RLDMM

2023-07-21

RLDMM

Looking at incorporating reinforcement learning! The basic idea being that the learning about the contingency space (U) influences the drift rate of the weiner distribution. So in an experimental setting this would amount to the participant having heard a cue and then having to predict the next stimulus. If the participant is very certain that the cue is associated with a particular stimulus then he/she will respond faster (higher absolute drift rate) compared to if very uncertain. The sign is also important so the drift rate is said to follow the following:

$$\delta_i = (E_i - (1 - E_i)) * \delta_s$$

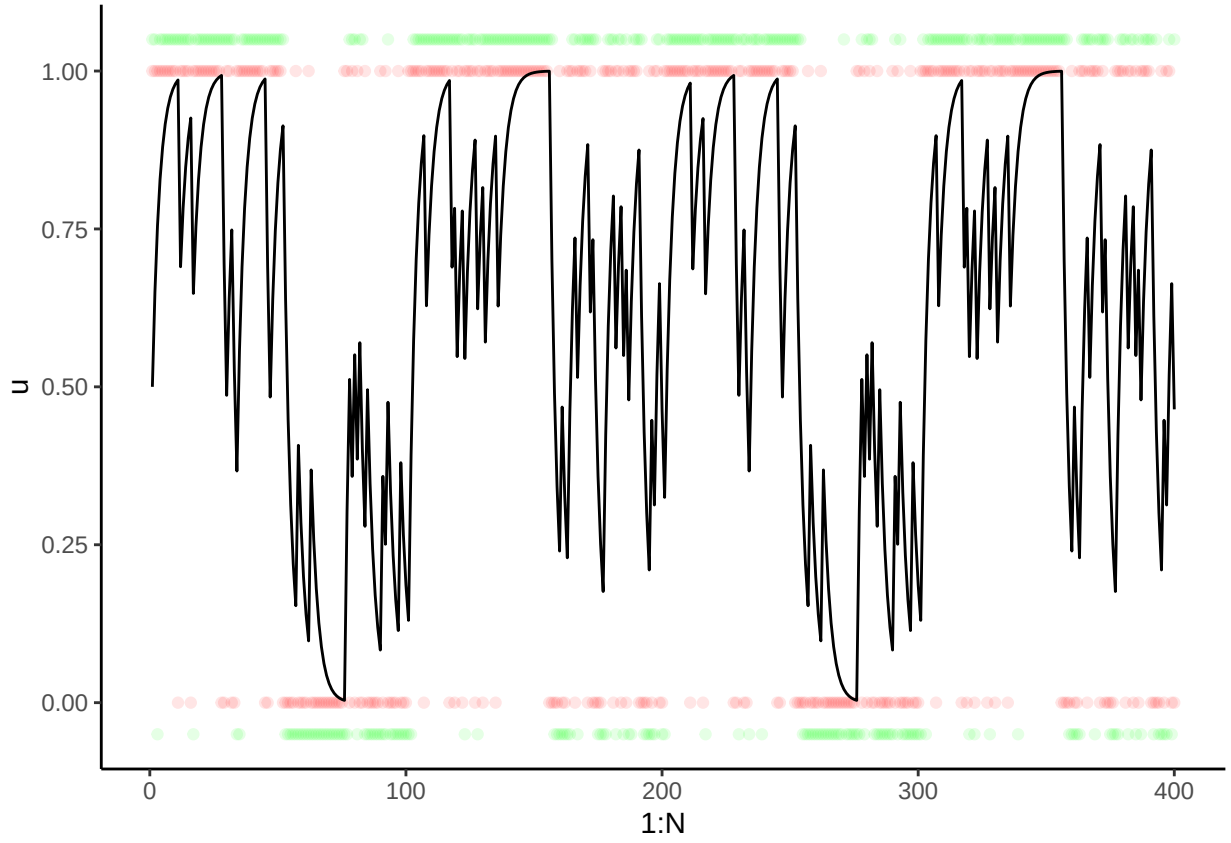
where δ_i is the drift rate on the given trial and δ_s is the subject specific parameter that determines how much the expectation influences the reaction times (drift rates). An example could be that the expectation is 0.2 we get $0.2 - (1 - 0.2) = 0.2 - 0.8 = -0.6$ or when the expectation is 0.8 we get $0.8 - (1 - 0.8) = 0.8 - 0.2 = 0.6$ The sign therefore makes sense: Lets simulate and see how it looks:

```
mod <- cmdstan_model(here::here("stan_scripts","RLHDMM_rng.stan"))
N = 400
u = rep(c(rbinom(50,1,0.8),rbinom(50,1,0.2),rbinom(50,1,0.8),rbinom(50,1,0.5)),N/200)

alpha = 4
delta = 2
beta = 0.5
tau = 0.1
lr = 0.3
e0 = 0.5
expectation = array(NA, N+1)
uncertainty = array(NA, N)
expectation[1] = e0

resp = data.frame()
for(i in 1:N){

  uncertainty[i] = (expectation[i]-(1-expectation[i]))*delta

  resp1 = rwiener(n = 1,
        alpha = alpha,
        delta = uncertainty[i],
        beta = beta,
        tau = tau)

  expectation[i+1] = expectation[i]+lr*(u[i]-expectation[i])

  resp = rbind(resp,resp1)
}
```

```
resp$u = u
resp$expectation = expectation[1:N]
resp$uncertainty = uncertainty[1:N]

resp$trial = 1:N
```

Lets now plot what this entails. The interesting part is how the behavior changes to see that the model makes sense! Here we then plot the expectation as a function of reaction times. However first we can plot how the expectation changes along with trials and inputs (U). Here the unputs are the red dots and the responses are the red dots and the black line is the the expectation.

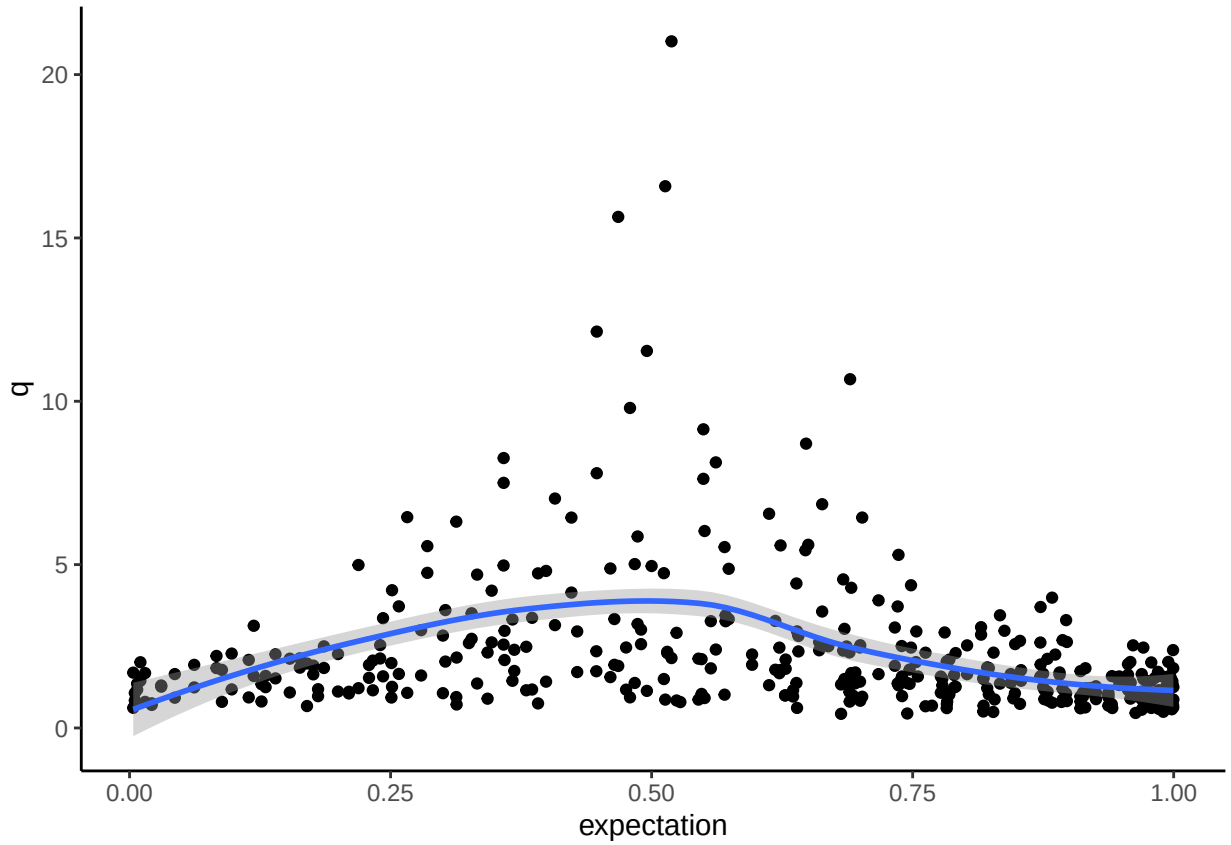
```
resp %>% mutate(resp = ifelse(resp == "upper",1.05,-0.05)) %>% ggplot()+
  geom_point(aes(x = 1:N, y = u),alpha = 0.1, col = "red")+
  geom_point(aes(x = 1:N, y = resp),alpha = 0.1, col = "green")+
  theme_classic()+
  geom_line(aes(x=1:N, y = expectation))
```



This looks nice the agent learns the contingency now for the expectation vs reaction time

```
resp %>% ggplot(aes(x = expectation, y = q))+
  geom_point()+
  theme_classic()+
  geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Given that the forward simulations look how we expect the question is if the model is inverseable! Lets use STAN again

```
resp = resp %>% mutate(resp2 = ifelse(resp == "upper",1,0))

mod = cmdstanr::cmdstan_model(here::here("stan_scripts","RLDDM.stan"))



data_stan = list(Nu = nrow(resp %>% filter(resp == "upper")),
                 Nl = nrow(resp %>% filter(resp == "lower")),
                 RTu = resp %>% filter(resp == "upper") %>% .$q,
                 RTl = resp %>% filter(resp == "lower") %>% .$q,
                 minRT = min(resp$q),
                 run_estimation = 1,
                 trials = nrow(resp),
                 u = resp$u,
                 indexupper = resp %>% filter(resp == "upper") %>% .$trial,
                 indexlower = resp %>% filter(resp == "lower") %>% .$trial,
                 resp = c(resp$resp2,0))




# fit1 <- mod$sample(
```

```
#     data = data_stan,
#     chains = 4,
#     parallel_chains = 4,
#     adapt_delta = 0.8,
#     max_treedepth = 10,
#     refresh = 10
#     )
#
#
# fit1$save_object(here::here("models","RLDDM_model.RDS"))

fit1 <- readRDS(here::here("models","RLDDM_model.RDS"))
```

Looks good

Lets look at the summary

```
flextable::flextable(data.frame(fit1$summary()) %>% mutate_if(is.numeric, round,2) %>% head(10))
```

```
## Warning: fonts used in ‘flextable‘ are ignored because the ‘pdflatex‘ engine is
## used and not ‘xelatex‘ or ‘lualatex‘. You can avoid this warning by using the
## ‘set_flextable_defaults(fonts_ignore=TRUE)‘ command or use a compatible engine
## by defining ‘latex_engine: xelatex‘ in the YAML header of the R Markdown
## document.
```

variable	mean	median	sd	mad	q5	q95	rhat	ess_bulk	ess_tail
lp___	-143,657.21	-143,657.00	1.58	1.48	-143,660.00	-143,655.00	1	1,910.90	
alpha	3.29	3.29	0.01	0.01	3.28	3.30	1	3,094.94	3,018.39
beta	0.53	0.53	0.00	0.00	0.53	0.53	1	5,139.66	2,891.65
delta	1.03	1.03	0.00	0.00	1.02	1.03	1	2,911.70	2,814.56
tau_raw	-0.27	-0.27	0.03	0.03	-0.32	-0.23	1	2,839.19	2,554.49
lr	0.42	0.42	0.00	0.00	0.41	0.42	1	2,538.24	2,677.29
expect[1]	0.50	0.50	0.00	0.00	0.50	0.50			
expect[2]	0.71	0.71	0.00	0.00	0.71	0.71	1	2,538.16	2,677.29
expect[3]	0.83	0.83	0.00	0.00	0.83	0.83	1	2,538.27	2,677.29
expect[4]	0.90	0.90	0.00	0.00	0.90	0.90	1	2,538.09	2,677.29

Predictive checks?

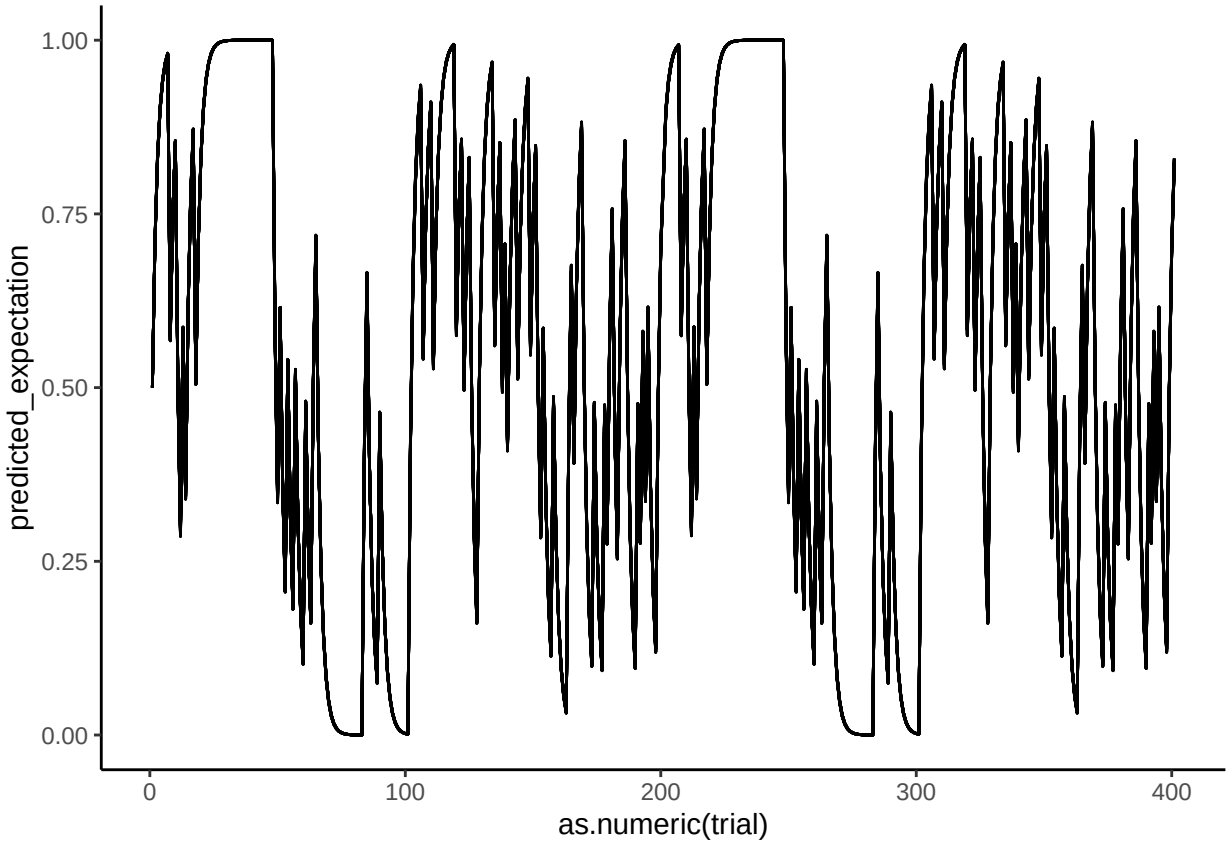
```
library(posterior)

n_check = 20

rts = as_draws_df(fit1$draws()) %>% select(matches("out\\[\\d+,1\\]")) %>% slice(1:n_check) %>% mutate(
  mutate(trial = gsub(".*\\[(\\d+).*\\]", "\\1", trial))
```

```
## Warning: Dropping 'draws_df' class as required metadata was removed.
```

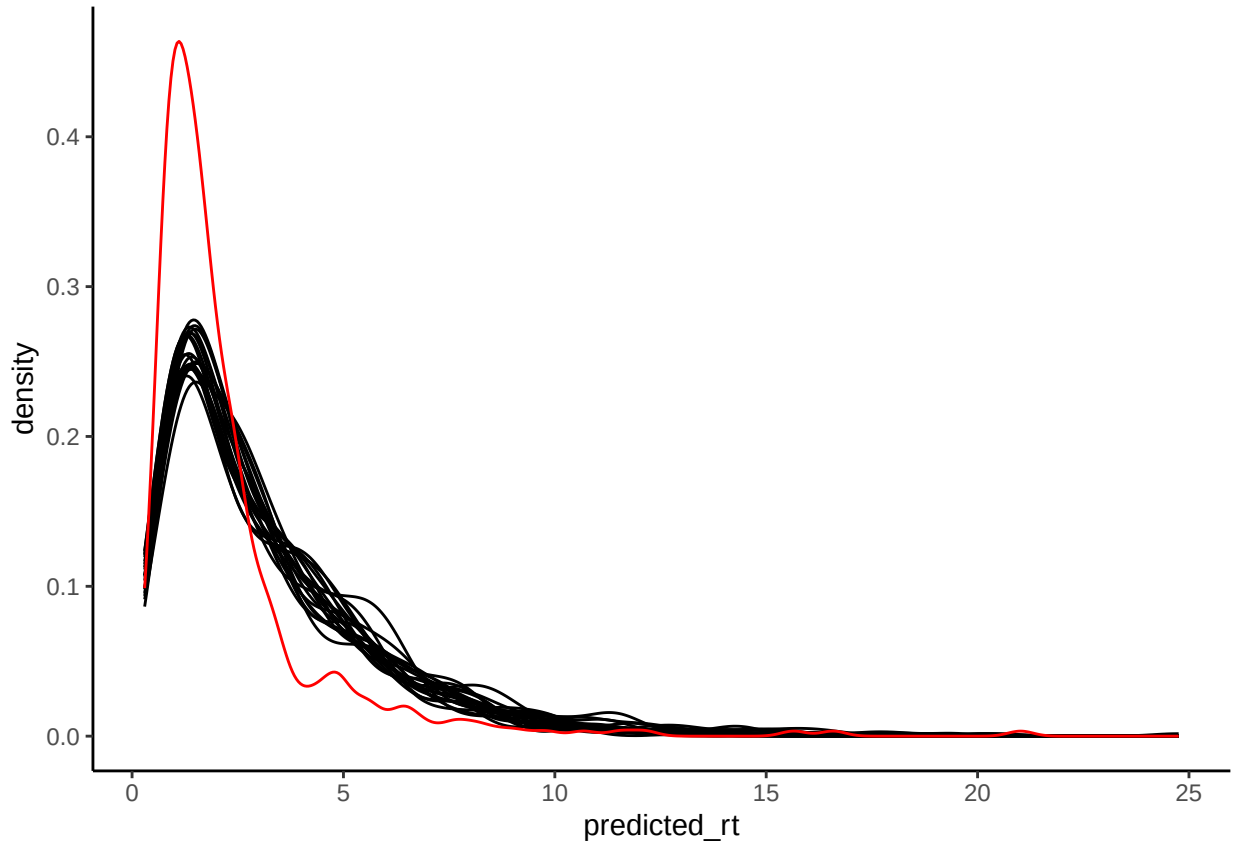
```
expect = as_draws_df(fit1$draws()) %>% select(matches("expect\\[\\d+\\]")) %>% slice(1:n_check) %>% muta
  mutate(trial = gsub(".*\\[(\\d+).*\\]", "\\1", trial))
```

```
## Warning: Dropping 'draws_df' class as required metadata was removed.
```

```
expect %>% ggplot()+geom_line(aes(x = as.numeric(trial), y = predicted_expectation, group = as.factor(d
```




```
rts %>% ggplot()+geom_density(aes(x = predicted_rt, group = draw))+geom_density(data = resp, aes(x = q)
```




```
rts = as_draws_df(fit1$draws()) %>% select(matches("out\\[\\d+,1\\]")) %>% mutate(draw = 1:4000) %>% pi
  mutate(trial = gsub(".*\\[(\\d+).*\\]", "\\1", trial))
```

```
## Warning: Dropping 'draws_df' class as required metadata was removed.
```

```
expect = as_draws_df(fit1$draws()) %>% select(matches("expect\\[\\d+\\]")) %>% mutate(draw = 1:4000) %>
  mutate(trial = gsub(".*\\[(\\d+).*\\]", "\\1", trial))
```

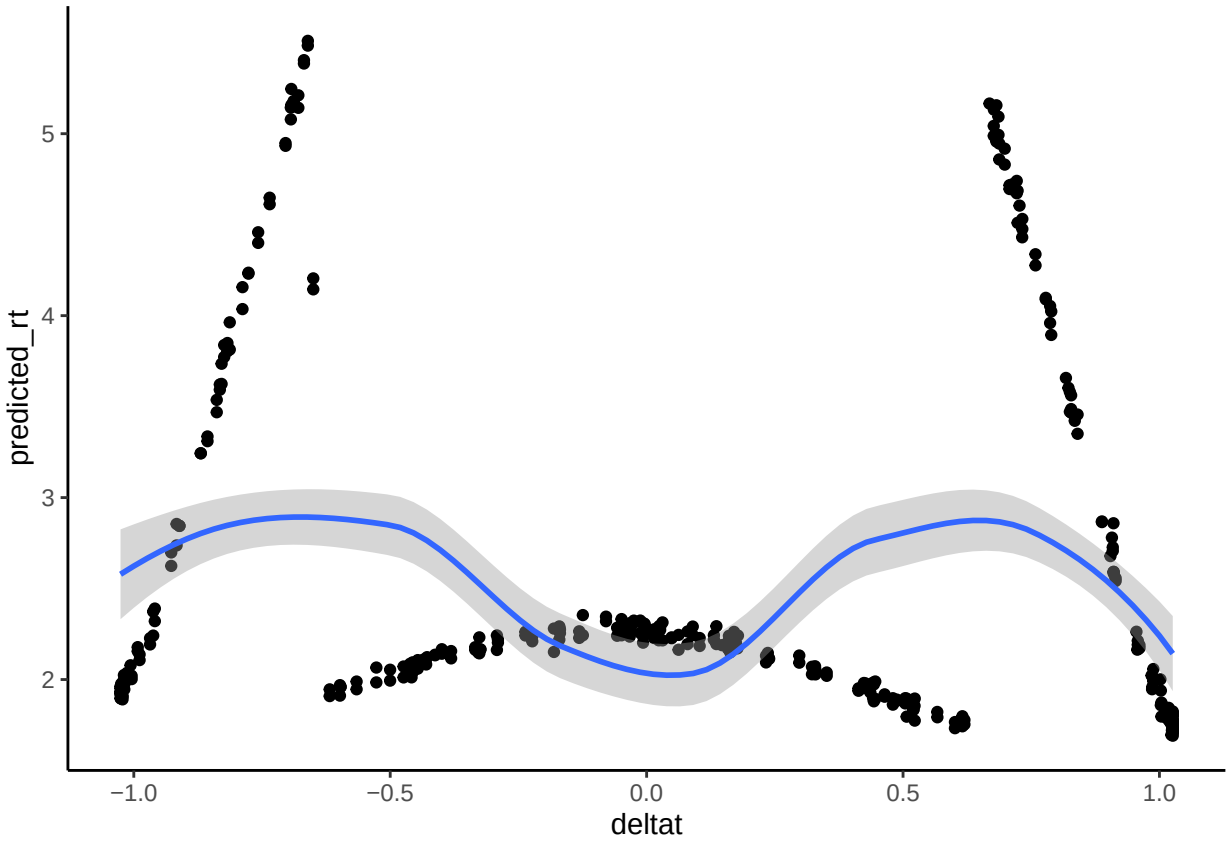
```
## Warning: Dropping 'draws_df' class as required metadata was removed.
```

```
deltat = as_draws_df(fit1$draws()) %>% select(matches("deltat\\[\\d+\\]")) %>% mutate(draw = 1:4000) %>
  mutate(trial = gsub(".*\\[(\\d+).*\\]", "\\1", trial))
```

```
## Warning: Dropping 'draws_df' class as required metadata was removed.
```

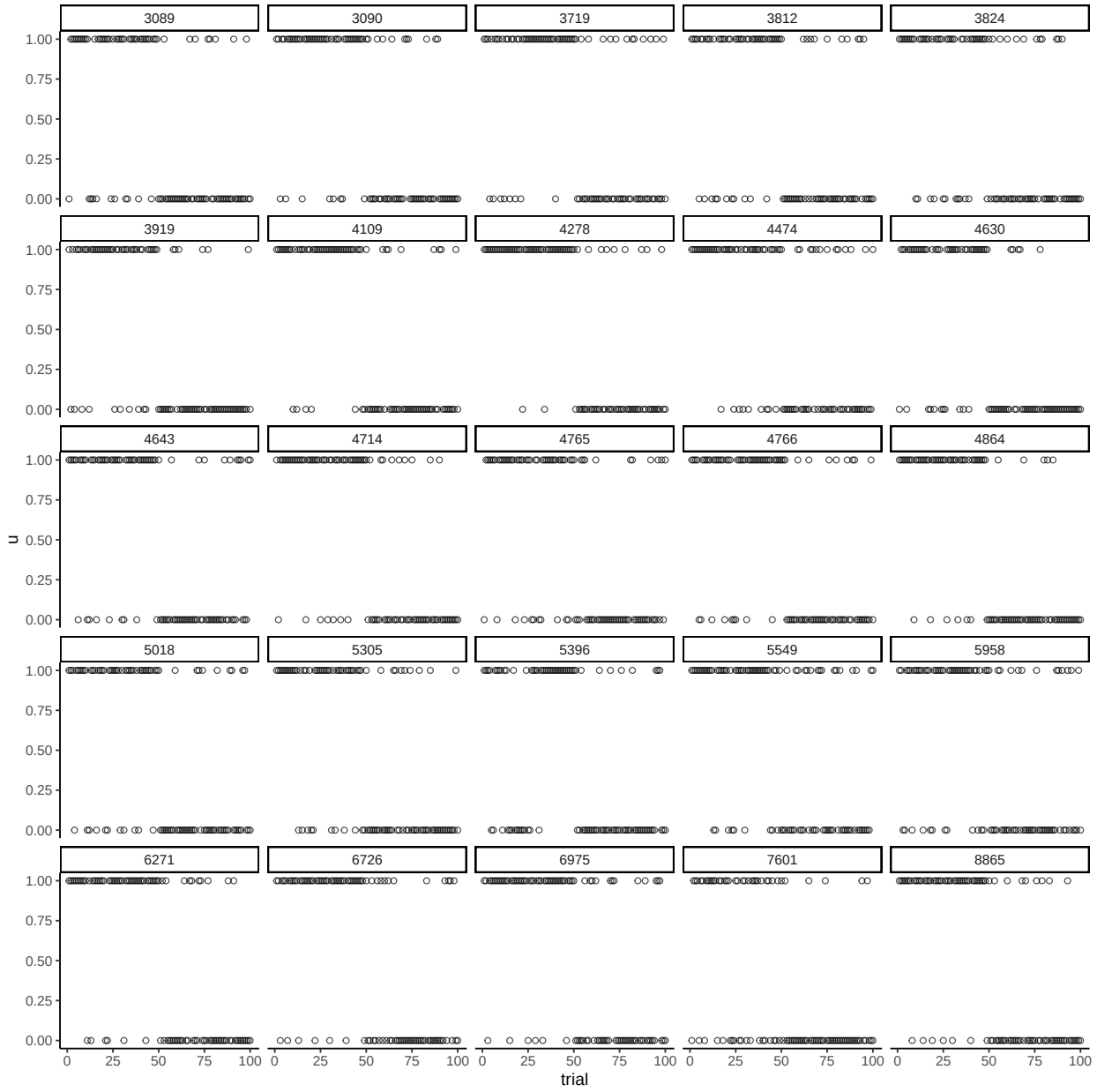
```
inner_join(deltat,rts) %>% group_by(trial) %>%
  summarize(sd_expect = sd(deltat),
            deltat = median(deltat),
            sd_rt = sd(predicted_rt),
            predicted_rt = median(predicted_rt)) %>%
  ggplot(aes(x = deltat, y = predicted_rt)) +geom_point()+geom_smooth()+theme_classic()+
  scale_x_continuous(breaks = seq(-2,2,by = 0.5))
```

```
## Joining with `by = join_by(draw, trial)`
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Before moving to incorporating more things one crucial point for all learning experiments is the input sequence. Which is down to the individual trials and not just how the underlying probability varies through the task. Lets demonstrate the difference in a reversal paradigm with 100 trials of 50 trials with 80% probability and 50 trials of 20% probability

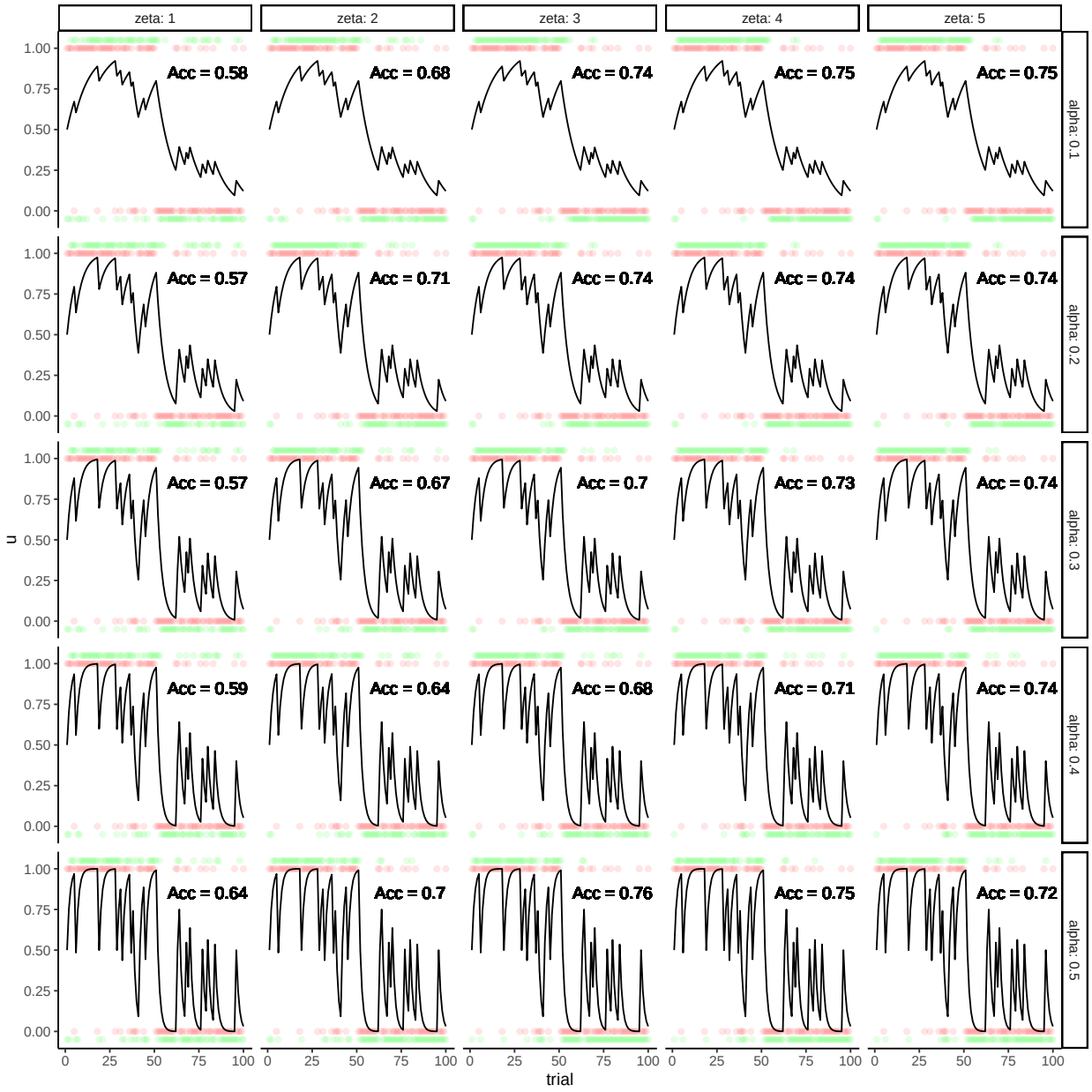
```
get_con = function(seed){
  set.seed(seed)
  u = c(rbinom(50,1,0.8),rbinom(50,1,0.2))
  trial = 1:100
  return(list(data.frame(u = u, trial = trial, seed = seed)))
}
seeds = as.list(round(rnorm(25,5000,1000),0))

results <- future_map(seeds, ~get_con(.x), .progress = TRUE, .options = furrr_options(seed = TRUE))

map_dfr(results,1) %>% ggplot(aes(x = trial, y = u))+
  geom_point(shape = 21, alpha = 0.7)+
  facet_wrap(~seed, nrow = 5, ncol = 5)+
  theme_classic()
```



We see a clear difference, but how does this translate to the parameters of a learning paradigm. Lets start by looking at the simplest case a rescorla wagner learning model with a constant learning rate α and a decision temperature parameter ζ that will govern the consistency in responses (explore / exploit). Lets start with a single seed with differing learning rates and decision temperatures:

```
get_rw = function(parameters){
  alpha = parameters$alpha
  zeta = parameters$zeta
  seed = parameters$seed

  data = get_con(seed)[[1]]
  u = data$u
  expectation = array(NA,nrow(data)+1)
  resp = array(NA, nrow(data))
```

```
  expectation[1] = 0.5
  for(i in 1:nrow(data)){
    resp[i] = rbinom(1,1,(expectation[i]^zeta)/((expectation[i]^zeta)+(1-expectation[i])^zeta))
    expectation[i+1] = expectation[i]+alpha*(u[i]-expectation[i])

  }
  data$expectation = expectation[1:100]
  data$resp = resp
  data$id = parameters$id
  data$alpha = parameters$alpha
  data$zeta = parameters$zeta
  data$seed = parameters$seed
  data$pcorrect = sum(data$u == data$resp)/nrow(data)

  return(list(data))
}

seed = 123
alpha = seq(0.1,0.5, by = 0.1)
zeta = seq(1,5, by = 1)

parameters = expand.grid(zeta = zeta,
                         alpha = alpha,
                         seed = seed) %>%
  mutate(id = 1:nrow(.))

data_list <- split(parameters, parameters$id)

results <- future_map(data_list, ~get_rw(.x), .progress = TRUE, .options = furrr_options(seed = TRUE))

map_dfr(results,1) %>% mutate(resp = ifelse(resp == 1, 1.05,-0.05)) %>%
  ggplot()+
  geom_point(aes(x = trial, y = u),alpha = 0.1, col = "red")+
  geom_point(aes(x = trial, y = resp),alpha = 0.1, col = "green")+
  geom_line(aes(x = trial, y = expectation))+
  facet_grid(alpha~zeta, labeller = label_both)+
  geom_text(aes(x = 80, y = 0.85, label = paste0("Acc = ", pcorrect)), size = 4)+
  theme_classic()
```

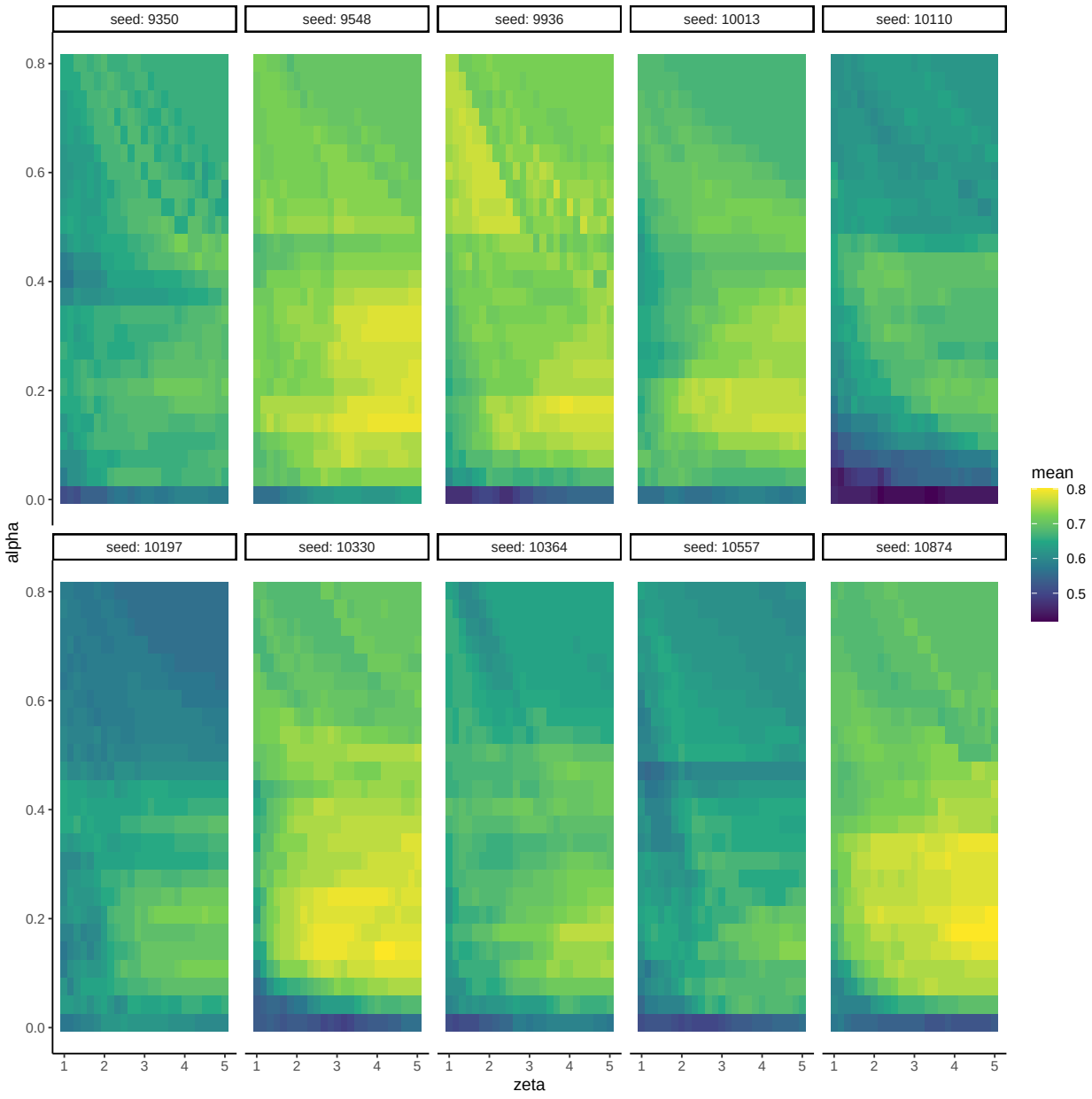


So to get an idea of how the seed will influence the what parameters will lead to optimal behavior we can take differing learning rates, decision temperatures and seeds and plot these as a function of accuracy.

```
seed = round(rnorm(10,10000,500))
alpha = seq(0.01,0.8, length.out = 25)
zeta = seq(1,5, length.out = 25)

parameters = expand.grid(zeta = zeta,
                         alpha = alpha,
                         seed = seed) %>%
  mutate(id = 1:nrow(.))

data_list <- split(parameters, parameters$id)

results <- future_map(data_list, ~get_rw(.x), .progress = TRUE, .options = furrr_options(seed = TRUE))
```

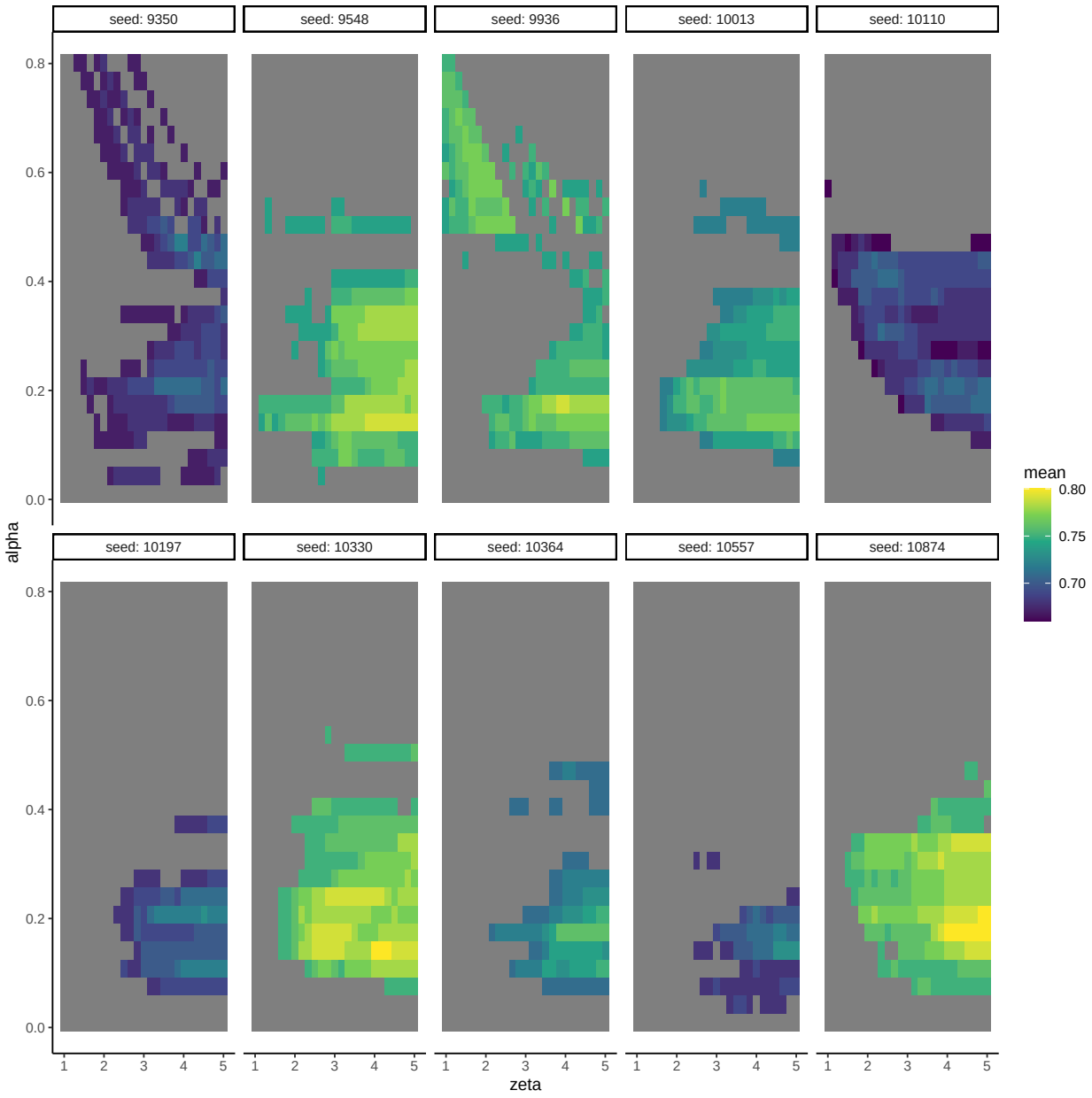
```
max = map_dfr(results,1) %>% group_by(seed) %>% summarize(max = max(pcorrect))


map_dfr(results,1) %>% group_by(id,alpha,zeta,seed) %>%
  summarize(mean = mean(pcorrect)) %>%
  ggplot(aes(y = alpha, x = zeta, fill = mean))+
  geom_raster()+
  facet_wrap(~seed, ncol = 5, nrow = 5, labeller = label_both)+
  scale_fill_continuous(type = "viridis",)+
  theme_classic()
```

```
## `summarise()` has grouped output by 'id', 'alpha', 'zeta'. You can override
## using the `.groups` argument.
```



```
map_dfr(results,1) %>% group_by(id,alpha,zeta,seed) %>%
  summarize(mean = mean(pcorrect)) %>% inner_join(.,max) %>% mutate(mean = ifelse(mean >= max-0.05, mean
  ggplot(aes(y = alpha, x = zeta, fill = mean))+
  geom_raster()+
  facet_wrap(~seed, ncol = 5, nrow = 5, labeller = label_both)+
  scale_fill_continuous(type = "viridis",)+
  theme_classic()
```

```
## `summarise()` has grouped output by 'id', 'alpha', 'zeta'. You can override
## using the `.groups` argument.
## Joining with `by = join_by(seed)`
```



How about other learning models: Kalman filtering, hierarchical gaussian filtering or volatile kalman filtering?

```
vkf_agent = function(parameters){
  seed = parameters$seed
  set.seed(seed)

  trials_per_reversal = parameters$trials_per_reversal
  #n_reversals = parameters$n_reversals

  u = c(rbinom(50,1,0.8),
        rbinom(trials_per_reversal,1,0.2),
        rbinom(trials_per_reversal,1,0.8),
        rbinom(trials_per_reversal,1,0.2),
        rbinom(trials_per_reversal,1,0.8),
        rbinom(trials_per_reversal,1,0.2),
        rbinom(trials_per_reversal,1,0.8),
        rbinom(50,1,0.2))

  N = length(u)

  omega = parameters$omega
  v0 = parameters$v0
  w0 = parameters$w0
  m0 = parameters$m0

  lambda = parameters$lambda

  zeta = parameters$zeta


  k = array(NA, N+1)
  a = array(NA, N+1)
  m = array(NA, N+1)
  v = array(NA, N+1)
  w = array(NA, N+1)
  w2 = array(NA, N+1)
  real_resp = array(NA, N+1)

  w[1] = w0
  v[1] = v0
  m[1] = m0

  s = function(x){
    return(1/(1+exp(-x)))
  }

  agg = tryCatch({
      for(i in 1:N){

      k[i+1] = (w[i]+v[i])/(w[i]+v[i]+omega)
      a[i+1] = sqrt((w[i]+v[i]))
      m[i+1] = m[i]+a[i+1]*(u[i]-s(m[i]))

      w2[i] = (1-k[i+1])*(w[i])
```

```
      w[i+1] = (1-k[i+1])*(w[i]+v[i])

      v[i+1] = v[i]+lambda*((m[i+1]-m[i])^2+w[i]+w[i+1]-2*w2[i]-v[i])

      real_resp[i] = rbinom(1,1,(s(m[i])^zeta)/((s(m[i])^zeta)+(1-s(m[i]))^zeta))

      }
      resp = data.frame(k = k,a = a, m = m, w2 = w2, w = w, v = v, resp = real_resp)
      resp$u = c(u,NA)
      resp$trial = 1:(N+1)
      resp = resp %>% mutate(correct = ifelse(real_resp == u, 1, 0))
  #

      resp = resp[1:N,]
      agg = data.frame(pcorrect =  sum(resp$correct)/N,
                       correct = sum(resp$correct),
                       omega = omega,
                       lambda = lambda,
                       zeta = zeta,
                       trials = N,
                       seed = seed,
#                       n_r = n_reversals,
                       t_p_r = trials_per_reversal)


      }, warning=function(w) {
          ## do something about the warning, maybe return 'NA'
          return(data.frame(pcorrect = NA,
                       correct = NA,
                       omega = omega,
                       lambda = lambda,
                       zeta = zeta,
                       seed = seed,
                       trials = N,
#                       n_r = n_reversals,
                       t_p_r = trials_per_reversal))
  })

return(list(agg))
}
```

```
vkf_agent(parameters = data.frame(n_reversals = 5,
                                  trials_per_reversal = 20,
                                  omega = 1,
                                  v0 = 0.1,
                                  seed = 123,
                                  w0 = 0.1,
                                  m0 = 0.5,
                                  lambda = 1,
                                  zeta = 5,
                                  id = 2))
```

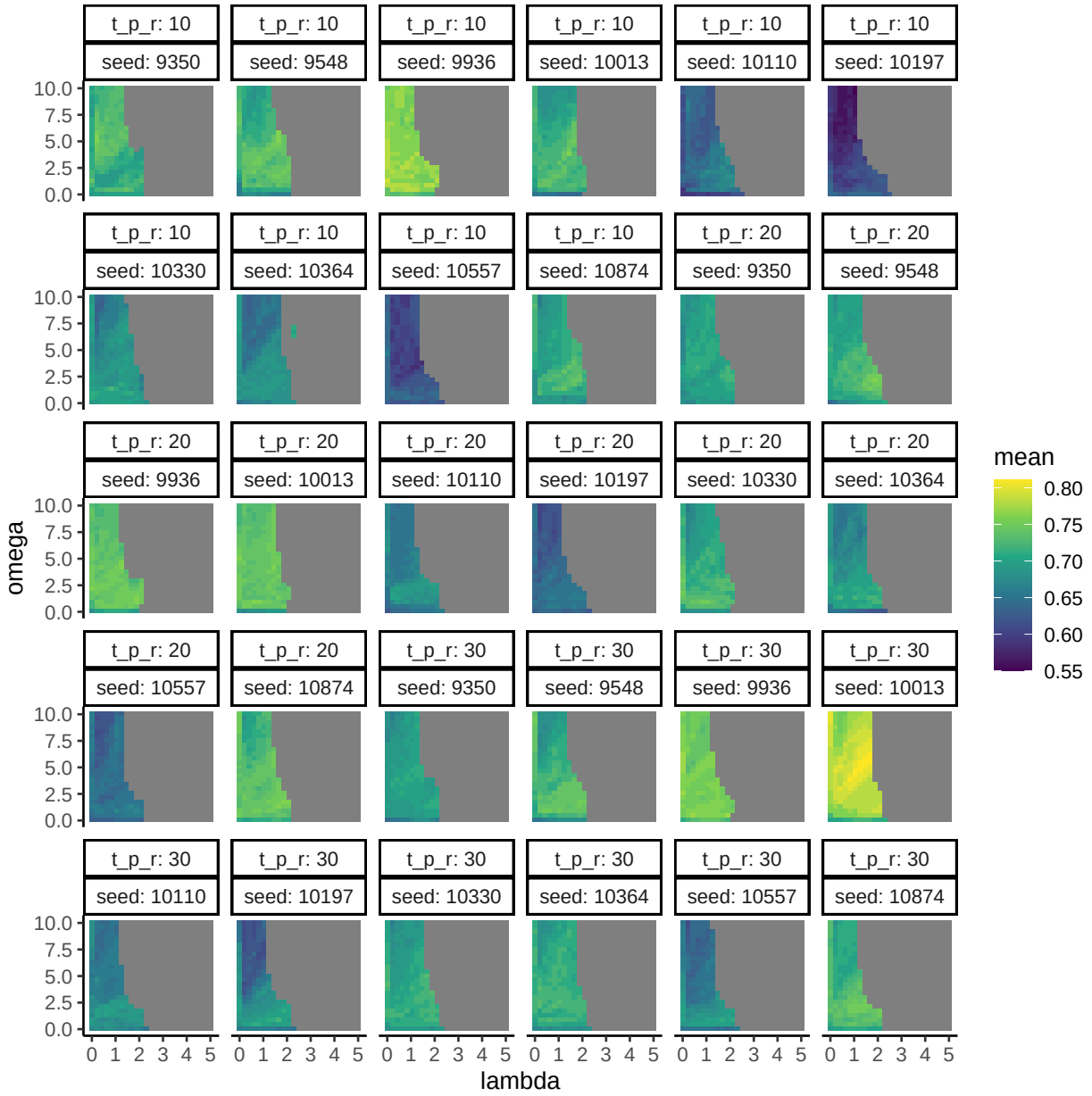
```
## [[1]]
```

```
##    pcorrect correct omega lambda zeta trials seed t_p_r
## 1 0.7181818     158     1      1    5    220  123    20
```

```
#n_reversals = seq(5,20,length.out = 4)
#n_reversals = seq(5,length.out = 1)

trials_per_reversal = seq(10,30, by = 10)
#trials_per_reversal = seq(20, length.out = 1)


omega = seq(0.1,10, length.out = 25)
lambda = seq(0.01,5, length.out = 25)


v0 = seq(0.1, length.out = 1)

m0 = seq(0.5, length.out = 1)

w0 = seq(0.1, length.out = 1)

zeta = seq(5,5, by = 1)

replicate = 1

parameters = expand.grid(omega= omega,
                         lambda = lambda,
#                          n_reversals = n_reversals,
                         v0 = v0,
                         w0 = w0,
                         m0 = m0,
                         seed = seed,
                         zeta = zeta,
                         replicate = replicate,
                         trials_per_reversal = trials_per_reversal) %>%
  mutate(id = 1:nrow(.))

data_list <- split(parameters, parameters$id)


cores = availableCores()-1

plan(multisession, workers = 5)

results <- future_map(data_list, ~vkf_agent(.x), .progress = TRUE, .options = furrr_options(seed = TRUE

params = map_dfr(results, 1)

params2 = na.omit(params)


## seed analysis

optimals = params2[params2$pcorrect == max(params2$pcorrect, na.rm = T) | params2$pcorrect > max(params
optimal = params2[params2$pcorrect == max(params2$pcorrect, na.rm = T),]
```

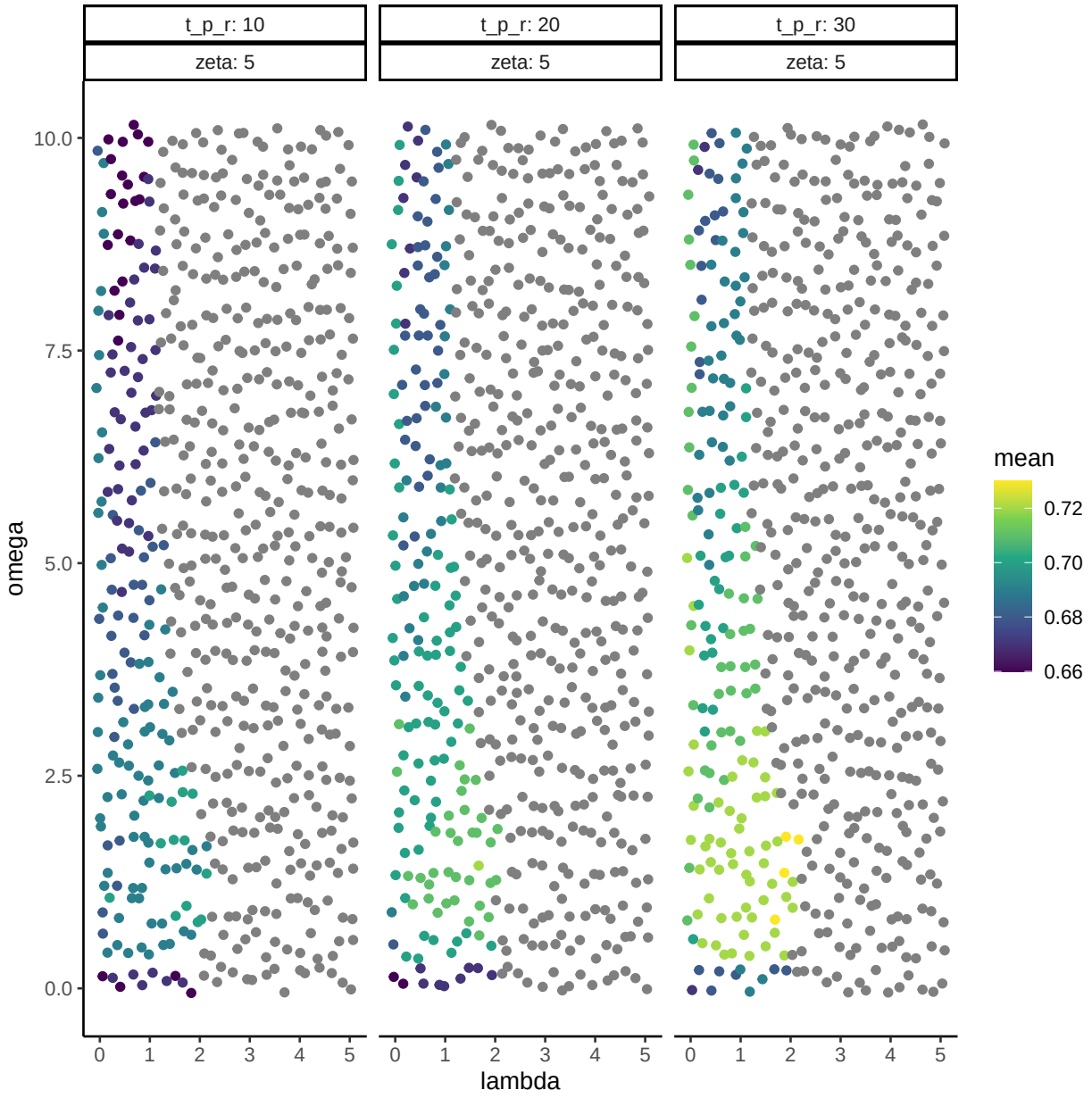
```
params %>% group_by(t_p_r,omega,zeta,lambda, seed) %>% summarize(mean = mean(pcorrect)) %>%
  mutate(mean = round(mean,2)) %>%
  ggplot(aes(y = omega, x = lambda))+
  geom_raster(aes(fill = mean))+
  facet_wrap(t_p_r~seed, labeller = label_both)+
  scale_fill_continuous(type = "viridis")+
  theme_classic()
```

```
## `summarise()` has grouped output by 't_p_r', 'omega', 'zeta', 'lambda'. You can
## override using the `.groups` argument.
```



```
  # theme_classic()+
  # geom_point(data = optimals, size = 1)+
  # geom_point(data = optimal, size = 2, col = "red")
  #

params %>% group_by(t_p_r,omega,zeta,lambda) %>% summarize(mean = mean(pcorrect)) %>%
  mutate(mean = round(mean,2)) %>%
  ggplot(aes(y = omega, x = lambda, col = mean))+
  geom_jitter()+
  facet_wrap(t_p_r~zeta, labeller = label_both)+
  scale_color_continuous(type = "viridis",)+
  theme_classic()
```

```
## `summarise()` has grouped output by 't_p_r', 'omega', 'zeta'. You can override
## using the `.groups` argument.
```




```
hgf_agent = function(parameters){

  sigmoid = function(x) {
    1 / (1 + exp(-x))}


  seed = parameters$seed
  set.seed(seed)

  trials_per_reversal = parameters$trials_per_reversal
  #n_reversals = parameters$n_reversals

  u = c(rbinom(50,1,0.8),
        rbinom(trials_per_reversal,1,0.2),
```

```
        rbinom(trials_per_reversal,1,0.8),
        rbinom(trials_per_reversal,1,0.2),
        rbinom(trials_per_reversal,1,0.8),
        rbinom(trials_per_reversal,1,0.2),
        rbinom(trials_per_reversal,1,0.8),
        rbinom(50,1,0.2))

  N = length(u)


  theta = parameters$theta
  kappa = parameters$kappa
  omega = parameters$omega
  zeta = parameters$zeta


  mu1hat = array(NA,N+1)
  pe1 = array(NA,N+1)
  pe2 = array(NA,N+1)
  w2 = array(NA,N+1)
  r2 = array(NA,N+1)
  pi3hat = array(NA,N+1)
  pi3 = array(NA,N+1)
  sa1hat = array(NA,N+1)
  mu2 = array(NA,N+1)
  sa2 = array(NA,N+1)
  sa2hat = array(NA,N+1)
  mu3 = array(NA,N+1)
  sa3 = array(NA,N+1)
  y = array(NA,N+1)
  belief = array(NA,N+1)


  mu2[1] = 0
  sa2[1] = 4
  mu3[1] = 0
  sa3[1] = 1


agg = tryCatch({
      for(t in 2:(N+1)){

        mu1hat[t] = sigmoid(mu2[t-1])

        sa2hat[t] = sa2[t-1]+exp(kappa*mu3[t-1]+omega)

        sa1hat[t] = mu1hat[t]*(1-mu1hat[t])

        pe1[t] = u[t-1]-mu1hat[t]


        sa2[t] = 1/((1/sa2hat[t])+sa1hat[t])
```

```
        mu2[t] = (mu2[t-1]+pe1[t]*sa2[t])


        pe2[t] = ((sa2[t]+(mu2[t]-mu2[t-1])^2)/(sa2[t-1]+exp(kappa*mu3[t-1]+omega)))-1

        r2[t] = (exp(kappa*mu3[t-1]+omega)-sa2[t-1])/(sa2[t-1]+exp(kappa*mu3[t-1]+omega))

        w2[t] = exp(kappa*mu3[t-1]+omega)/(sa2[t-1]+exp(kappa*mu3[t-1]+omega))

        pi3hat[t] = 1/(sa3[t-1]+theta)

        pi3[t] = pi3hat[t]+(kappa^2/2)*w2[t]*(w2[t]+r2[t]*pe2[t])

        sa3[t] = 1/pi3[t]

        mu3[t] = mu3[t-1]+sa3[t]*(kappa/2)*w2[t]*pe2[t]

        belief[t] = mu1hat[t]^zeta/(mu1hat[t]^zeta+(1-mu1hat[t])^zeta)

        y[t] = rbinom(1,1,belief[t])

    }

    resp = data.frame(resp = y[2:(N+1)])
    resp$u = u
    resp$trial = 1:N


    resp = resp %>% mutate(correct = ifelse(resp == u, 1, 0))

    agg = data.frame(pcorrect =  sum(resp$correct)/N,
                     correct = sum(resp$correct),
                     omega = omega,
                     theta = theta,
                     kappa = kappa,
                     zeta = zeta,
                     trials = N,
                     seed = seed,
                     t_p_r = trials_per_reversal)


    }, warning=function(w) {
        ## do something about the warning, maybe return 'NA'
        return(data.frame(pcorrect = NA,
                     correct = NA,
                     omega = omega,
                     theta = theta,
                     zeta = zeta,
                     kappa = kappa,
                     seed = seed,
                     trials = N,
                     t_p_r = trials_per_reversal))
})
```

```
return(list(agg))
}
```

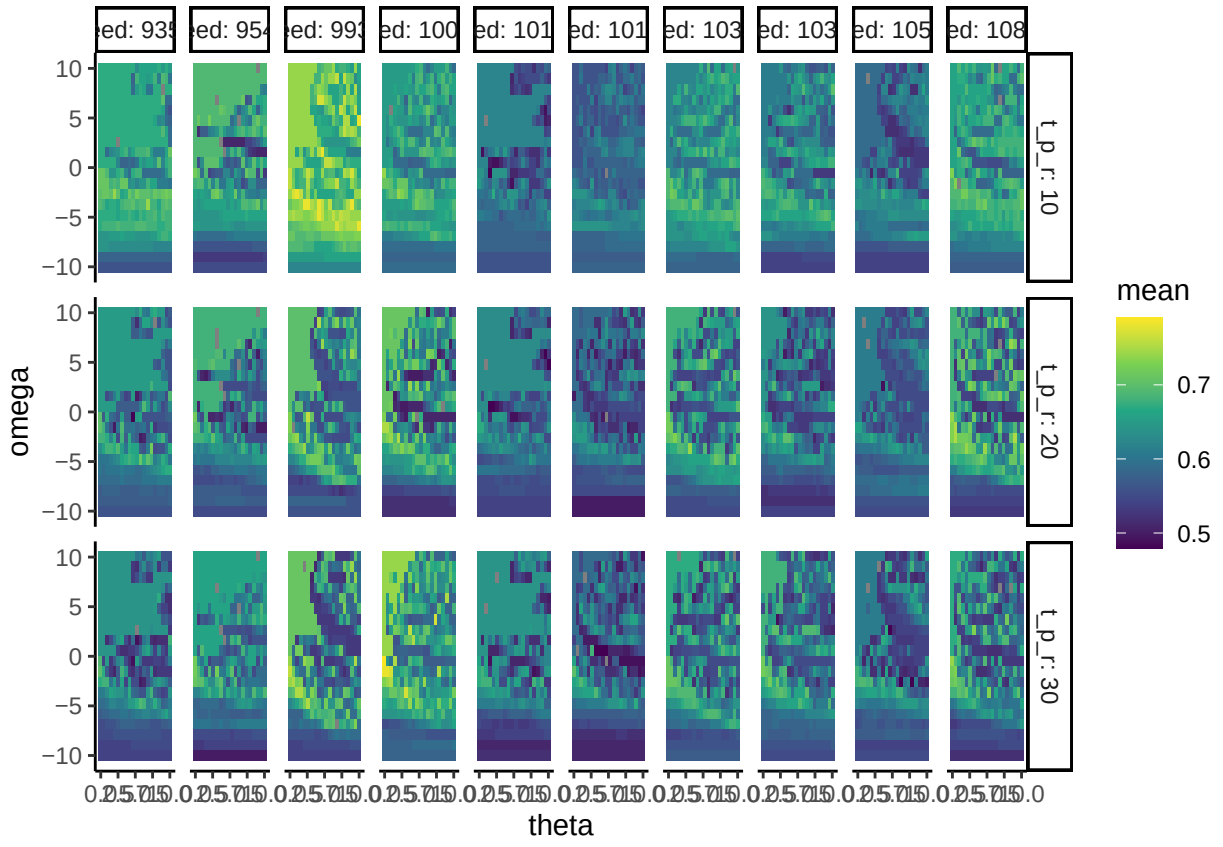
```
hgf_agent(parameters = data.frame(seed = 1113,
                          trials_per_reversal = 20,
                          theta = 2,
                          omega = -1,
                          kappa = 1,
                          zeta = 3))
```

```
## [[1]]
##    pcorrect correct omega theta kappa zeta trials seed t_p_r
## 1 0.6454545     142    -1     2     1    3    220 1113    20
```

```
trials_per_reversal = seq(10,30, by = 10)
#trials_per_reversal = seq(20, length.out = 1)


omega = seq(-10,10, length.out = 20)
theta = seq(0,10, length.out = 20)
kappa = seq(1, length.out = 1)
zeta = seq(5,5, by = 1)


replicate = 1



parameters = expand.grid(trials_per_reversal = trials_per_reversal,
                          omega= omega,
                          theta = theta,
                          kappa = kappa,
                          seed = seed,
                          zeta = zeta,
                          replicate = replicate) %>%
  mutate(id = 1:nrow(.))

data_list <- split(parameters, parameters$id)


cores = availableCores()-1

plan(multisession, workers = 4)

results <- future_map(data_list, ~hgf_agent(.x), .progress = TRUE, .options = furrr_options(seed = TRUE

params = map_dfr(results, 1)
```

```
params2 = na.omit(params)

## seed analysis

optimals = params2[params2$pcorrect == max(params2$pcorrect, na.rm = T) | params2$pcorrect > max(params2
optimal = params2[params2$pcorrect == max(params2$pcorrect, na.rm = T),]

params %>% group_by(t_p_r,omega,theta, seed) %>% summarize(mean = mean(pcorrect)) %>%
  mutate(mean = round(mean,2)) %>%
  ggplot(aes(y = omega, x = theta))+
  geom_raster(aes(fill = mean))+
  facet_grid(t_p_r~seed, labeller = label_both)+
  scale_fill_continuous(type = "viridis")+
  theme_classic()
```

```
## `summarise()` has grouped output by 't_p_r', 'omega', 'theta'. You can override
## using the `.groups` argument.
```




```
  # theme_classic()+
  # geom_point(data = optimals, size = 1)+
  # geom_point(data = optimal, size = 2, col = "red")+theme(axis.text.x = element_text(angle = 90, vjus
  #
```