A step-by-step guide to reinforcement learning evidence accumulation models of instrumental learning tasks

Steven Miletić, Niek Stevenson, (?), Andrew Heathcote

29 November, 2025

Introduction

In this tutorial, we will demonstrate how to design combinations of reinforcement learning and evidence accumulation models (RL-EAMs) and fit them to data from instrumental learning tasks using hierarchical Bayesian methods. We will utilize the *EMC2* package, assuming some prior knowledge of its functionalities (for reference, see the *EMC2* tutorial (Stevenson et al. 2024)).

First, let's load the required packages and data (note that you will also need the Hmisc package installed):

```r
rm(list = ls())
# TMP -- install the right branch
# remotes::install_github("ampl-psych/EMC2@trendy_customkernel_fix",dependencies=TRUE, Ncpus=8);.rs.re
# END TMP
library(EMC2)
library(Hmisc)
set.seed(1) # for reproducibility

# This is some code that contains plotting functions used later on in this tutorial
# code_url <- paste0(
#   "https://raw.githubusercontent.com/StevenM1/",
#   "dynamic_tutorial/refs/heads/main/plotting_utils.R")
code_url = './RL_plotting_utils.R'
source(code_url)
```

While *EMC2* is flexibly designed to allow researchers to implement any mapping, in most applications of RL-EAMs, it is assumed that drift rates are a linear function of the Q-values. Key is to map the Q-values to the accumulator drift rates. This entails that we:

1. Specify the covariates in the data;
2. Apply the delta-rule to each covariate separately;
3. Define the trial-wise mapping between covariates and each accumulator.

1. Covariate specification

In this section, we start by formatting the empirical data. We start by downloading dataset 1 from [Miletic et al elife]. It contains 24 columns of potentially useful variables; here, we aim to extract the minimum we need for modelling the data. Some are columns that *EMC2* always requires for choice tasks: columns indicating subject number (`subjects`), stimulus (`S`), and response and response time (`R`, `rt`). `S` and `R` are usually response-coded, meaning they typically refer to the response location or hand (left/right) or response button (e.g., 'z', 'm', 'left arrow key'). Let us start with extracting those:

```r
load('./data/data_exp1.RData')
# exclude subjects based on criteria of Miletic et al 2021
dat <- dat[!dat$excl,]
# exclude trials without a response
dat <- dat[!is.na(dat$rt),]

# Format subjects as factor and rename column
dat$subjects <- as.factor(dat$pp)

# Format response to factor: left or right
dat$R <- factor(dat$choiceDirection, levels=c('left', 'right'))

# In order to determine which direction was 'optimal', we can check
# whether p_win_left was larger than p_win_right
dat$S <- factor(ifelse(dat$p_win_left>dat$p_win_right, 'left', 'right'),
                levels=c('left', 'right'))
head(dat[,c('subjects', 'S', 'R', 'rt')])
```

```
     subjects     S     R       rt
3121        7  left  left 0.750092
3122        7  left right 0.866725
3123        7 right right 1.100030
3124        7  left  left 0.708232
3125        7 right right 1.174540
3126        7  left  left 1.483390
```

Instrumental learning tasks minimally add two pieces of information: the `feedback` (or `reward`) given on each trial, and the `symbol` that the feedback was associated with. It is not necessary that the feedback corresponds to the *chosen* symbol, such as in experiments with counterfactual feedback (Palminteri et al. 2017; Palminteri et al. 2015) - we only need to know to *which* symbol the feedback corresponds. Let's extract these:

```r
# In this experiment, there's both an `outcome` and `reward` column;
# in this specific experiment these are the same, but in later experiments
# in the same paper these could differ. Here, we arbitrarily pick `reward`,
# and recode it to [0, 1]
dat$reward <- dat$reward / 100

# Let's extract the symbols that were presented as `s_left` and `s_right`.
# and let's add a prefix "sym_"
dat$s_left <- paste0('sym_', dat$appear_left)
dat$s_right <- paste0('sym_', dat$appear_right)

head(dat[,c('subjects', 'S', 'R', 'rt', 'reward', 's_left', 's_right')])
```

```
     subjects     S     R       rt reward s_left s_right
3121        7  left  left 0.750092      1  sym_D   sym_e
3122        7  left right 0.866725      0  sym_T   sym_t
3123        7 right right 1.100030      0  sym_e   sym_D
3124        7  left  left 0.708232      0  sym_J   sym_h
3125        7 right right 1.174540      1  sym_k   sym_K
3126        7  left  left 1.483390      1  sym_J   sym_h
```

```
# based on the combination of R and s_left and s_right, we can now determine
# which symbol was chosen, and thus, corresponds to the reward.
```

Additionally, for simulating new data (but, strictly speaking, not for fitting), we need to be able to generate feedback in the same way that feedback was generated in the experiment. In the easiest case, feedback is deterministic and a feedback generator will simply map a chosen symbol to a corresponding reward. But in more realistic examples it will require sampling a random number from a binomial or Gaussian distribution, with the parameters of that distribution varying between symbols. As such, for simulating new data and assessing quality of fit, we need to know the feedback distribution parameters for each symbol and trial.

```
dat$p_left <- dat$p_win_left
dat$p_right <- dat$p_win_right

dat <- dat[,c('subjects', 'S', 'R', 'rt', 'reward',
              's_left', 's_right', 'p_left', 'p_right')]
head(dat)
```

```
     subjects     S     R       rt reward s_left s_right p_left p_right
3121        7  left  left 0.750092      1  sym_D   sym_e   0.70    0.30
3122        7  left right 0.866725      0  sym_T   sym_t   0.80    0.20
3123        7 right right 1.100030      0  sym_e   sym_D   0.30    0.70
3124        7  left  left 0.708232      0  sym_J   sym_h   0.65    0.35
3125        7 right right 1.174540      1  sym_k   sym_K   0.40    0.60
3126        7  left  left 1.483390      1  sym_J   sym_h   0.65    0.35
```

Note that the column names of `reward`, `s_left`, `s_right`, `p_left`, and `p_right` are arbitrary - as long as the user's own functions refer to the correct columns, their names can be anything.

For fitting in *EMC2*, we need to create one column per symbol, with the values indicating the feedback. If no feedback was given on the symbol, the value should be `NA`. An intuitive option would be to add such columns to the data. However, all columns in the data (except for `rt` and `R`) are assumed to be static factors of the design. Keep in mind that we also want to be able to *simulate* new data. And once we simulate new data, the covariate columns should depend on the simulated behavior. As such, the covariate columns are *not* factors of the design. For this reason, we should not add the covariate columns to the data directly, but rather define *functions* that create covariate columns based on the data. When simulating new data, these functions are re-applied after each trial, ensuring that the covariate columns are correct for simulated data. Here's the idea:

```
# First, let's define a function that creates column 'RS': response symbol.
RS <- function(data) {
  # which *symbol* was chosen?
  ifelse(data$R=='left', data$s_left, data$s_right)
}

# make a copy of the data (tmp) to illustrate the result if this function
# is applied to the data - but don't add the factor to the data.
tmp <- dat
tmp$RS <- RS(tmp)
head(tmp)
```

```
     subjects     S     R       rt reward s_left s_right p_left p_right    RS
3121        7  left  left 0.750092      1  sym_D   sym_e   0.70    0.30 sym_D
3122        7  left right 0.866725      0  sym_T   sym_t   0.80    0.20 sym_t
3123        7 right right 1.100030      0  sym_e   sym_D   0.30    0.70 sym_D
```

```
3124         7  left  left 0.708232      0  sym_J   sym_h   0.65    0.35 sym_J
3125         7 right right 1.174540      1  sym_k   sym_K   0.40    0.60 sym_K
3126         7  left  left 1.483390      1  sym_J   sym_h   0.65    0.35 sym_J
```

```
# With RS known, we can create a covariate column for symbol D, e, t, etc.
make_covariate_column <- function(data, covariate_symbol) {
    data[,covariate_symbol] <- NA
    data[data$RS == covariate_symbol, covariate_symbol] <- data[data$RS == covariate_symbol, 'reward']
    data[[covariate_symbol]]
}

tmp$sym_D <- make_covariate_column(tmp, "sym_D")
tmp$sym_e <- make_covariate_column(tmp, "sym_e")
tmp$sym_t <- make_covariate_column(tmp, "sym_t")
head(tmp)
```

```
     subjects     S     R       rt reward s_left s_right p_left p_right    RS
3121        7  left  left 0.750092      1  sym_D   sym_e   0.70    0.30 sym_D
3122        7  left right 0.866725      0  sym_T   sym_t   0.80    0.20 sym_t
3123        7 right right 1.100030      0  sym_e   sym_D   0.30    0.70 sym_D
3124        7  left  left 0.708232      0  sym_J   sym_h   0.65    0.35 sym_J
3125        7 right right 1.174540      1  sym_k   sym_K   0.40    0.60 sym_K
3126        7  left  left 1.483390      1  sym_J   sym_h   0.65    0.35 sym_J
     sym_D sym_e sym_t
3121     1    NA    NA
3122    NA    NA     0
3123     0    NA    NA
3124    NA    NA    NA
3125    NA    NA    NA
3126    NA    NA    NA
```

```
# To facilitate, we can wrap this with a function generator -
# that is, a function that returns a function, one for each column
# Each of those returned function generates a covariate column
# with the appropriate structure.
covariate_column_generator <- function(col_name) {
  function(data) {
    data[,col_name] <- NA
    data[data$RS == col_name, col_name] <- data[data$RS == col_name, 'reward']
    data[[col_name]]
  }
}

all_symbols <- unique(c(dat$s_left, dat$s_right))
make_covariate_columns <- setNames(lapply(all_symbols,
                                          covariate_column_generator), all_symbols)
```

Now, `make_covariate_columns` is a list of functions that should be applied to the data. We can tell *EMC2* to do this by providing the functions to the `design`:

```
design_RDM <- design(model=RDM,
                     data=dat,
                     functions=c(list(RS=RS), make_covariate_columns),
                     matchfun=function(d) d$S==d$lR,
                     formula=list(B ~ 1, v ~ 1, t0 ~ 1))
```

```
 Sampled Parameters:
[1] "B"  "v"  "t0"

 Design Matrices:
$B
 B
 1

$v
 v
 1

$t0
 t0
  1

$A
 A
 1

$s
 s
 1
```

```
# Once the design is combined with the data, the functions are applied.
# and we can find all covariate columns in the `dadm`:
emc <- make_emc(dat, design_RDM)
head(emc[[1]]$data[[1]])
```

```
  subjects     S     R        rt reward s_left s_right p_left p_right trials
1        7  left  left 0.7500000      1  sym_D   sym_e    0.7     0.3      1
2        7  left  left 0.7500000      1  sym_D   sym_e    0.7     0.3      1
3        7  left right 0.8666667      0  sym_T   sym_t    0.8     0.2      2
4        7  left right 0.8666667      0  sym_T   sym_t    0.8     0.2      2
5        7 right right 1.1000000      0  sym_e   sym_D    0.3     0.7      3
6        7 right right 1.1000000      0  sym_e   sym_D    0.3     0.7      3
     lR    lM winner    RS sym_D sym_T sym_e sym_J sym_k sym_h sym_t sym_K
1  left  TRUE   TRUE sym_D     1    NA    NA    NA    NA    NA    NA    NA
2 right FALSE  FALSE sym_D     1    NA    NA    NA    NA    NA    NA    NA
3  left  TRUE  FALSE sym_t    NA    NA    NA    NA    NA    NA     0    NA
4 right FALSE   TRUE sym_t    NA    NA    NA    NA    NA    NA     0    NA
5  left FALSE  FALSE sym_D     0    NA    NA    NA    NA    NA    NA    NA
6 right  TRUE   TRUE sym_D     0    NA    NA    NA    NA    NA    NA    NA
```

2. Apply the delta rule

Handling trialwise covariates works through `trend` objects in *EMC2*. In this object, you specify (1) the covariate of interest, (2) a kernel to apply to the covariate, and (3) which decision parameter is informed by the resulting covariate, and (4) the functional form of the mapping between the resulting covariate and the decision parameters, referred to as the 'base'. Now that we have prepared the data-augmented design matrix (`dadm`) (so that it contains all covariates as columns), we can now specify a `trend` object. The kernel that we want to apply is the `delta` rule, and it should inform drift rates `v` linearly, so we use a `linear` `base`.

```
trend <- make_trend(par_names='v',
                    cov_names=list(all_symbols),
                    kernels='delta',
                    bases='lin',
                    phase='posttransform')

design_RDM <- design(model=RDM,
                     data=dat,
                     functions=c(list(RS=RS), make_covariate_columns),
                     matchfun=function(d) d$S==d$lR,
                     formula=list(B ~ 1, v ~ 1, t0 ~ 1),
                     trend=trend)
```

```
 Sampled Parameters:
[1] "B"       "v"       "t0"      "v.w"     "v.q0"    "v.alpha"

 Design Matrices:
$B
 B
 1

$v
 v
 1

$t0
 t0
  1

$v.w
 v.w
   1

$v.q0
 v.q0
    1

$v.alpha
 v.alpha
       1

$A
 A
 1

$s
 s
 1
```

```
emc <- make_emc(dat, design_RDM)

# To illustrate the effect
kernel_pars <- c('v.q0'=0, 'v.alpha'=0.2)
```

```
head(cbind(emc[[1]]$data[[1]][,all_symbols], apply_kernel(kernel_pars, emc = emc, subject=1)))
```

```
  sym_D sym_T sym_e sym_J sym_k sym_h sym_t sym_K   1 2 3 4 5 6 7 8
1     1    NA    NA    NA    NA    NA    NA    NA 0.0 0 0 0 0 0 0 0
2     1    NA    NA    NA    NA    NA    NA    NA 0.0 0 0 0 0 0 0 0
3    NA    NA    NA    NA    NA    NA     0    NA 0.2 0 0 0 0 0 0 0
4    NA    NA    NA    NA    NA    NA     0    NA 0.2 0 0 0 0 0 0 0
5     0    NA    NA    NA    NA    NA    NA    NA 0.2 0 0 0 0 0 0 0
6     0    NA    NA    NA    NA    NA    NA    NA 0.2 0 0 0 0 0 0 0
```

3. Define the trial-wise mapping between covariates and each accumulator

You might notice that in the trialwise parameters, the drift rate v is one plus the sum of all Q-values across covariates. By default, *EMC2* applies the base to all updated covariates. However, on each trial, only two stimuli were shown, so only they should influence their respective drift rates. For this, we need to create a mapping between covariates and accumulators. Such a map is a matrix specifying the weight of each covariate for each drift rate - most will be 0. The simplest type of map is an RL-RD, which just maps 1 covariate to 1 accumulator in each row:

```
make_RL_RD_map <- function(dadm, cov_names) {
  dadm$lS <- NA
  dadm[dadm$lR=='left', 'lS'] <- dadm[dadm$lR=='left', 's_left']
  dadm[dadm$lR=='right', 'lS'] <- dadm[dadm$lR=='right', 's_right']

  # Return 1 if the column name matches the accumulator's latent stimulus
  # and 0 otherwise
  map <- sapply(cov_names, function(col) ifelse(dadm$lS == col, 1, 0))
  map
}

head(make_RL_RD_map(emc[[1]]$data[[1]], all_symbols))
```

```
     sym_D sym_T sym_e sym_J sym_k sym_h sym_t sym_K
[1,]     1     0     0     0     0     0     0     0
[2,]     0     0     1     0     0     0     0     0
[3,]     0     1     0     0     0     0     0     0
[4,]     0     0     0     0     0     0     1     0
[5,]     0     0     1     0     0     0     0     0
[6,]     1     0     0     0     0     0     0     0
```

```
# Combine with trend and design
trend <- make_trend(par_names='v',
                    cov_names=list(all_symbols),
                    kernels='delta',
                    bases='lin',
                    phase='posttransform',
                    maps = list(make_RL_RD_map))

design_RDM <- design(model=RDM,
                     data=dat,
                     functions=c(list(RS=RS), make_covariate_columns),
                     matchfun=function(d) d$S==d$lR,
                     formula=list(B ~ 1, v ~ 1, t0 ~ 1),
                     trend=trend)
```

```
 Sampled Parameters:
[1] "B"        "v"        "t0"       "v.w_map1" "v.q0"     "v.alpha" 

 Design Matrices:
$B
 B
 1

$v
 v
 1

$t0
 t0
  1

$v.w_map1
 v.w_map1
        1

$v.q0
 v.q0
    1

$v.alpha
 v.alpha
       1

$A
 A
 1

$s
 s
 1
```

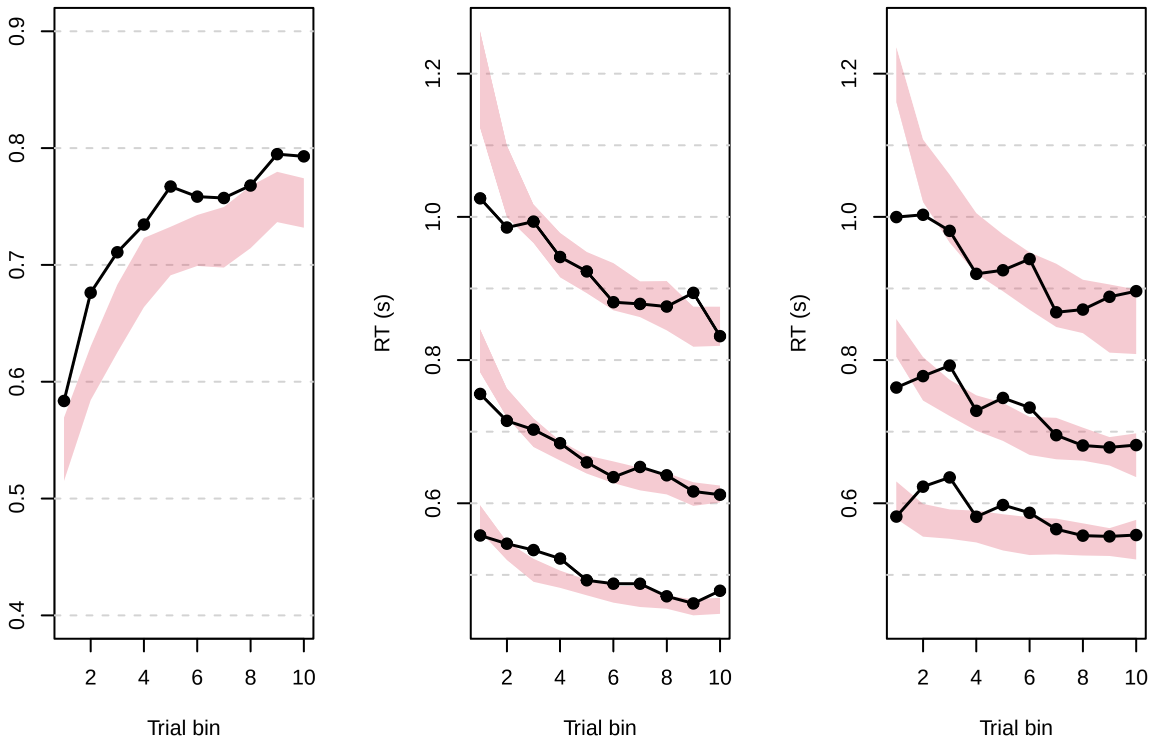
```
emc <- make_emc(dat, design_RDM)
```

Note how the drift rate is now only influenced by the covariate that corresponds to the symbol that the accumulator codes for.

We are now ready to fit the RL-RD!

```
emc <- fit(emc, cores_per_chain=10, fileName='./samples/exp1_subset.RData')
```

```
load('./samples/exp1_subset.RData')
ppC <- predict(emc, n_cores=40)
plot_exp1(dat=dat, pp=ppC)
```



RL-DDM

The above illustrates the basics, and with these, it is straightforward to construct other RL-EAMs. In most cases, it is a matter of changing the mapping function, and potentially the EAM itself. For example, a popular choice is the RL-DDM, which is the following:

```r
make_RL_DDM_map <- function(dadm, cov_names) {
  # Here, we simply find the symbol on the left and right,
  # and subtract one from the other
  map_left <- sapply(cov_names, function(col) ifelse(dadm$s_left == col, 1, 0))
  map_right <- sapply(cov_names, function(col) ifelse(dadm$s_right == col, 1, 0))
  map <- map_left-map_right
  map
}

# Combine with trend and design
trend <- make_trend(par_names='v',
                    cov_names=list(all_symbols),
                    kernels='delta',
                    bases='lin',
                    phase='posttransform',
                    maps = list(make_RL_DDM_map))

design_DDM <- design(model=DDM,
                     data=dat,
                     functions=c(list(RS=RS), make_covariate_columns),
                     matchfun=function(d) d$S==d$lR,
                     formula=list(a ~ 1, v ~ 1, t0 ~ 1),
                     trend=trend)
```

```
 Sampled Parameters:
[1] "a"        "v"        "t0"       "v.w_map1" "v.q0"     "v.alpha" 

 Design Matrices:
$a
 a
 1

$v
 v
 1

$t0
 t0
  1

$v.w_map1
 v.w_map1
        1

$v.q0
 v.q0
    1

$v.alpha
 v.alpha
       1

$s
 s
 1

$st0
 st0
   1

$sv
 sv
  1

$SZ
 SZ
  1

$Z
 Z
 1
```

```
emc <- make_emc(dat, design_DDM)
head(attr(emc[[1]]$data[[1]], 'covariate_maps')[[1]])
```

```
     sym_D sym_T sym_e sym_J sym_k sym_h sym_t sym_K
[1,]     1     0    -1     0     0     0     0     0
[2,]     0     1     0     0     0     0    -1     0
[3,]    -1     0     1     0     0     0     0     0
[4,]     0     0     0     1     0    -1     0     0
[5,]     0     0     0     0     1     0     0    -1
[6,]     0     0     0     1     0    -1     0     0
```

```
emc <- fit(emc, cores_per_chain=20, fileName='./samples/exp1_subset_rlddm.RData')
```

However, the RL-DDM does not fit the response time distributions very well:

```
## plot here
load('./samples/exp1_subset_rlddm.RData')
ppC <- predict(emc, n_cores=40)
plot_exp1(dat=dat, pp=ppC)
```

An alternative option is the RL-ARD, which combines a difference and a sum map in a race architecture:

```
make_RL_ARD_difference <- function(dadm, cov_names) {
  dadm$lS <- NA
  dadm[dadm$lR=='left', 'lS'] <- dadm[dadm$lR=='left', 's_left']
  dadm[dadm$lR=='right', 'lS'] <- dadm[dadm$lR=='right', 's_right']

  dadm$lSother <- NA
  dadm[dadm$lR=='left', 'lSother'] <- dadm[dadm$lR=='left', 's_right']
  dadm[dadm$lR=='right', 'lSother'] <- dadm[dadm$lR=='right', 's_left']

  # Return 1 if the column name matches the accumulator's latent stimulus
  # Return -1 if the column name matches the *other* accumulator's latent stimulus
  # and 0 otherwise
  maplS <- sapply(cov_names, function(col) ifelse(dadm$lS == col, 1, 0))
  maplSother <- sapply(cov_names, function(col) ifelse(dadm$lSother == col, 1, 0))
  maplS-maplSother
}

make_RL_ARD_sum <- function(dadm, cov_names) {
  dadm$lS <- NA
  dadm[dadm$lR=='left', 'lS'] <- dadm[dadm$lR=='left', 's_left']
  dadm[dadm$lR=='right', 'lS'] <- dadm[dadm$lR=='right', 's_right']

  dadm$lSother <- NA
  dadm[dadm$lR=='left', 'lSother'] <- dadm[dadm$lR=='left', 's_right']
  dadm[dadm$lR=='right', 'lSother'] <- dadm[dadm$lR=='right', 's_left']

  # Return 1 if the column name matches the accumulator's latent stimulus
  # Return 1 if the column name matches the *other* accumulator's latent stimulus
  # and 0 otherwise
  maplS <- sapply(cov_names, function(col) ifelse(dadm$lS == col, 1, 0))
  maplSother <- sapply(cov_names, function(col) ifelse(dadm$lSother == col, 1, 0))
  maplS+maplSother
```

```r
}

# Combine with trend and design
trend <- make_trend(par_names='v',
                    cov_names=list(all_symbols),
                    kernels='delta',
                    bases='lin',
                    phase='posttransform',
                    maps = list('d'=make_RL_ARD_difference,
                                's'=make_RL_ARD_sum))

design_RDM <- design(model=RDM,
                     data=dat,
                     functions=c(list(RS=RS), make_covariate_columns),
                     matchfun=function(d) d$S==d$lR,
                     formula=list(B ~ 1, v ~ 1, t0 ~ 1),
                     trend=trend)
```

```
 Sampled Parameters:
[1] "B"       "v"       "t0"      "v.w_d"   "v.w_s"   "v.q0"    "v.alpha"

 Design Matrices:
$B
 B
 1

$v
 v
 1

$t0
 t0
  1

$v.w_d
 v.w_d
     1

$v.w_s
 v.w_s
     1

$v.q0
 v.q0
    1

$v.alpha
 v.alpha
       1

$A
 A
 1
```

```
$s
 s
 1
```

```
emc <- make_emc(dat, design_RDM)
```

```
emc <- fit(emc, cores_per_chain=20, fileName='./samples/exp1_subset_rlard.RData')
```

```
load('./samples/exp1_subset_rlard.RData')
ppC <- predict(emc, n_cores=40)
plot_exp1(dat=dat, pp=ppC)
```

–>

–>

–>

–>

Palminteri, Stefano, Mehdi Khamassi, Mateus Joffily, and Giorgio Coricelli. 2015. "Contextual Modulation of Value Signals in Reward and Punishment Learning." *Nature Communications* 6. https://doi.org/10.1038/ncomms9096.

Palminteri, Stefano, Germain Lefebvre, Emma J. Kilford, and Sarah-Jayne Blakemore. 2017. "Confirmation Bias in Human Reinforcement Learning: Evidence from Counterfactual Feedback Processing." *PLOS Computational Biology* 13 (8): e1005684. https://doi.org/10.1371/journal.pcbi.1005684.

Stevenson, Niek, Michelle Donzallaz, Reilly James Innes, Birte Forstmann, Dora Matzke, and Andrew Heathcote. 2024. "EMC2: An R Package for Cognitive Models of Choice." https://doi.org/10.31234/osf.io/2e4dq.