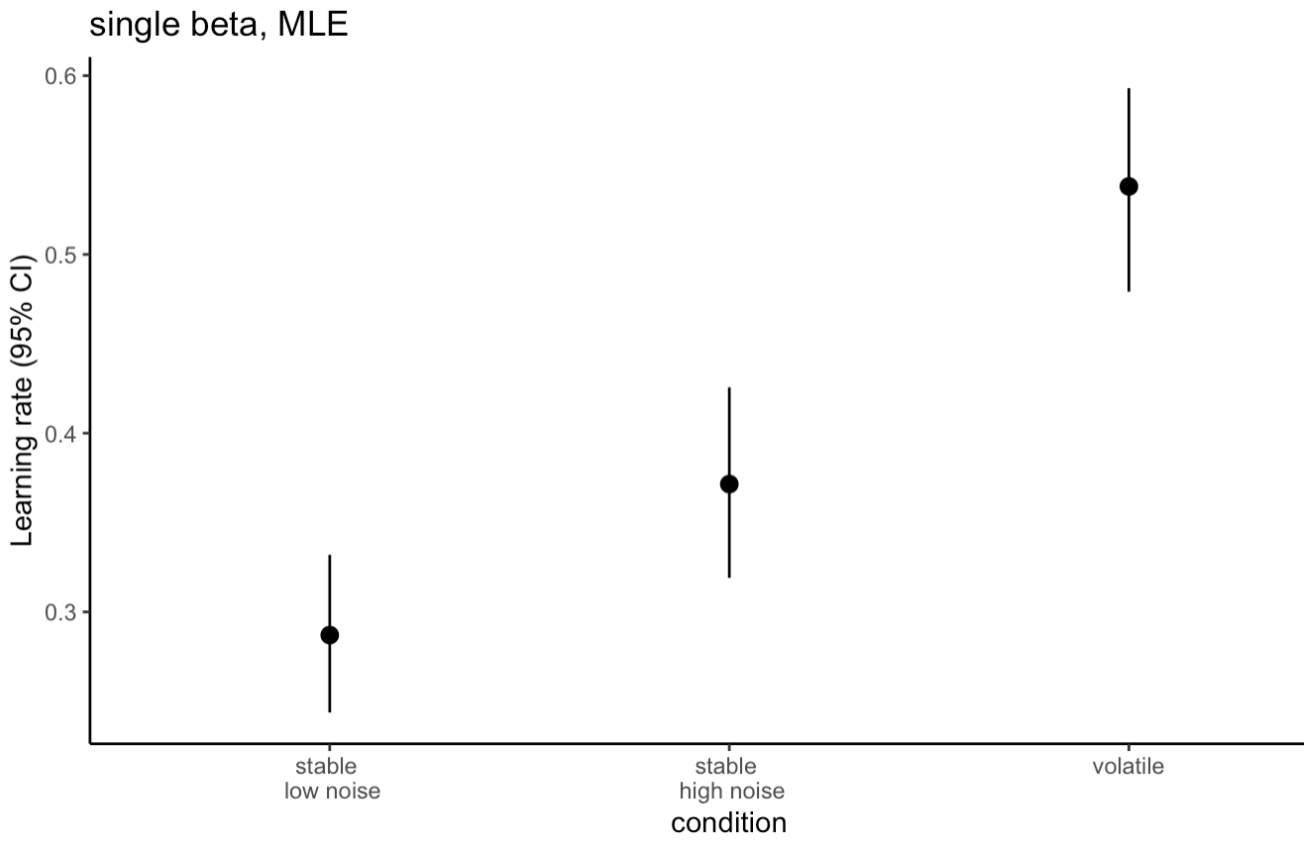
Comparisons / Decision tree

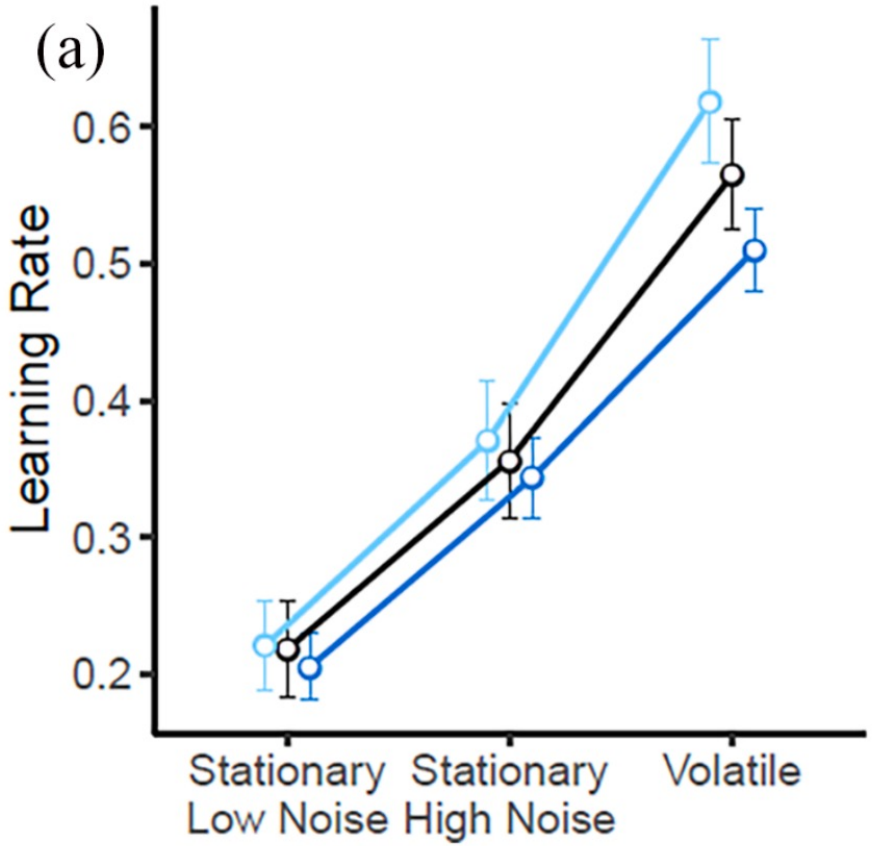
- Mean or mode of distribution?
- With or without informative priors?
- Single model or three separate fits?

$$P(\text{choose bandit k}) = \frac{e^{\beta v_t^k}}{\sum_{j=1}^{n} e^{\beta v_t^j}}$$

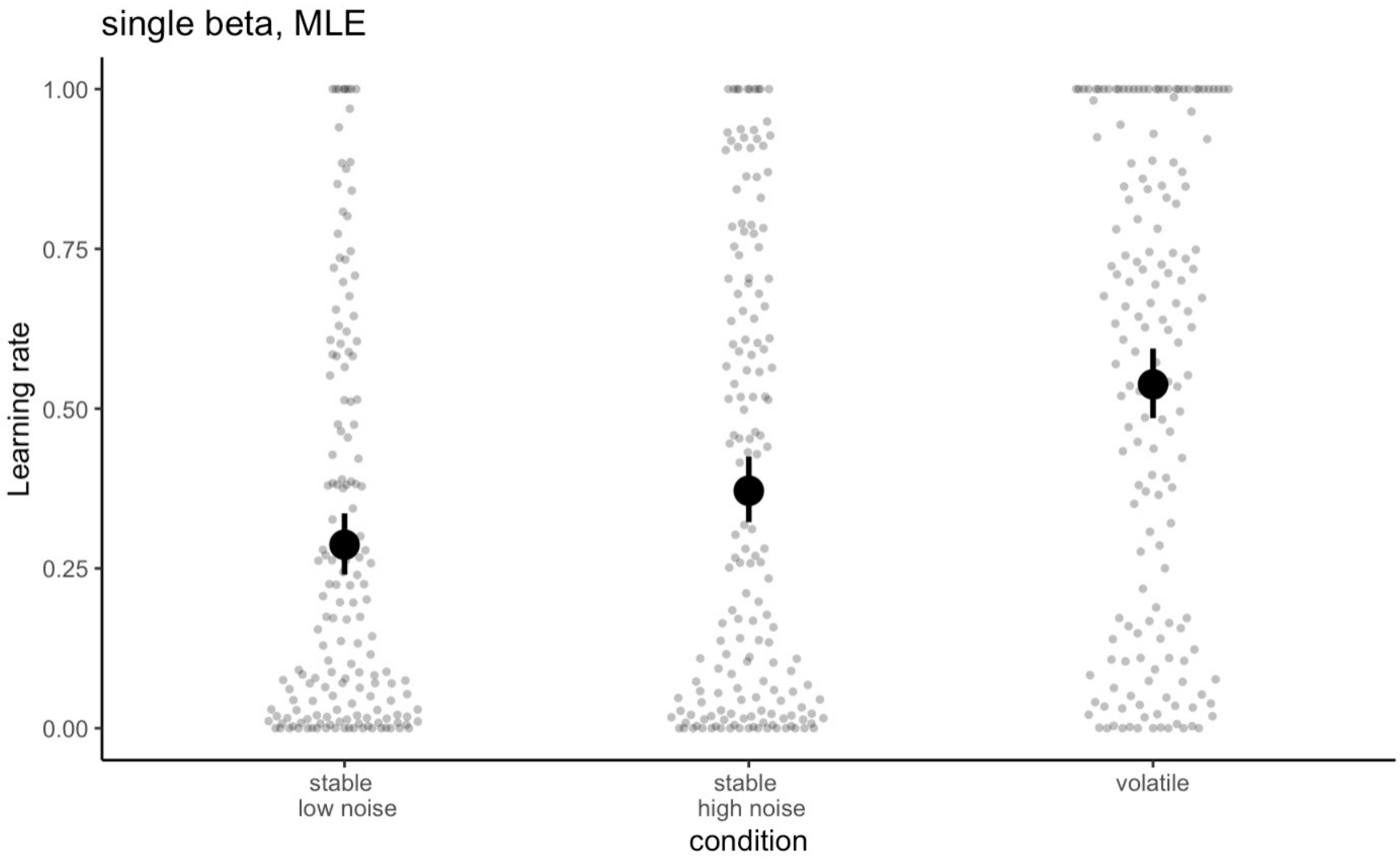
```
if condition(t) == 1
    v_trial = v_trial + (alpha1.*PE(t,:));
elseif condition(t) == 2
    v_trial = v_trial + (alpha2.*PE(t,:));
elseif condition(t) == 3
    v_trial = v_trial + (alpha3.*PE(t,:));
end
```

Assume constant inverse temperature

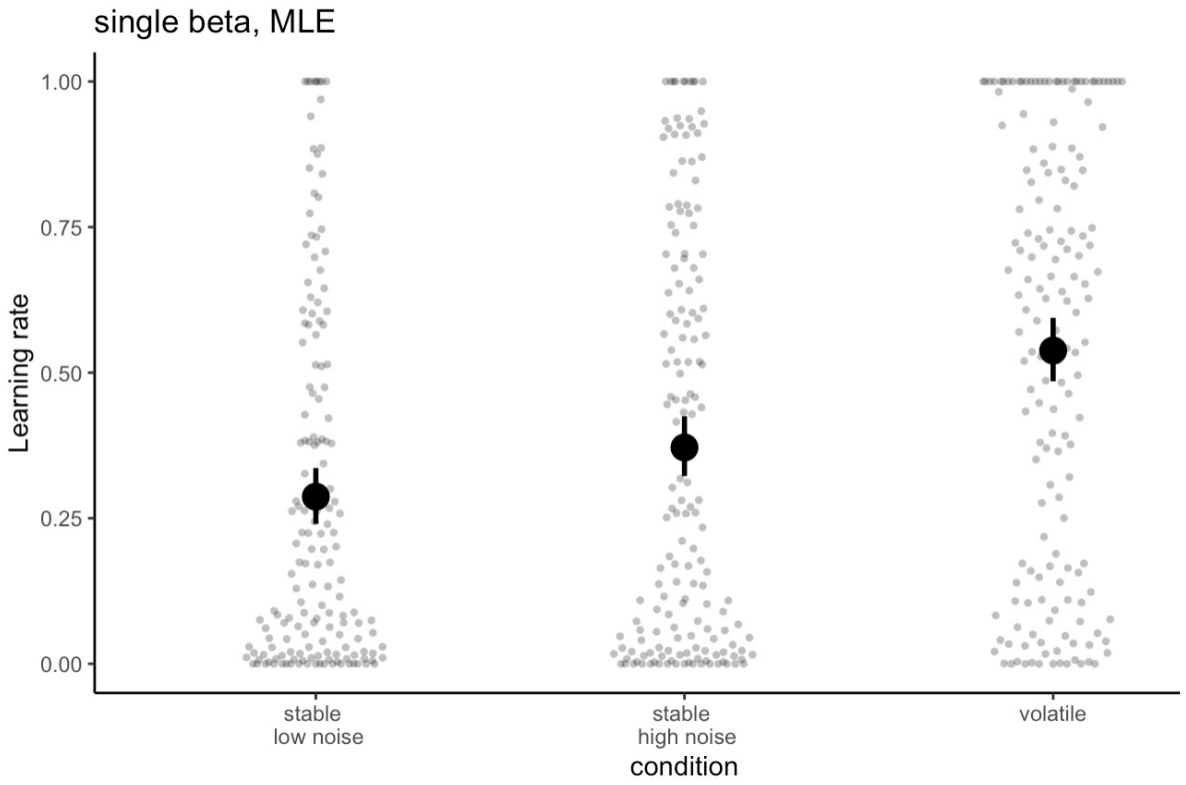


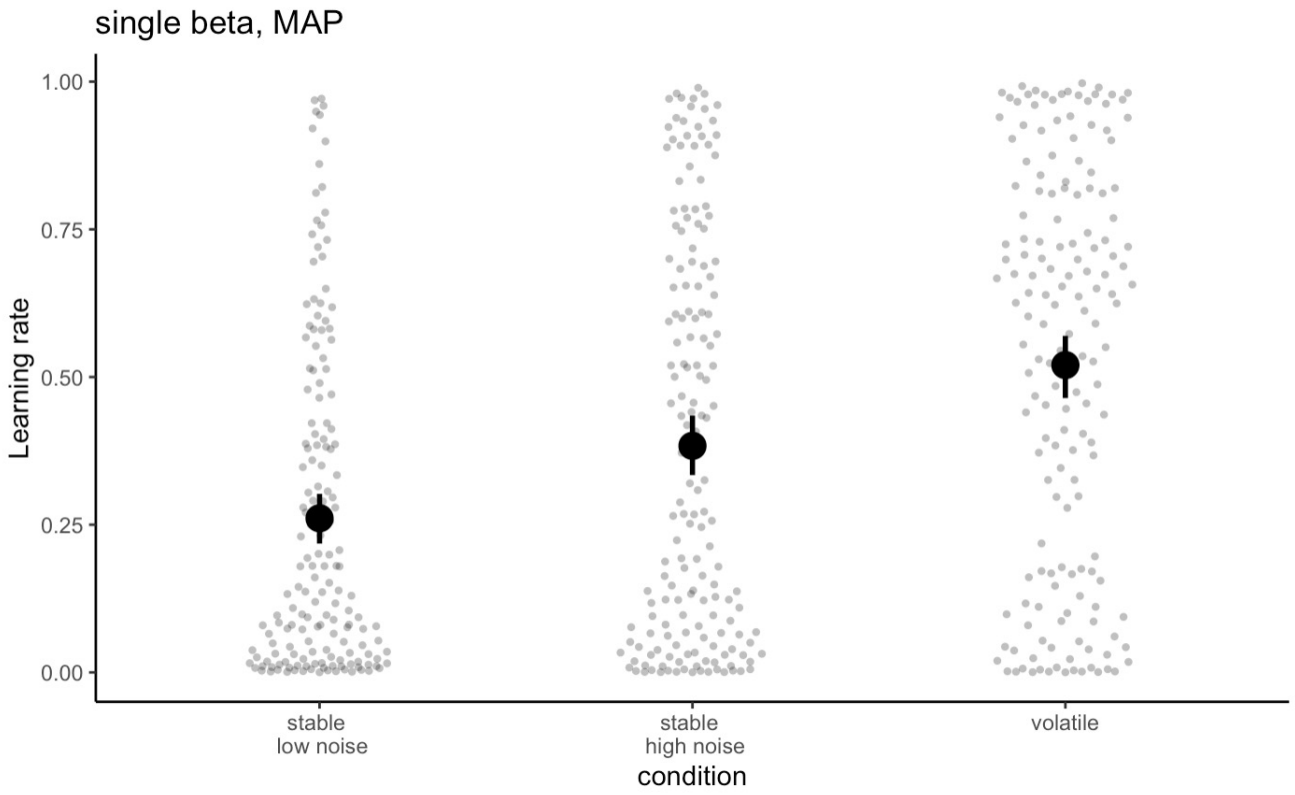


However, we see an ominous sign…



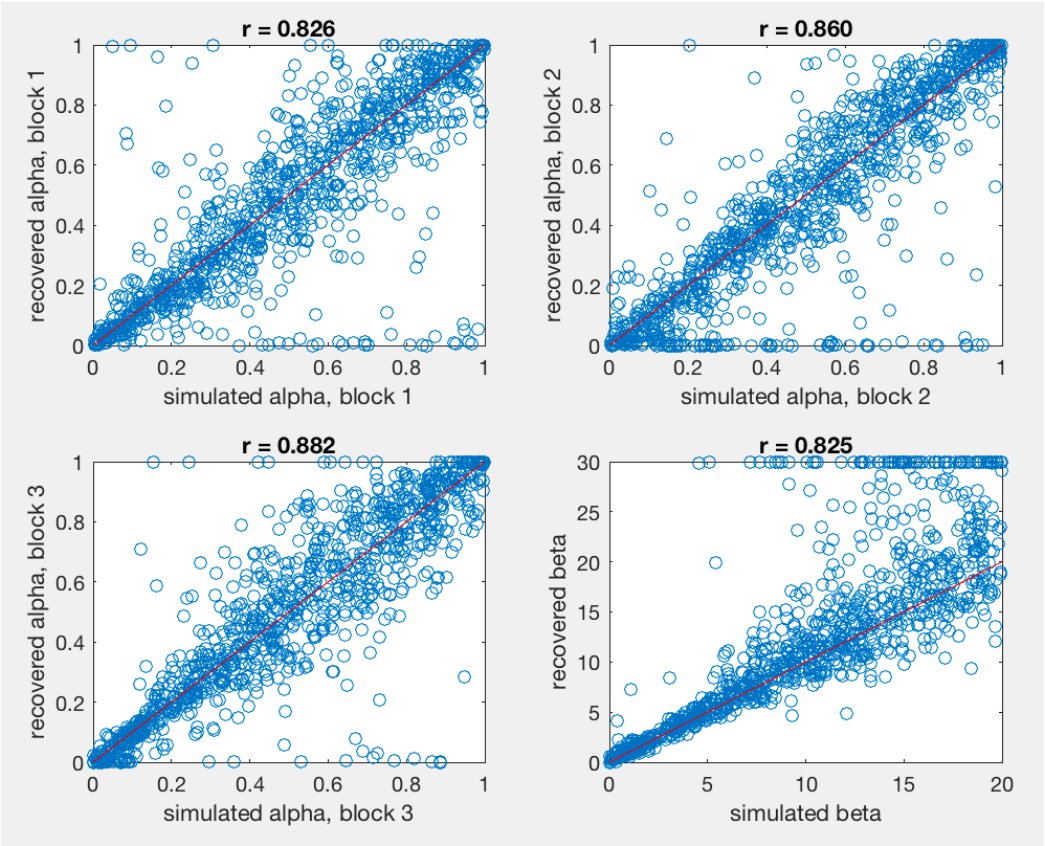
MLE versus MAP



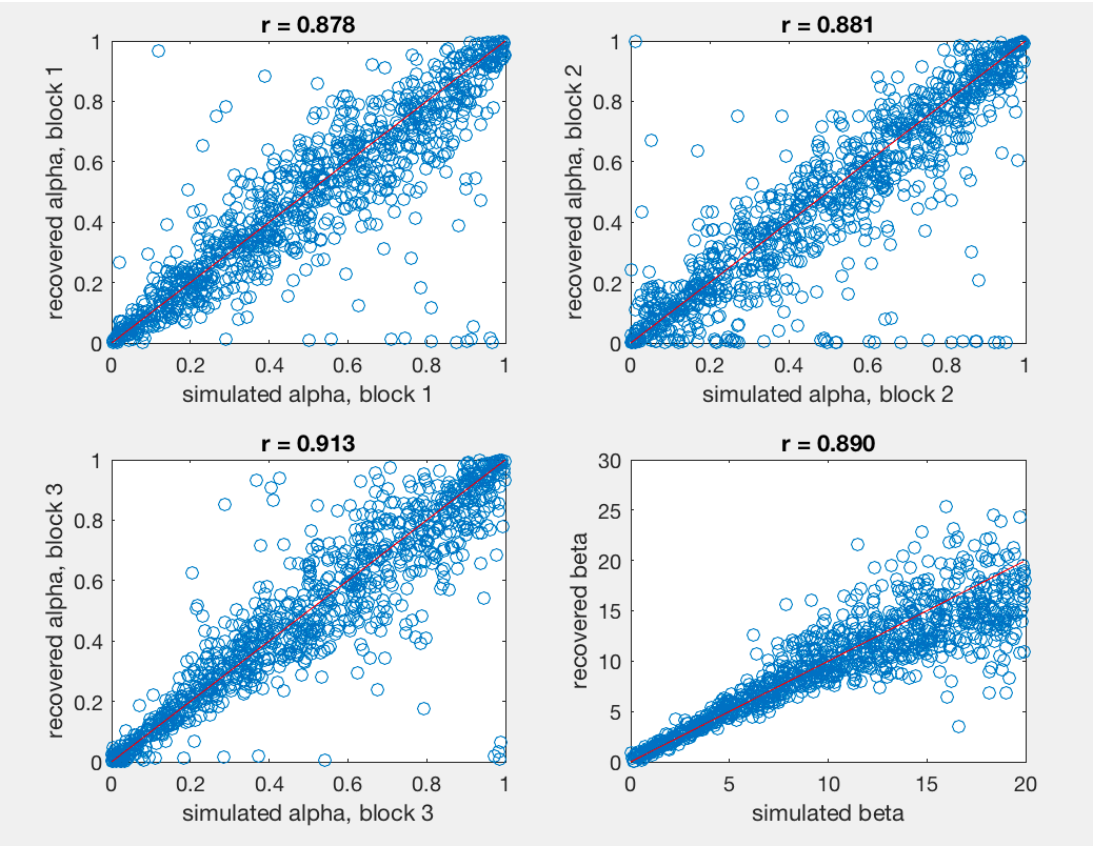


MLE versus MAP: parameter recovery

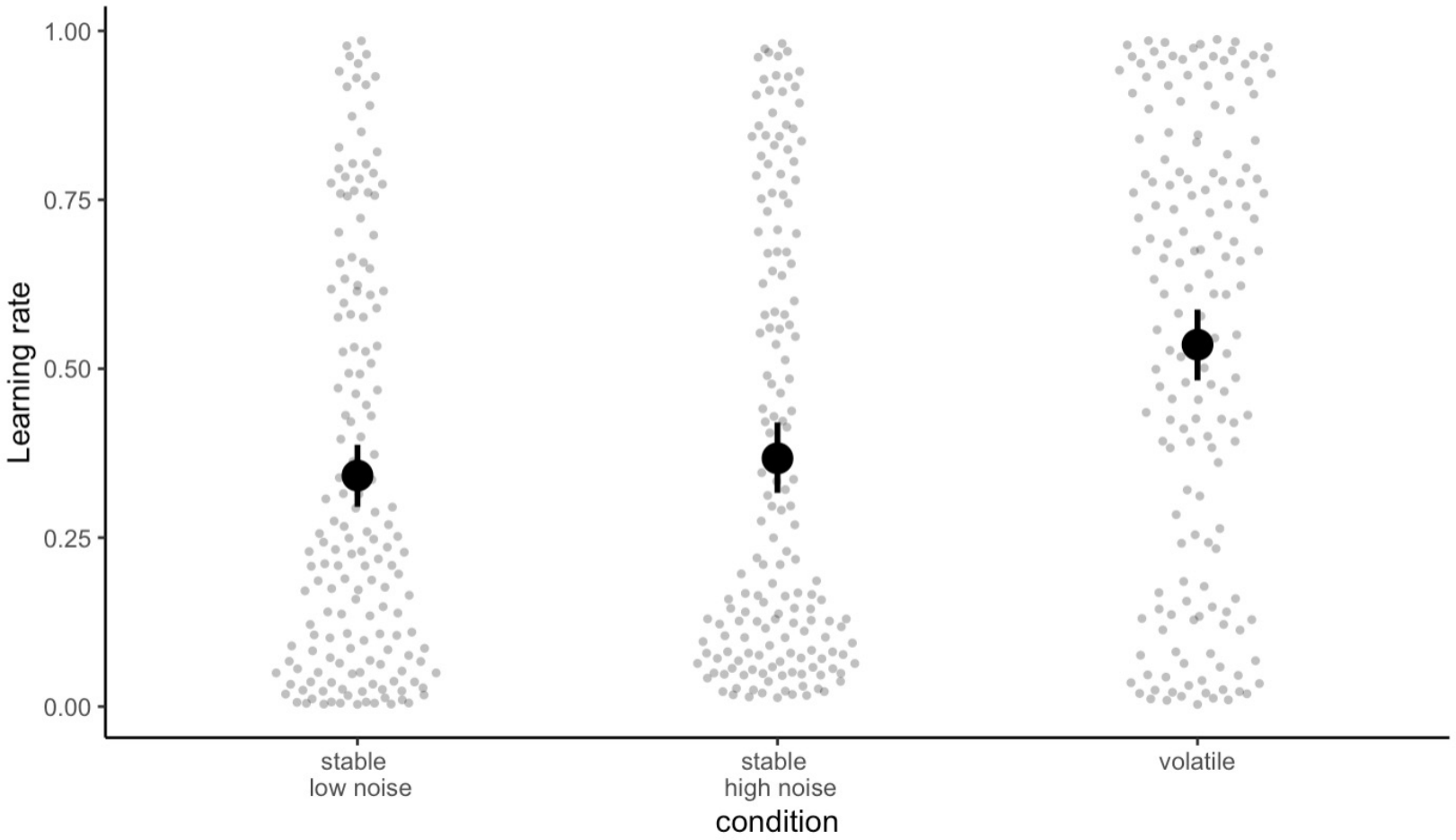
`PL_RW_multiblock_fmin_Recovery`



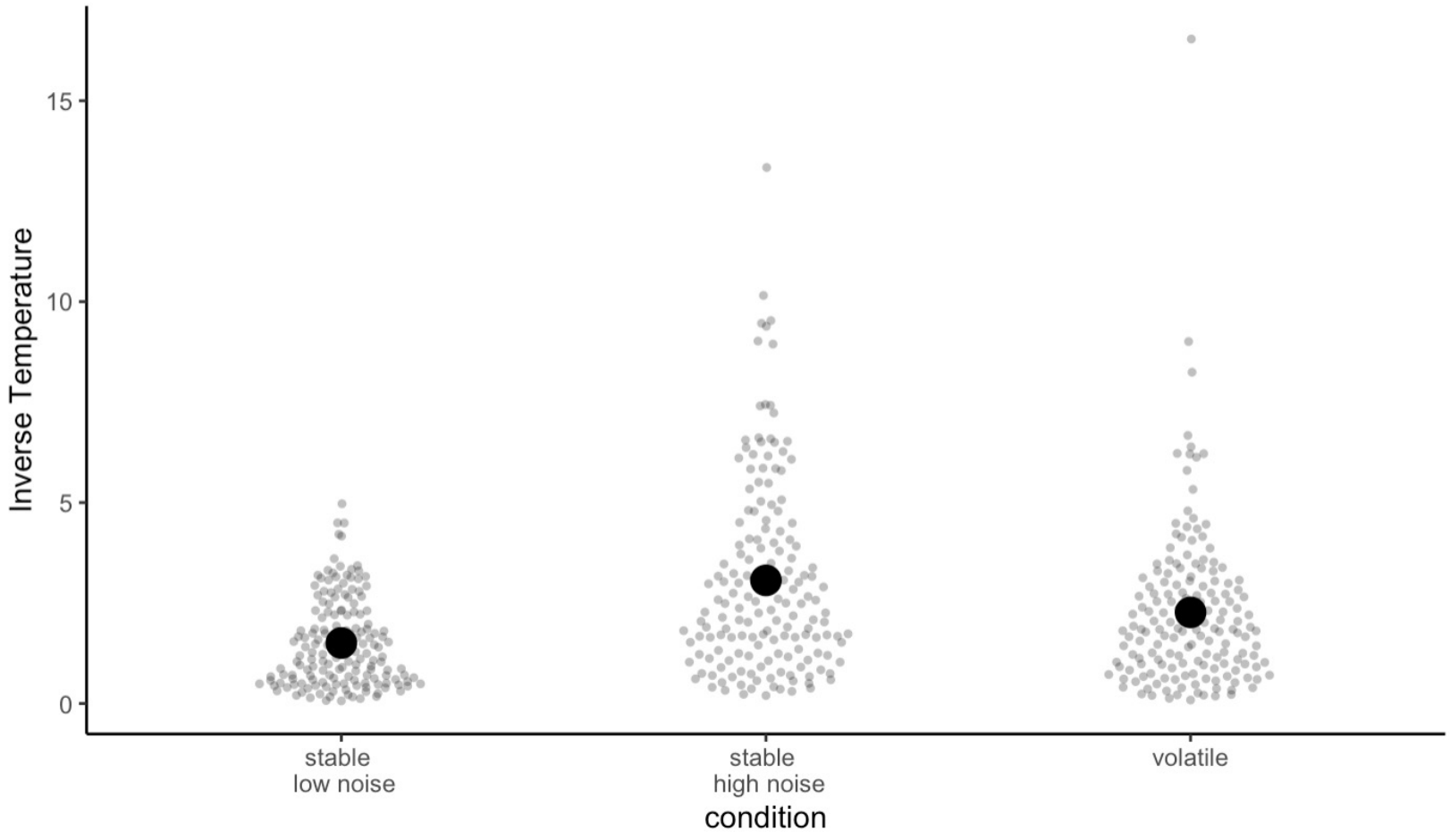
`PL_RW_multiblock_fmin_Recovery(true)`

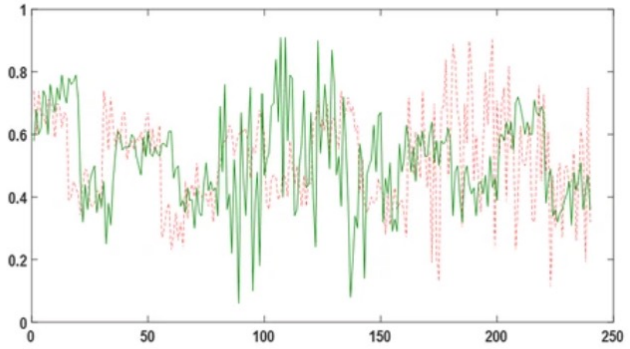


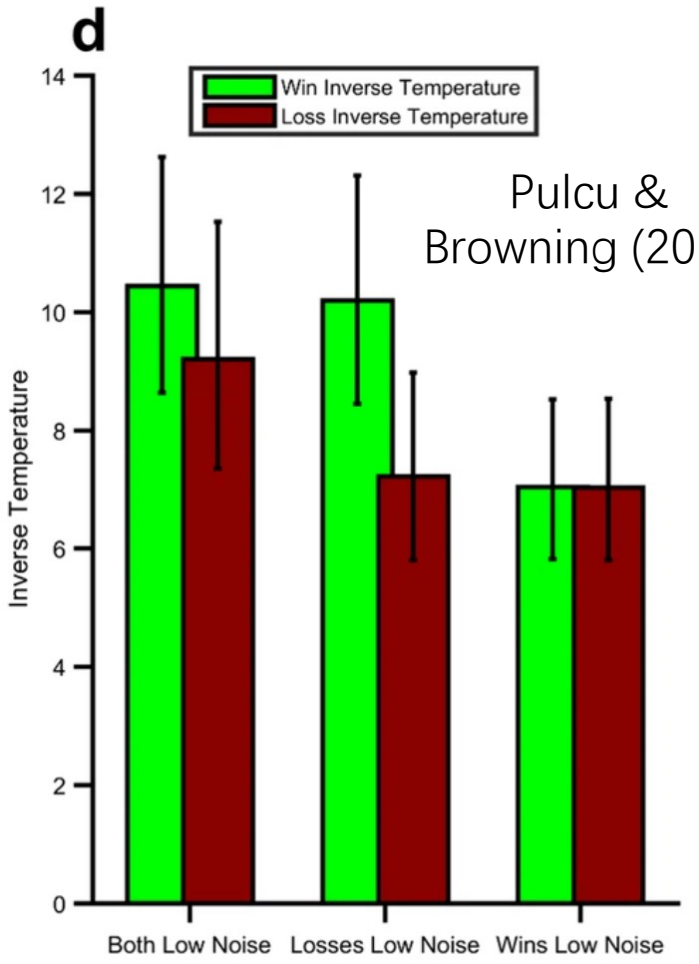
MAP, three different inverse temperatures



Changing value sensitivity?



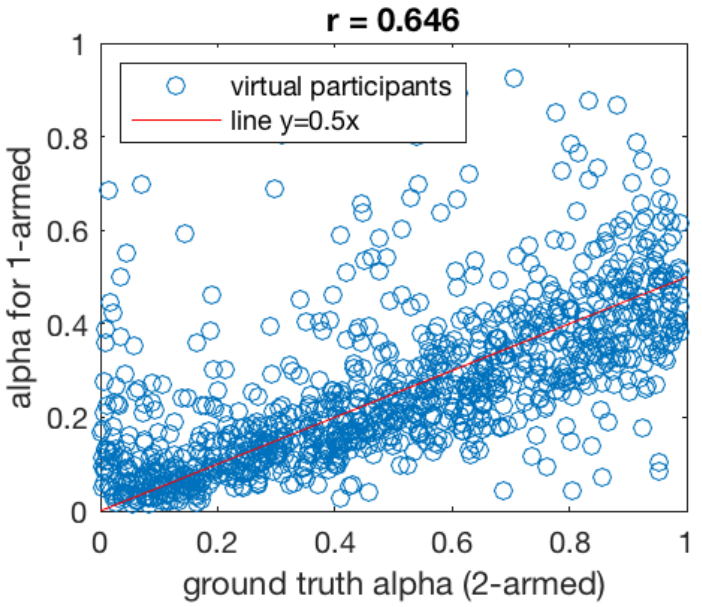


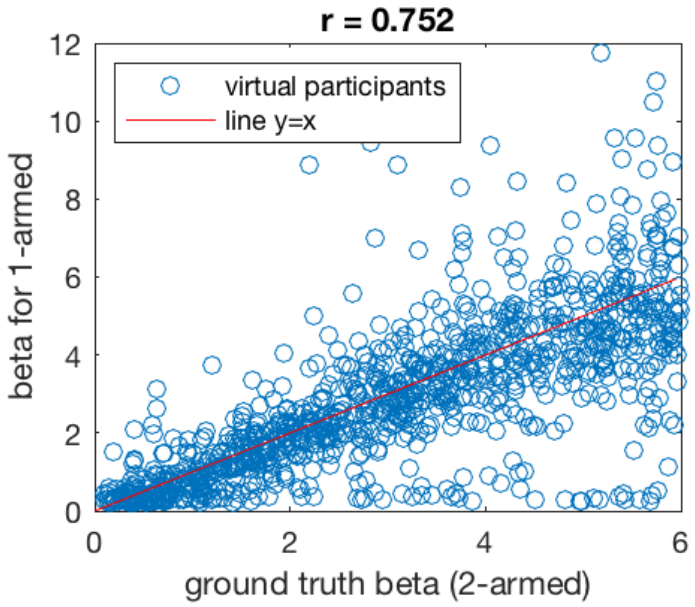


Pulcu & Browning (2017)

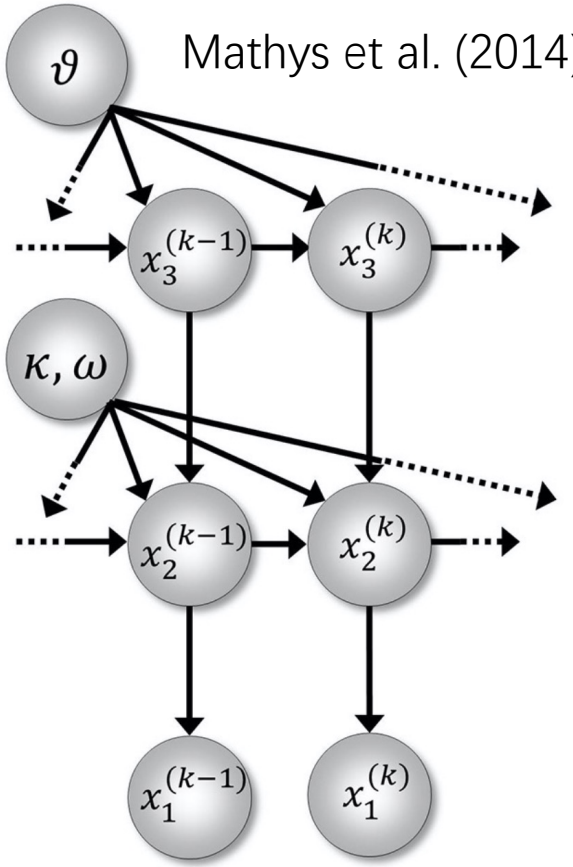
What if participants do not consider the task as a one-armed bandit?

- @No counterfactual update
- Expectation: learning rate halved
- Simulation (assume two-armed) => model fitting (one-armed):



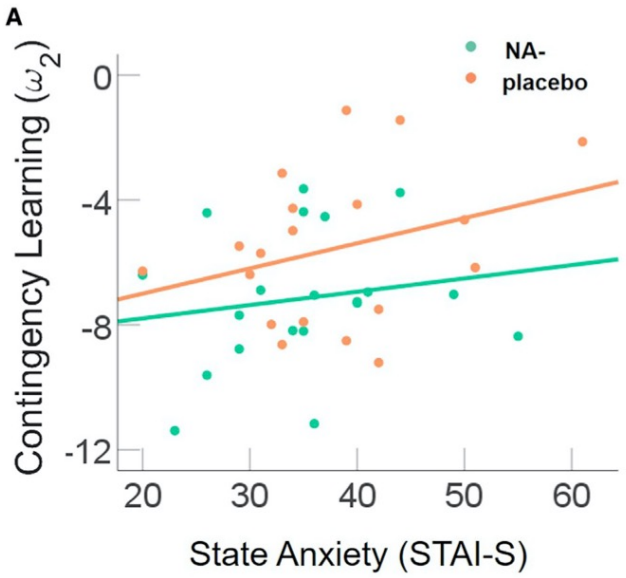


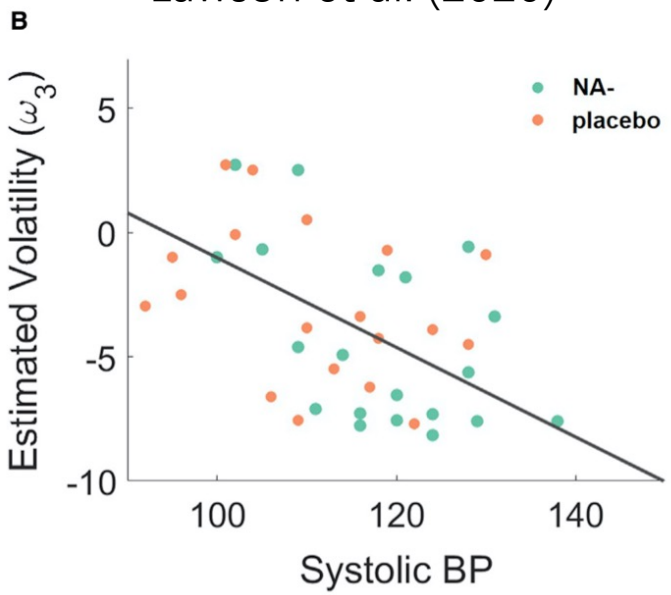
Hierarchical Gaussian Filter (HGF)



Mathys et al. (2014)

Lawson et al. (2020)





- Question: how exactly are these free parameters related to adaptive learning in stable versus volatile environments?

Overall Summary

- Model simulations can precede model fitting for understanding model properties, and for doing parameter recovery analysis

- Quantitative and falsifiable predictions about anxiety?

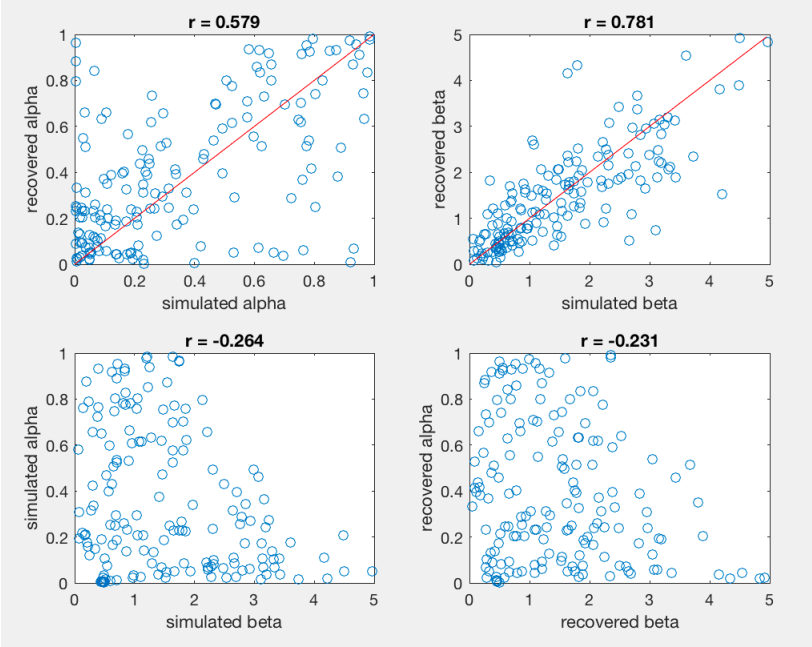
- Model fitting: construct likelihood functions → estimate the parameters → model comparison & further checks

Thank you for listening!

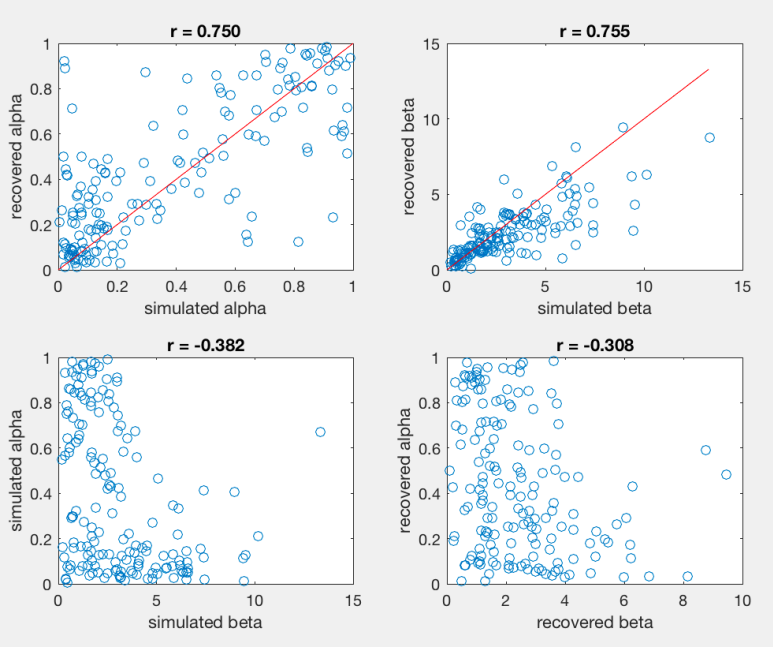
Additional Slides/Raw materials

Goris 2020 three model fits recovery (MAP)

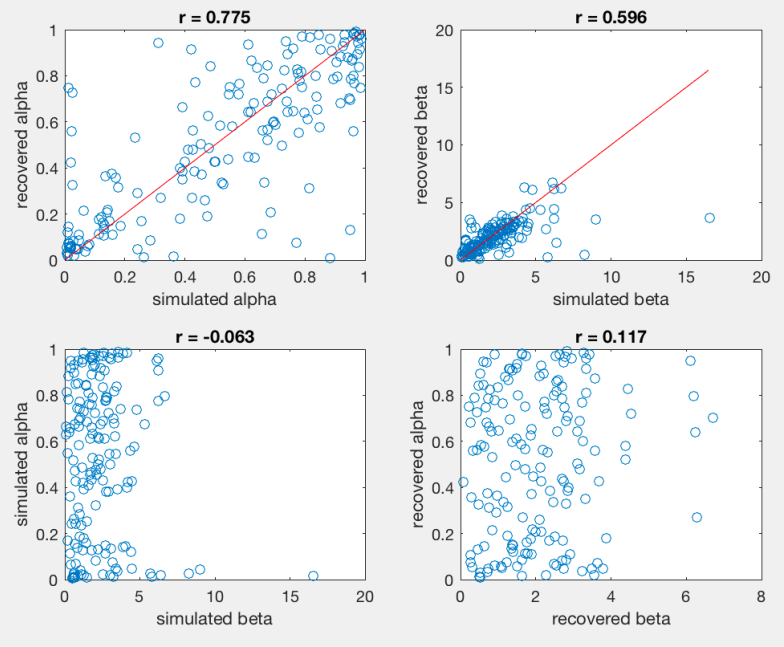
=> Unconventional: using the real parameters, not 1000 simulations with values drawn from distributions



Stable Low Noise



Stable High Noise

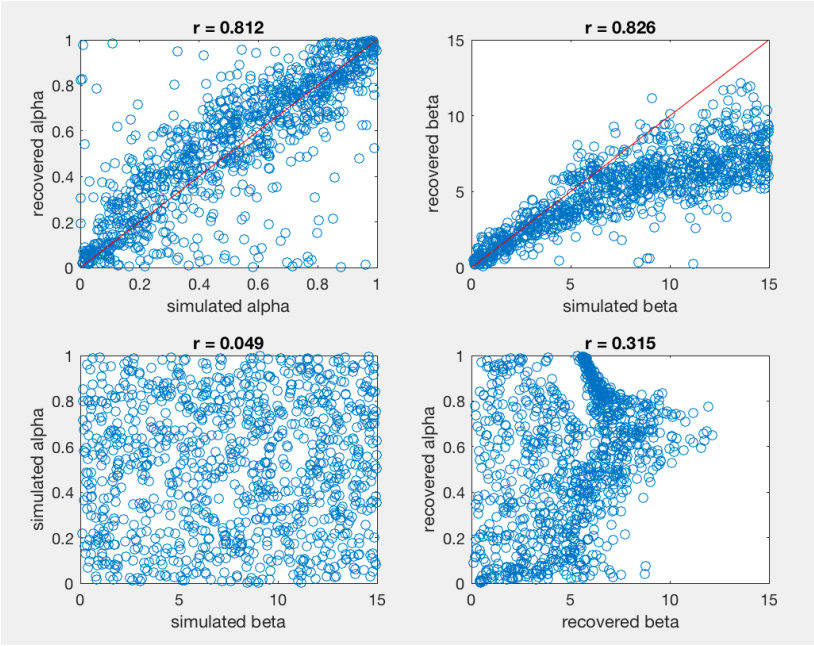


Volatile

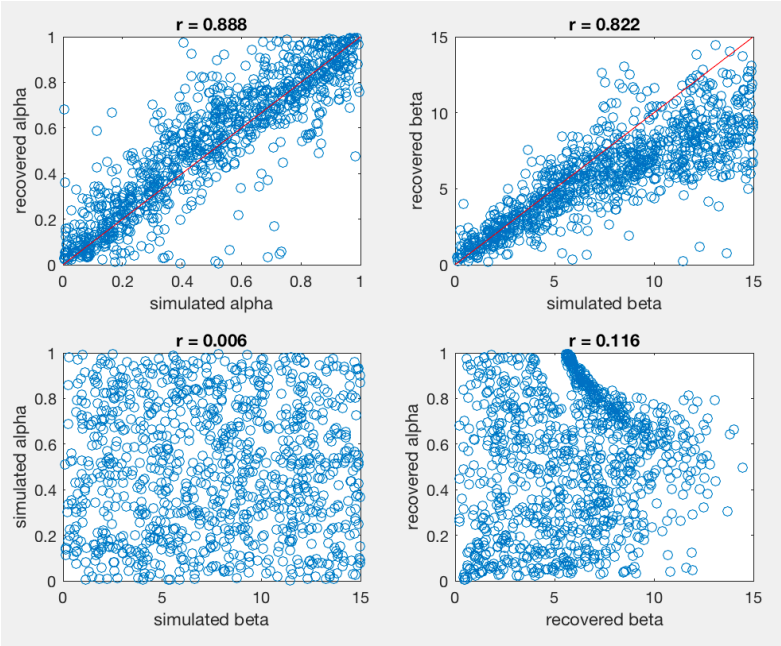
Goris 2020 three model fits recovery (MAP)

=> Parameter values from uniform distributions with range from empirical data

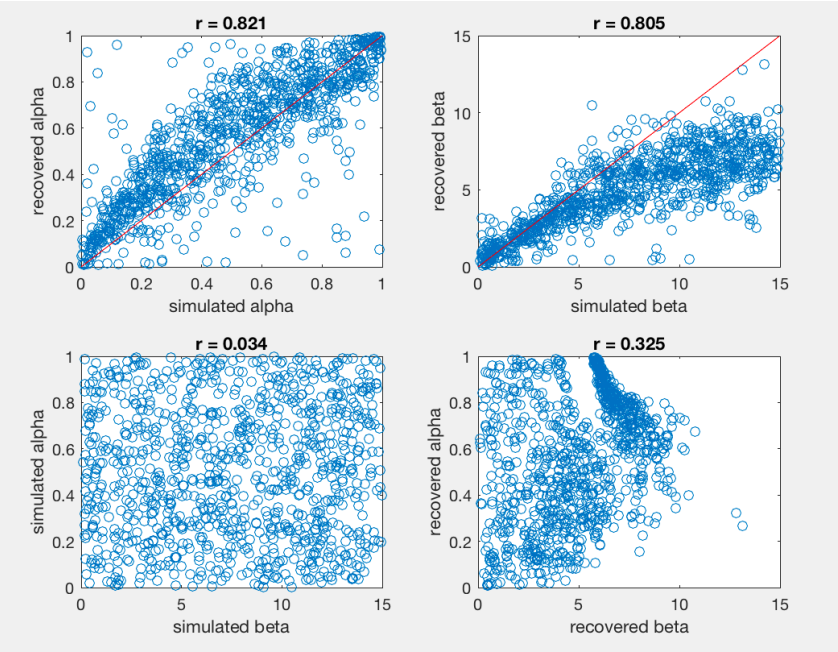
```
nsimus = 1000;
alpha_range = unifrnd(0,1,[nsimus,1]);
beta_range = unifrnd(0.1,15,[nsimus,1]);%min & max of the betas
```



Stable Low Noise



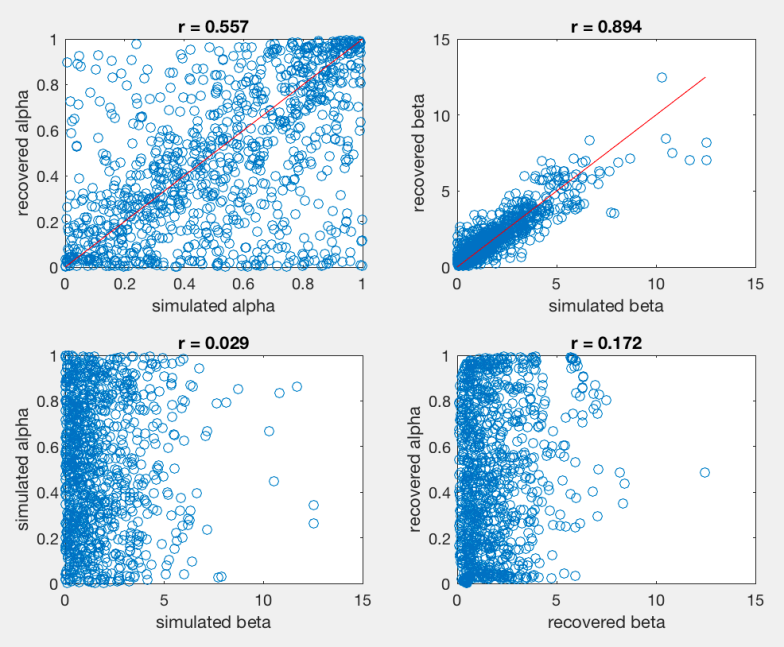
Stable High Noise



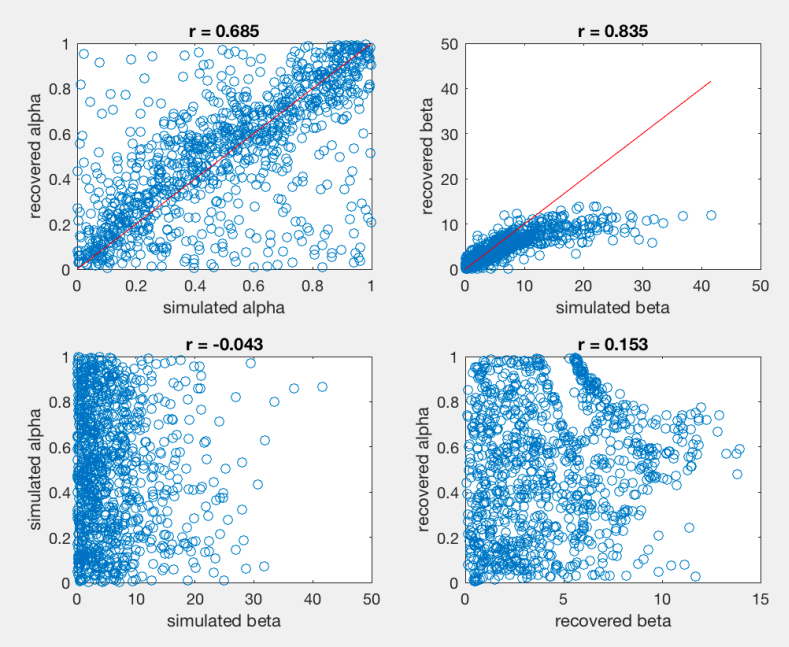
Volatile

Goris 2020 three model fits recovery (MAP)

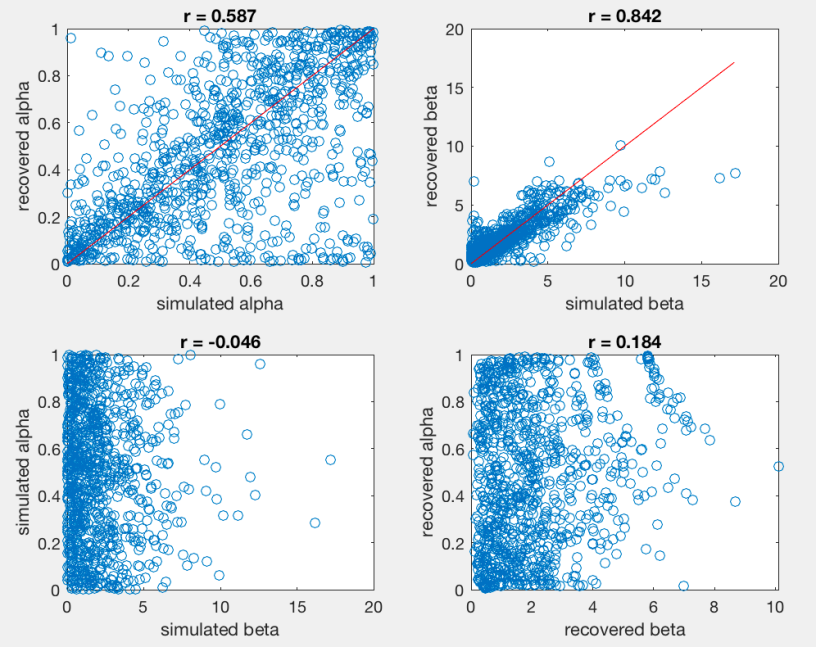
=> Learning rate from uniform distribution; beta from exponential distribution with mean from data



Stable Low Noise



Stable High Noise



Volatile

Other influences: β as regression weights

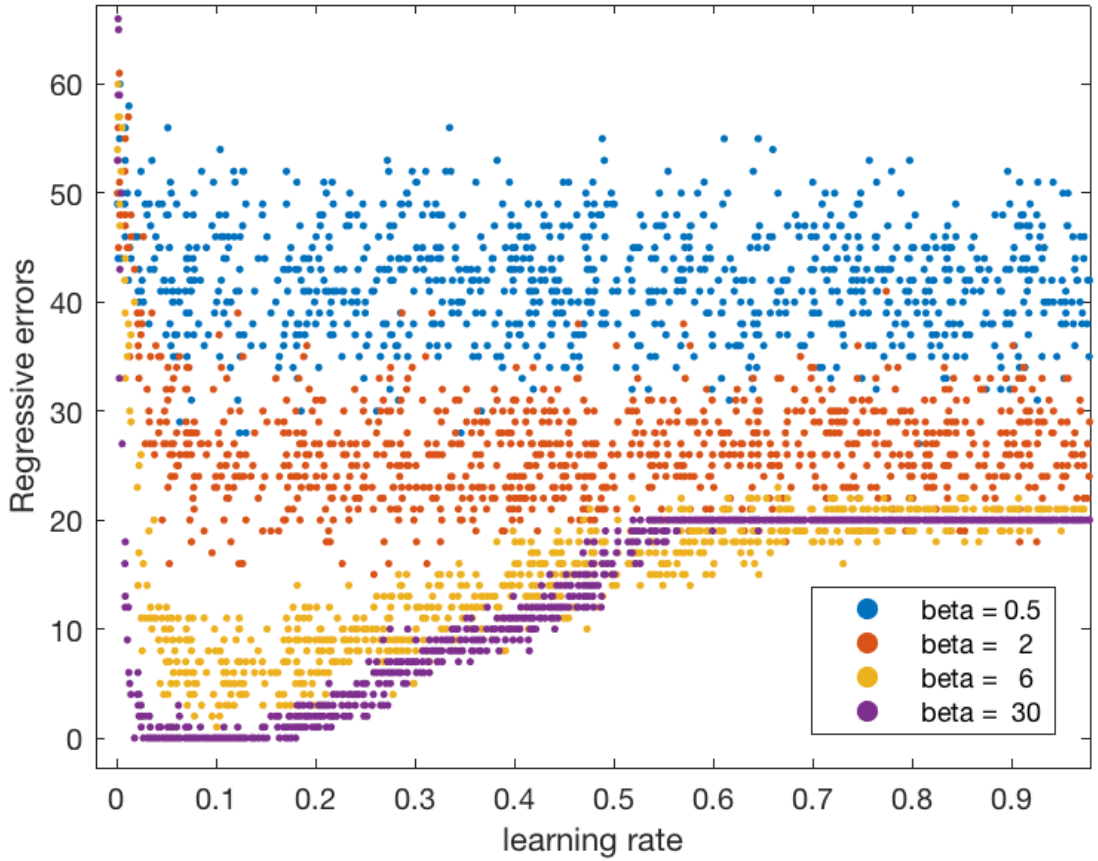
$$\mathrm{P(choose\ bandit\ k\ on\ trial\ t)} = \frac{e^{\beta v_t^k}}{\sum_{j=1}^{n} e^{\beta v_t^j}}$$

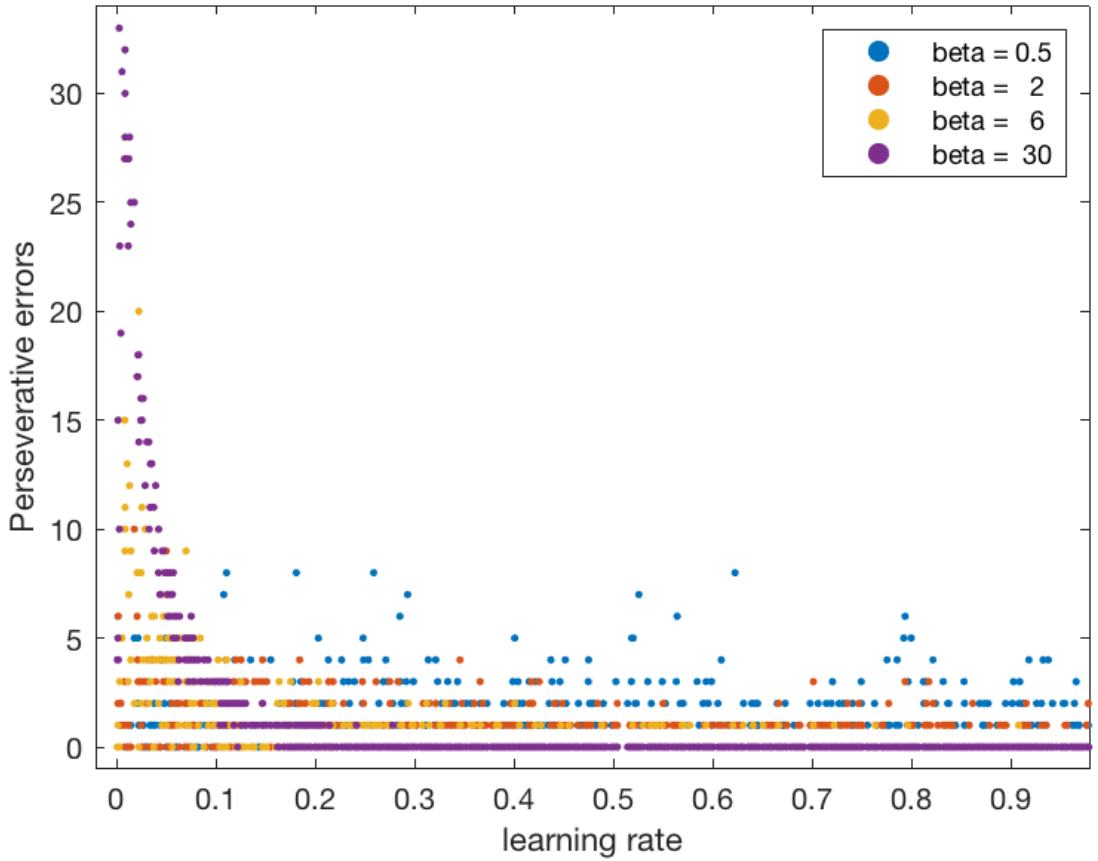
"Outcome Uncertainty" : $$T_i(t+1) = \begin{cases} 0, & \textit{if selected} = i \\ T_i(t) + 1, & \textit{otherwise} \end{cases}$$

Choice stickiness: $$S_i(t+1) = \begin{cases} 0, & \textit{otherwise} \\ 1, & \textit{if selected} = i \end{cases}$$

Modified from Aberg et a. (2021)

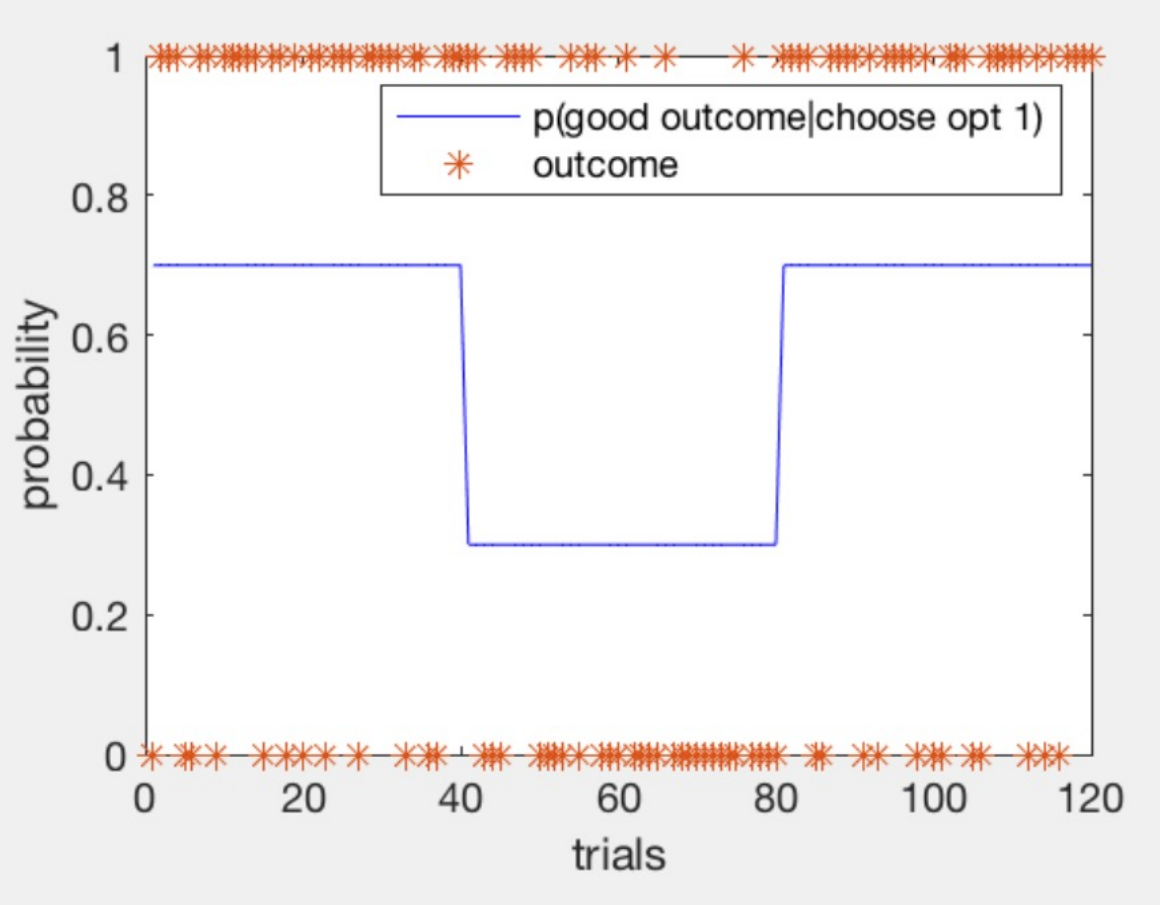
MATLAB: Probabilistic reversal learning task

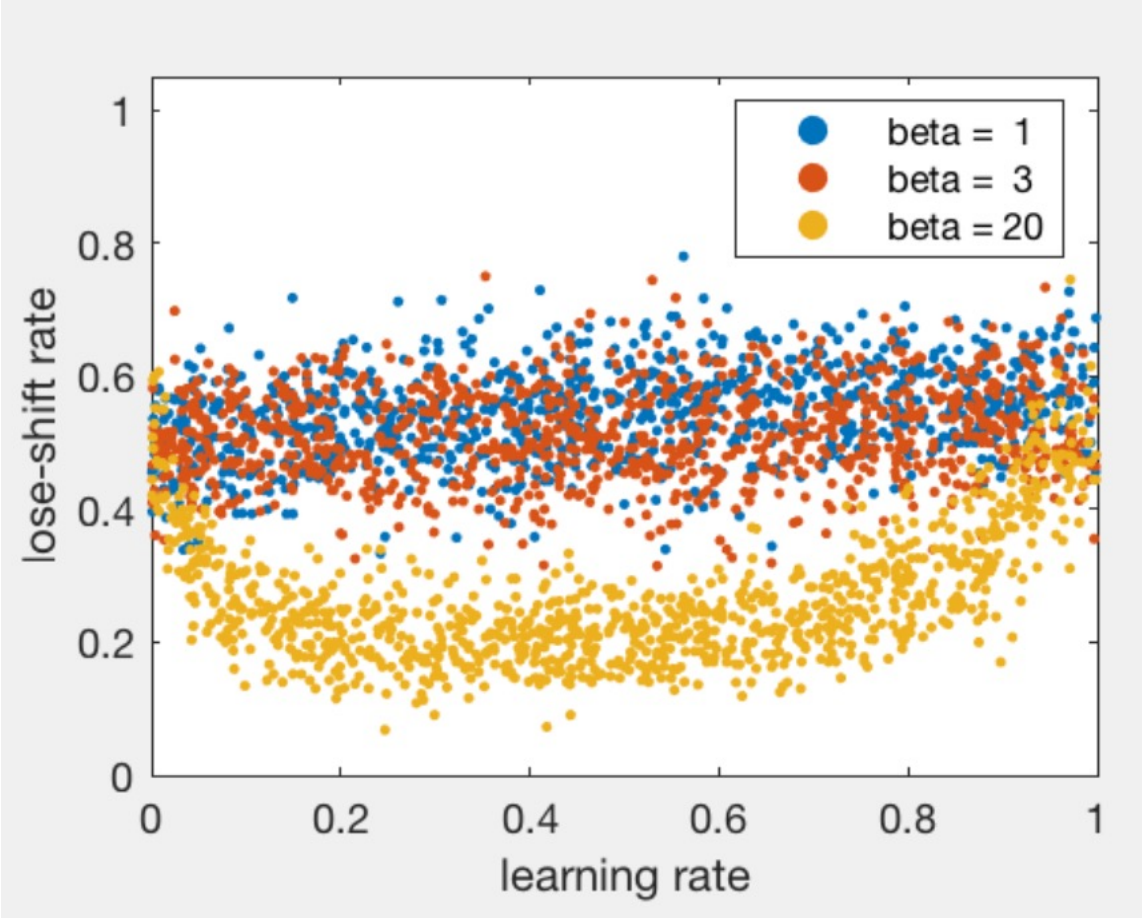





```
[perser,regres] = PRL_metric_calc(simulation,trial_when_reversal_occurred)
```

Lose-shift in no counterfactual updating?





50%+

0 as outcome => value becomes 0, same as the bad option··· (meanwhile, things may be different for a +1 and -1 schedule)

Kalman filter implementation

Piray & Daw 2021:

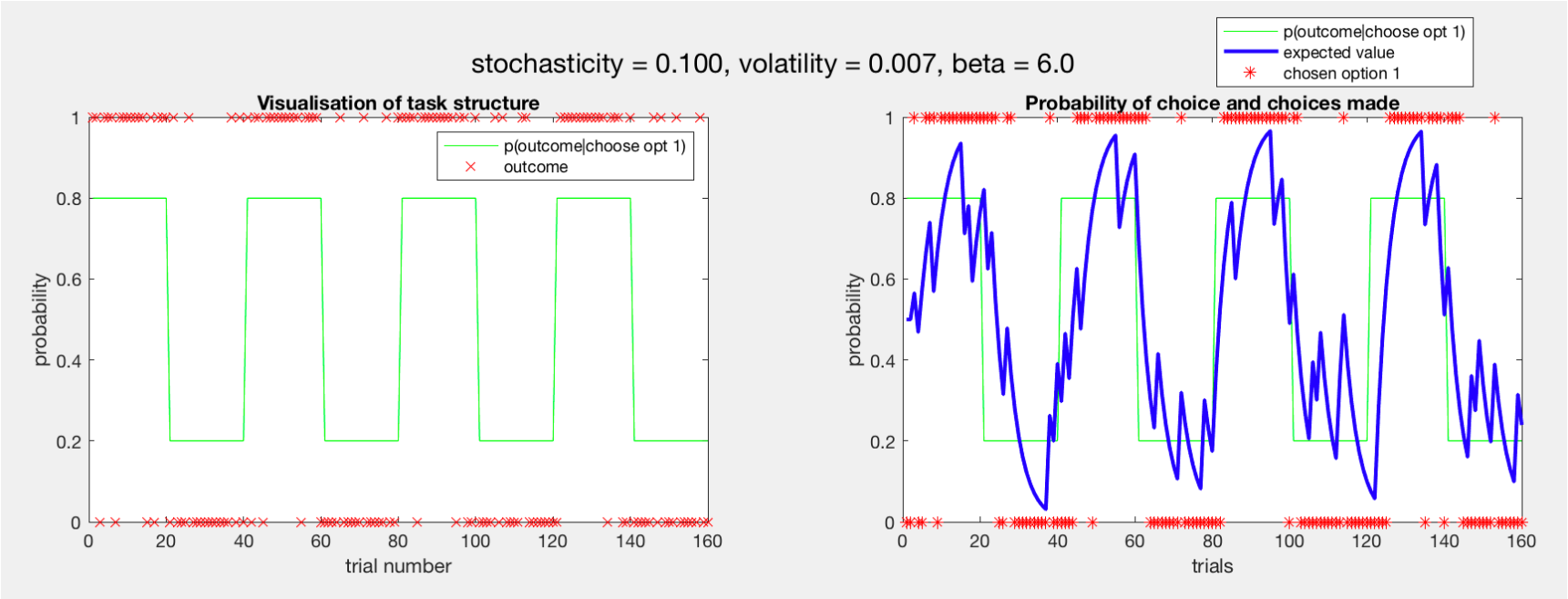
$$\delta_t = o_t - m_t$$

$$\alpha_t = \frac{w_t + v_t}{w_t + v_t + s_t}$$

$$m_{t+1} = m_t + \alpha_t \delta_t$$

$$w_{t+1} = (1 - \alpha_t)(w_t + v_t)$$

(But with fixed s and v)

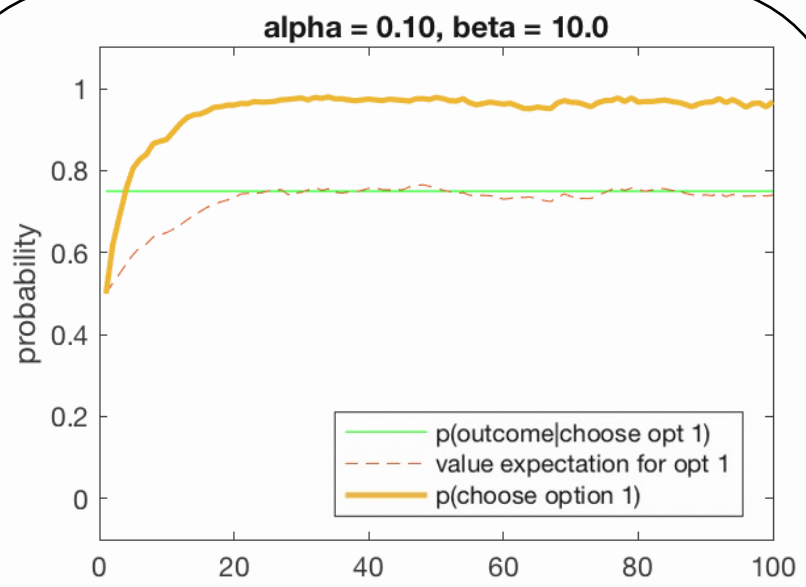


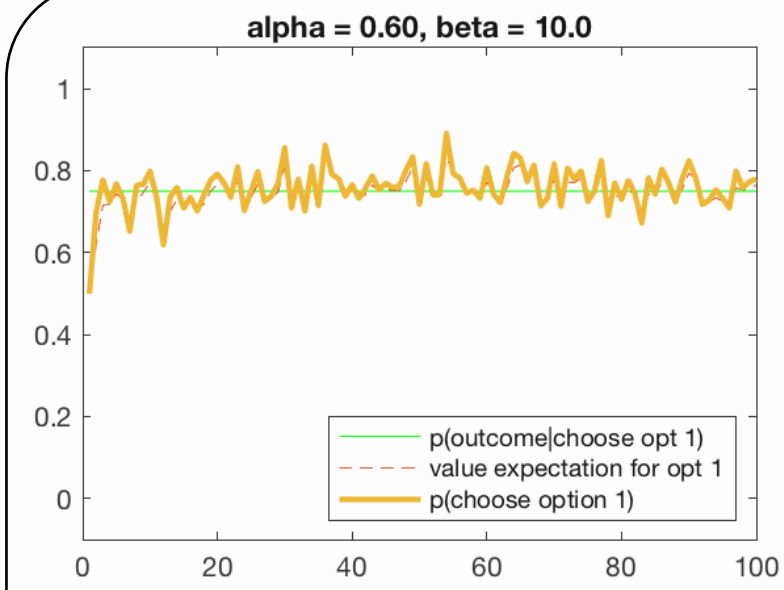
Stable

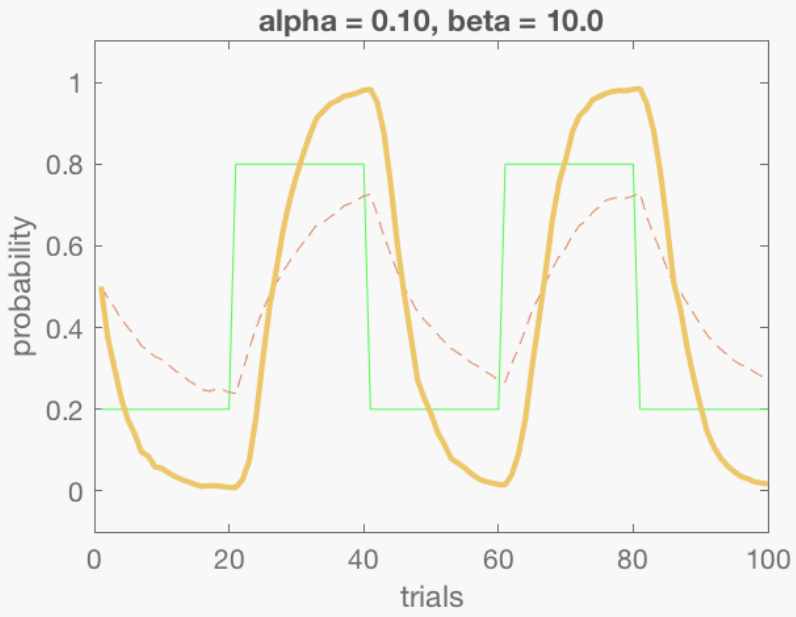
(One-armed bandit, binary outcomes)

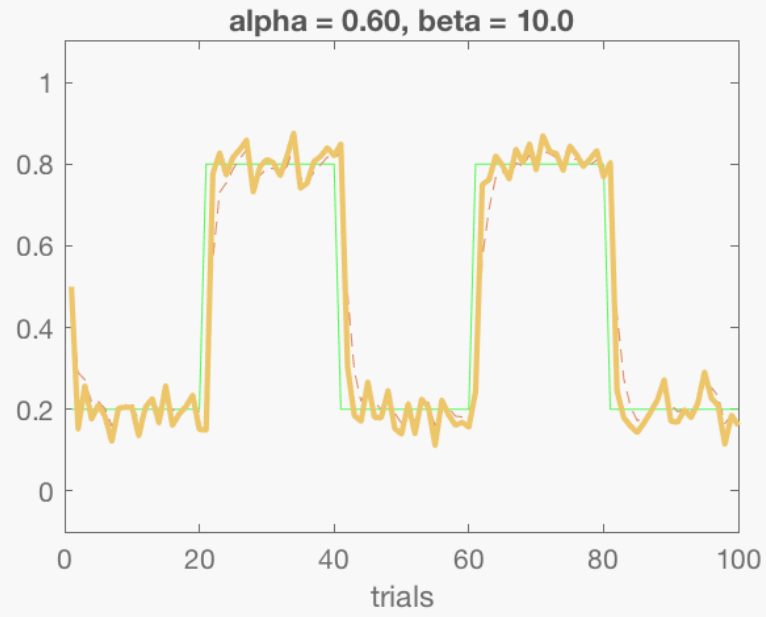
Volatile

(One-armed bandit, binary outcomes)









Low learning rate

High learning rate