Modelling structured trial-by-trial variability in evidence accumulation

Steven Miletić et al.

25 September, 2025

Introduction

In this tutorial, we will demonstrate how to design dynamic evidence accumulation models and fit them to data using hierarchical Bayesian methods. We will utilize the *EMC2* package, assuming some prior knowledge of its functionalities (for reference, see the *EMC2* tutorial). Specifically, we will fit models to dataset 1 from Miletic et al. (2025), which corresponds to the 'choice, short [CS]' condition from Wagenmakers et al. (2004). In this experiment, participants were presented with a single digit and required to determine whether the digit was even or odd.

First, let's load the required packages and data:

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

# This is some code that contains plotting functions used later on in this tutorial
source('./plotting_utils.R')

# Load in the data
load("datasets/dataset_1.RData")

# Inspect the data
print(head(dat))
```

```
  subjects    S    R    rt
1        1 even even 0.542
2        1 even even 0.426
3        1 even even 0.501
4        1 even even 0.403
5        1  odd  odd 0.497
6        1 even even 0.518
```

Here, `subjects` refers to subject number, `S` to the presented stimulus (even/odd), `R` the response (even/odd), and `rt` the response time in seconds.

Baseline models

We begin by setting up the 'static' baseline models without structured trial-by-trial variability. For a comprehensive tutorial on specifying designs in *EMC2*, please refer to the *EMC2* tutorial [ref]. We will set up two baseline models: first using a racing diffusion model (RDM), and then using a diffusion decision model (DDM). In the RDM, we model drift rates with a mean-difference parametrization. The `mapped_pars()` function is useful for tracking how parameter vectors are mapped onto each design cell:

```
# set up contrast matrix for mean-difference parametrisation
ADmat <- matrix(c(-.5,.5), ncol=1, dimnames=list(NULL,'d'))
design_RDM <- design(model=RDM,
                 data=dat,
                 contrast=list(lM=ADmat),
                 matchfun=function(d) d$S==d$lR,
                 formula=list(B ~ 1, v ~ lM, t0 ~ 1))
```

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

 Design Matrices:
$B
 B
 1

$v
    lM v v_lMd
  TRUE 1   0.5
 FALSE 1  -0.5

$t0
 t0
  1

$A
 A
 1

$s
 s
 1
```

```
mapped_pars(design_RDM)
```

```
$v
  lM
 TRUE   : exp(v + 0.5 * v_lMd)
 FALSE  : exp(v - 0.5 * v_lMd)
```

In the DDM, we need to remember to flip the sign of the drift rate for one of the two stimulus types (in this case, even):

```r
Smat <- matrix(c(-1,1), nrow = 2,dimnames=list(NULL,"dif"))
design_DDM <- design(model=DDM,
                     data=dat,
                     contrasts=list(S=Smat),
                     formula=list(Z ~ 1, v ~ S, a~1, t0 ~ 1))
```

```
 Sampled Parameters:
[1] "Z"      "v"      "v_Sdif" "a"      "t0"

 Design Matrices:
$Z
 Z
 1

$v
    S v v_Sdif
 even 1     -1
  odd 1      1

$a
 a
 1

$t0
 t0
  1

$s
 s
 1

$st0
 st0
   1

$sv
 sv
  1

$SZ
 SZ
  1
```

```r
mapped_pars(design_DDM)
```

```
$v
  S
 even  : v - v_Sdif
 odd   : v + v_Sdif
```

Trend specification

Both the descriptive trends and the formal mechanisms of dynamics work through `trend` objects in *EMC2*. In this object, you specify (1) the covariate of interest (e.g., time on task), (2) a kernel to apply to the covariate (e.g., linear, power, exponential, polynomial, or delta rule), 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'. The `trend_help()` function gives an overview of the options:

```
trend_help()
```

```
Available kernels:
  lin_decr: Decreasing linear kernel: k = -c
  lin_incr: Increasing linear kernel: k = c
  exp_decr: Decreasing exponential kernel: k = exp(-d_ed * c)
  exp_incr: Increasing exponential kernel: k = 1 - exp(-d_ei * c)
  pow_decr: Decreasing power kernel: k = (1 + c)^(-d_pd)
  pow_incr: Increasing power kernel: k = 1 - (1 + c)^(-d_pi)
  poly2: Quadratic polynomial: k = d1 * c + d2 * c^2
  poly3: Cubic polynomial: k = d1 * c + d2 * c^2 + d3 * c^3
  poly4: Quartic polynomial: k = d1 * c + d2 * c^2 + d3 * c^3 + d4 * c^4
  delta: Standard delta rule kernel: k = q[i].
         Updates q[i] = q[i-1] + alpha * (c[i-1] - q[i-1]).
         Parameters: q0 (initial value), alpha (learning rate).
  delta2: Dual kernel delta rule: k = q[i].
          Combines fast and slow learning rates
          and switches between them based on dSwitch.
          Parameters: q0 (initial value), alphaFast (fast learning rate),
          propSlow (alphaSlow = propSlow * alphaFast), dSwitch (switch threshold).
  deltab: Threshold learning delta rule kernel: k = q_[FM,i].
          Updates q[i] = q[i-1] + alpha * (B[i]/c[i-1] - q[i-1]).
          Parameters: q0 (initial value), alpha (learning rate).

Available base types:
  lin: Linear base: parameter + w * k
  exp_lin: Exponential linear base: exp(parameter) + exp(w) * k
  centered: Centered mapping: parameter + w*(k - 0.5)
  add: Additive base: parameter + k
  identity: Identity base: k

Trend options:
  premap: Trend is applied before parameter mapping. This means the trend parameters
          are mapped first, then used to transform cognitive model parameters before
          their mapping.
  pretransform: Trend is applied after parameter mapping but before transformations.
                Cognitive model parameters are mapped first, then trend is applied,
                followed by transformations.
  posttransform: Trend is applied after both mapping and transformations.
                 Cognitive model parameters are mapped and transformed first,
                 then trend is applied.
```

The last section, 'trend options', warrants some extra attention. In *EMC2*, the user can define parameters using model formula language, as we did above when defining the designs. *EMC2* uses these to create a design matrix by mapping the relevant factors to each design cell. In the case of the RDM, once mapped,

in each design cell, only the `v`, `B`, `t0`, `s`, (and optionally `A`) parameters per accumulator remain. However, in many applications, the parameter of interest is defined as a between-accumulator difference (e.g., `v_lMd` in the RDM example above), or a between-condition difference (e.g., the effect of speed-accuracy trade-off cues on thresholds). These parameters only exist pre-mapping, and thus, the trend should be applied prior to mapping by setting `premap` to `TRUE`.

There is an important caveat here. By default, *EMC2* estimates parameters with a lower bound (e.g., non-decision time, thresholds, RDM's drift rates) on the log scale, and parameters with both a lower and upper bound on the probit scale. Parameters are *mapped* on the scale on which they are estimated, and *then* transformed. This also implies that if a trend is applied prior to mapping, the resulting parameters might later be transformed, leading to a potentially unwanted non-linear effect of the covariate on the parameter.

Let's clarify with an example. Let's try to impose a linear trend on thresholds in case of the RDM:

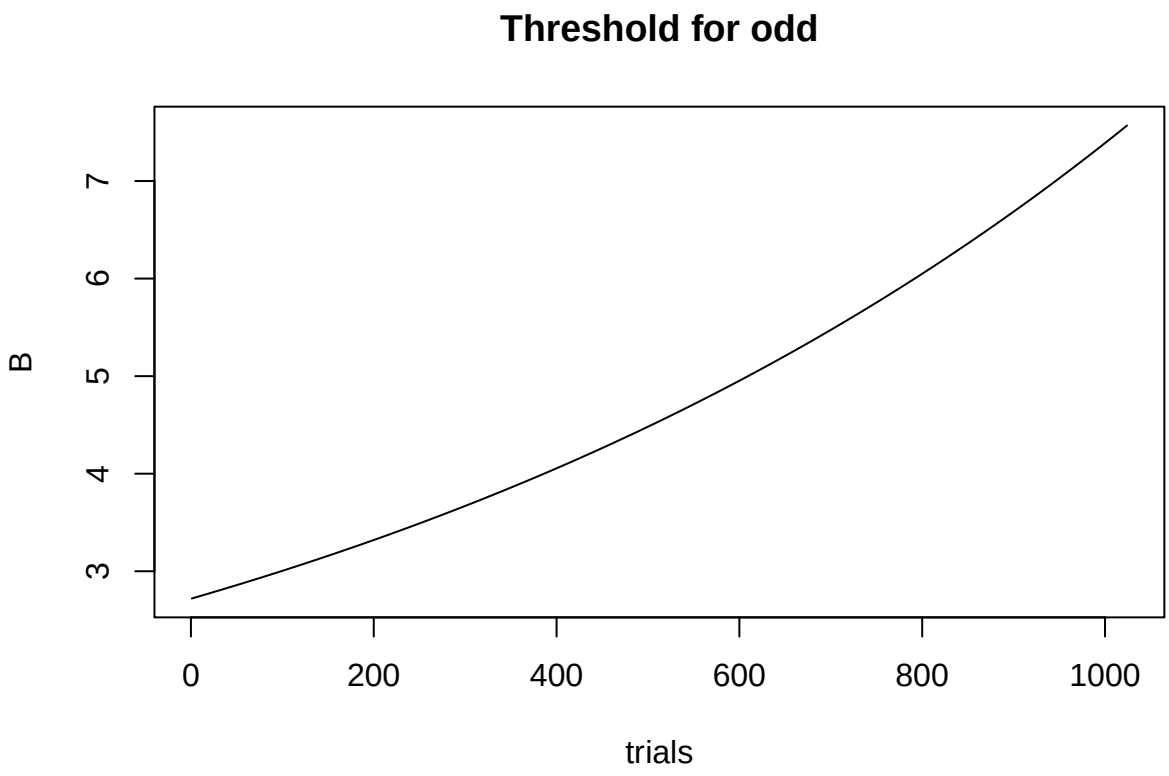
```
# Add a 'trials' column specifying trial number
dat <- EMC2:::add_trials(dat)

# Rescale the effect of this covariate to a larger parameter range to help sampling
dat$trials2 <- dat$trials/1000
lin_trend <- make_trend(cov_names='trials2',
                        kernels = 'lin_incr',
                        par_names='B',
                        bases='lin',
                        premap=TRUE)

design_RDM_lin_B <- design(model=RDM,
                           data=dat,
                           contrast=list(lM=ADmat),
                           covariates='trials2',   # specify relevant covariate columns
                           matchfun=function(d) d$S==d$lR,
                           formula=list(B ~ 1, v ~ lM, t0 ~ 1),
                           trend=lin_trend)       # add trend
```

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

 Design Matrices:
$B
 B
 1

$v
    lM v v_lMd
  TRUE 1   0.5
 FALSE 1  -0.5

$t0
 t0
  1

$B.w
 B.w
   1

$A
```

```
 A
 1

$s
 s
 1
```

Note that we are now sampling one extra parameter compared to before, `B.w`, which is the weight of the influence of trial number on thresholds. In $B_t = B_0 + B.w * trial$. Let's define a set of parameters and look at the resulting trial-by-trial thresholds:

```
emc <- make_emc(dat, design=design_RDM_lin_B)
p_vector <- c('B'=1, 'v'=1, 'v_lMd'=1, 't0'=0.1, 'B.w'=1)

# Visualize trend
plot_trend(p_vector, emc=emc,
           par_name='B', subject=1,
           lR_filter='odd', main='Threshold for odd')
```



Clearly, this is not a linear increase. This happens because the threshold is sampled on the log scale, so an exponential transform is applied after mapping the parameter vector to the design cells. Since the linear trend was applied prior to mapping, this linear effect becomes non-linear. `mapped_pars()` can be used to clarify:

```
mapped_pars(design_RDM_lin_B)
```

```
$B
```

```
 intercept  : exp(B_t)
 Trends:
 B_t = B + B.w * trials2

$v
  lM
 TRUE   : exp(v + 0.5 * v_lMd)
 FALSE  : exp(v - 0.5 * v_lMd)
```

One option to prevent this from happening is to apply the trend after mapping and transformation, as follows:

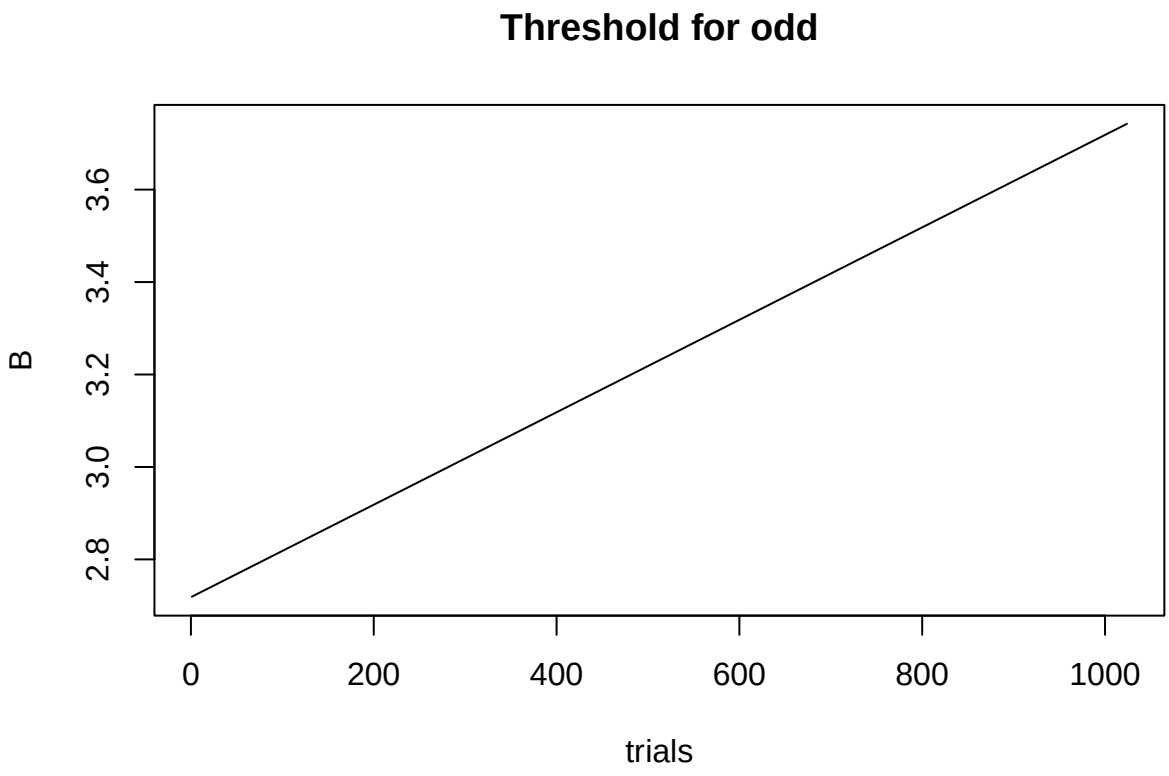
```
lin_trend2 <- make_trend(cov_names='trials2',
                         kernels = 'lin_incr',
                         par_names='B',
                         bases='lin',
                         premap=FALSE, pretransform=FALSE)
design_RDM_lin_B2 <- design(model=RDM,
                          data=dat,
                          contrast=list(lM=ADmat),
                          covariates='trials2',   # specify relevant covariate columns
                          matchfun=function(d) d$S==d$lR,
                          formula=list(B ~ 1, v ~ lM, t0 ~ 1),
                          trend=lin_trend2)       # add trend
```

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

 Design Matrices:
$B
 B
 1

$v
    lM v v_lMd
  TRUE 1   0.5
 FALSE 1  -0.5

$t0
 t0
  1

$B.w
 B.w
   1

$A
 A
 1

$s
 s
 1
```

```
# note how the trend is added after the transformation
mapped_pars(design_RDM_lin_B2)
```

```
$B

 intercept  : exp(B) + B_t
 Trends:
 B_t = B.w * trials2

$v
  lM
 TRUE   : exp(v + 0.5 * v_lMd)
 FALSE  : exp(v - 0.5 * v_lMd)
```

```
emc <- make_emc(dat, design=design_RDM_lin_B2)
p_vector <- c('B'=1, 'v'=1, 'v_lMd'=1, 't0'=0.1, 'B.w'=1)
# Visualize trend
plot_trend(p_vector, emc=emc,
           par_name='B', subject=1,
           lR_filter='odd', main='Threshold for odd')
```



However, posttransform implies postmap, and many parameters of interest are defined only prior to mapping. So applying trends to those premap parameter types is not possible in combination with posttransform. Instead, the user can tell *EMC2* to estimate parameters on their natural scales by turning off transformations of the relevant parameters. For example:

```r
lin_trend3 <- make_trend(cov_names='trials2',
                         kernels = 'lin_incr',
                         par_names='B',
                         bases='lin',
                         premap=TRUE)  # back to premap
design_RDM_lin_B3 <- design(model=RDM,
                          data=dat,
                          contrast=list(lM=ADmat),
                          covariates='trials2',
                          matchfun=function(d) d$S==d$lR,
             # here, we tell EMC2 to sample the threshold on the natural scale
                          transform=list(func=c('B'='identity')),
                          formula=list(B ~ 1, v ~ lM, t0 ~ 1),
                          trend=lin_trend3)       # add trend
```

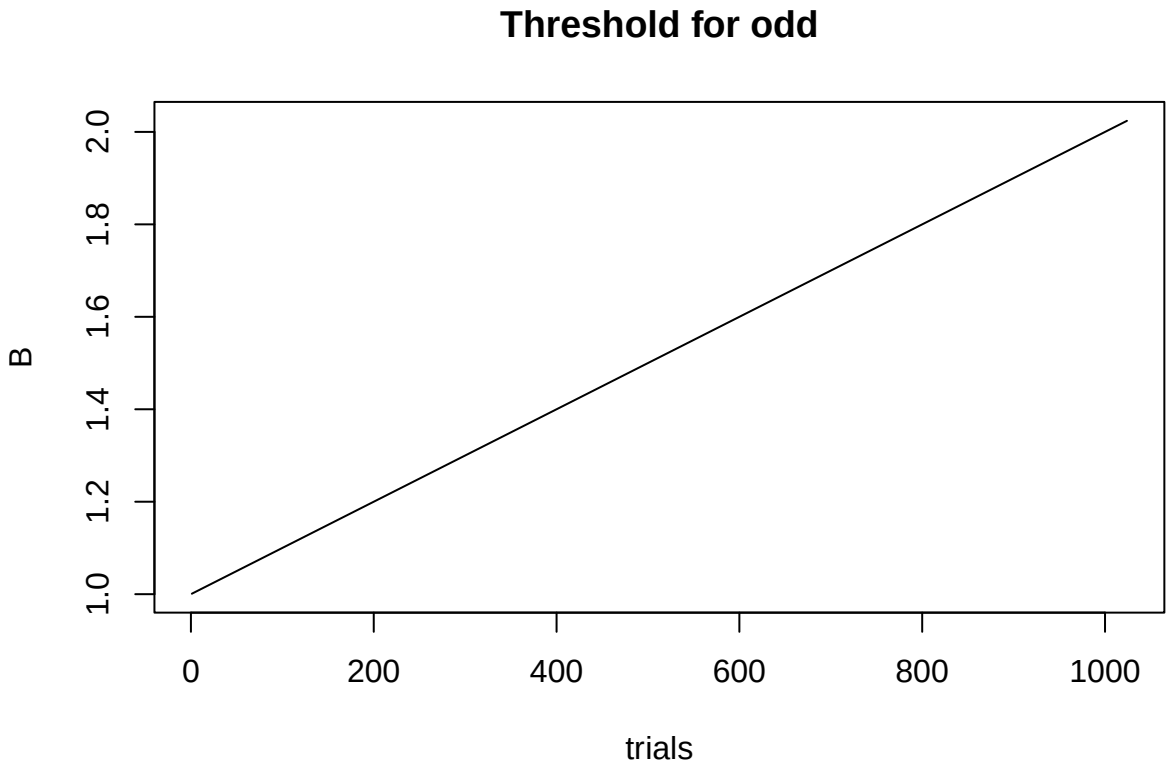
```
 Sampled Parameters:
[1] "B"     "v"     "v_lMd" "t0"    "B.w"

 Design Matrices:
$B
 B
 1

$v
    lM v v_lMd
  TRUE 1   0.5
 FALSE 1  -0.5

$t0
 t0
  1

$B.w
 B.w
   1

$A
 A
 1

$s
 s
 1
```

```r
# note how the transformation is no longer applied at all
mapped_pars(design_RDM_lin_B3)
```

```
$B

 intercept  : B_t
 Trends:
 B_t = B + B.w * trials2
```

```
$v
  lM
 TRUE   : exp(v + 0.5 * v_lMd)
 FALSE  : exp(v - 0.5 * v_lMd)
```

```
emc <- make_emc(dat, design=design_RDM_lin_B3)
p_vector <- c('B'=1, 'v'=1, 'v_lMd'=1, 't0'=0.1, 'B.w'=1)

# visualize trend
plot_trend(p_vector, emc=emc,
           par_name='B', subject=1,
           lR_filter='odd', main='Threshold for odd')
```

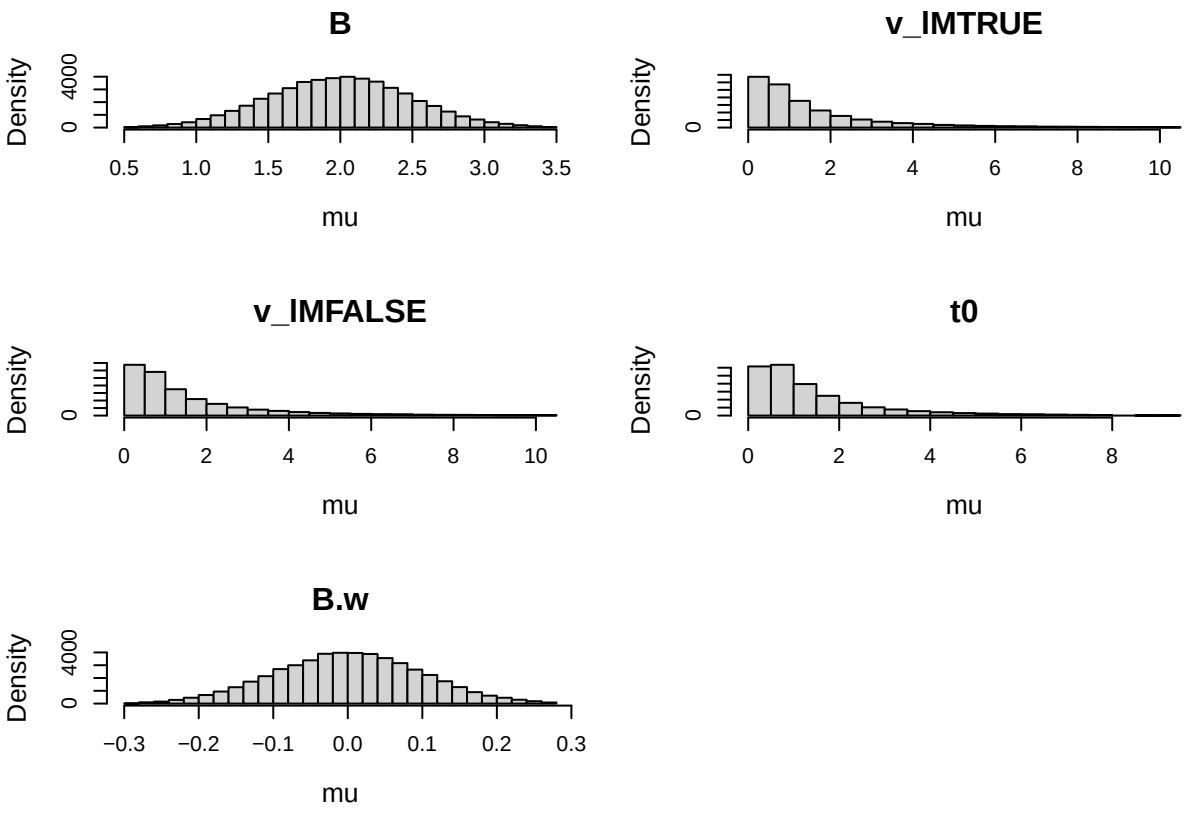


This leads to the expected effect. As a cautionary note, keep in mind that the default priors in *EMC2* are Gaussian distributions centered on 0 with unit variance. Since many cognitive model parameters cannot be negative, a N(0,1) prior is poorly chosen for those parameters that do not have support on the real line. For estimation and sampling, this usually has little influence in practice, but it may be important when estimating Bayes Factors.

With all that in mind, we can now start sampling our first model. We set a N(2,0.5) prior on the threshold B to reduce the prior density on negative thresholds. This is a somewhat subjective choice based on earlier experience, so it reflects my (but perhaps not your) prior belief. The rest of the priors are left to their default N(0,1) – on the scale on which they are sampled.

```
prior_linB <- prior(design_RDM_lin_B3, mu_mean=c(B=2, B.w=0), mu_sd=c(B=0.5,B.w=0.1))
emc <- make_emc(dat, design=design_RDM_lin_B3, prior_list = prior_linB)
```

```
plot(emc, prior=TRUE)
```

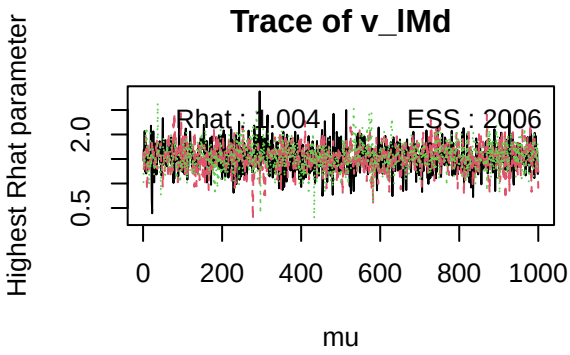


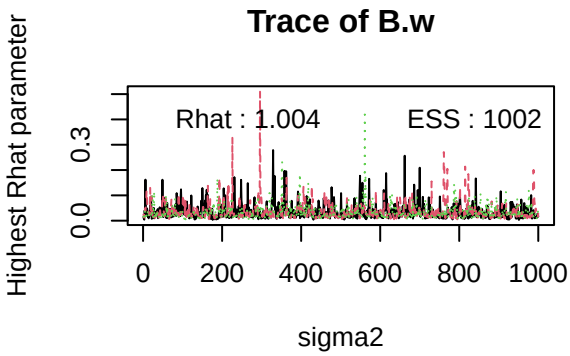

```
emc <- fit(emc, cores_per_chain=6, cores_for_chains=3,
                fileName='./samples/ds1_linB.RData')
```

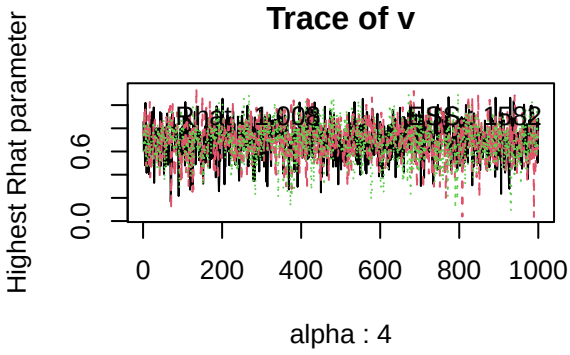
```
check(emc)
```

```
Iterations:
     preburn burn adapt sample
[1,]       0    0     0   1000
[2,]       0    0     0   1000
[3,]       0    0     0   1000

 mu
            B        v    v_lMd       t0      B.w
Rhat    1.001    1.002    1.004    1.001    1.001
ESS  2094.000 1943.000 2006.000 2070.000 2038.000

 sigma2
        B        v    v_lMd   t0      B.w
Rhat    1    1.002    1.002    1    1.004
ESS  2078 1682.000 1657.000 1270 1002.000

 alpha highest Rhat : 4
            B        v    v_lMd   t0      B.w
```

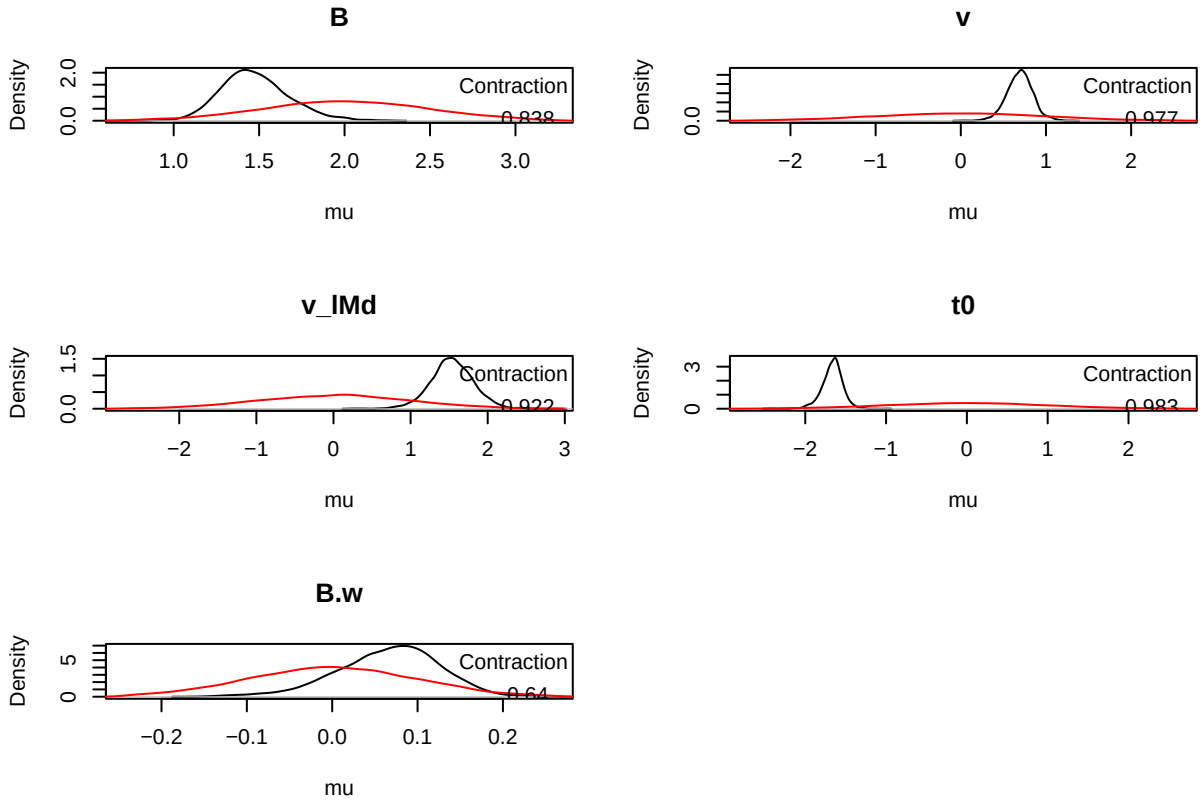
```
Rhat    1.001    1.008    1.003    1    1.002
ESS  2063.000 1582.000 1570.000 2401 2339.000
```








```
plot_pars(emc)
```

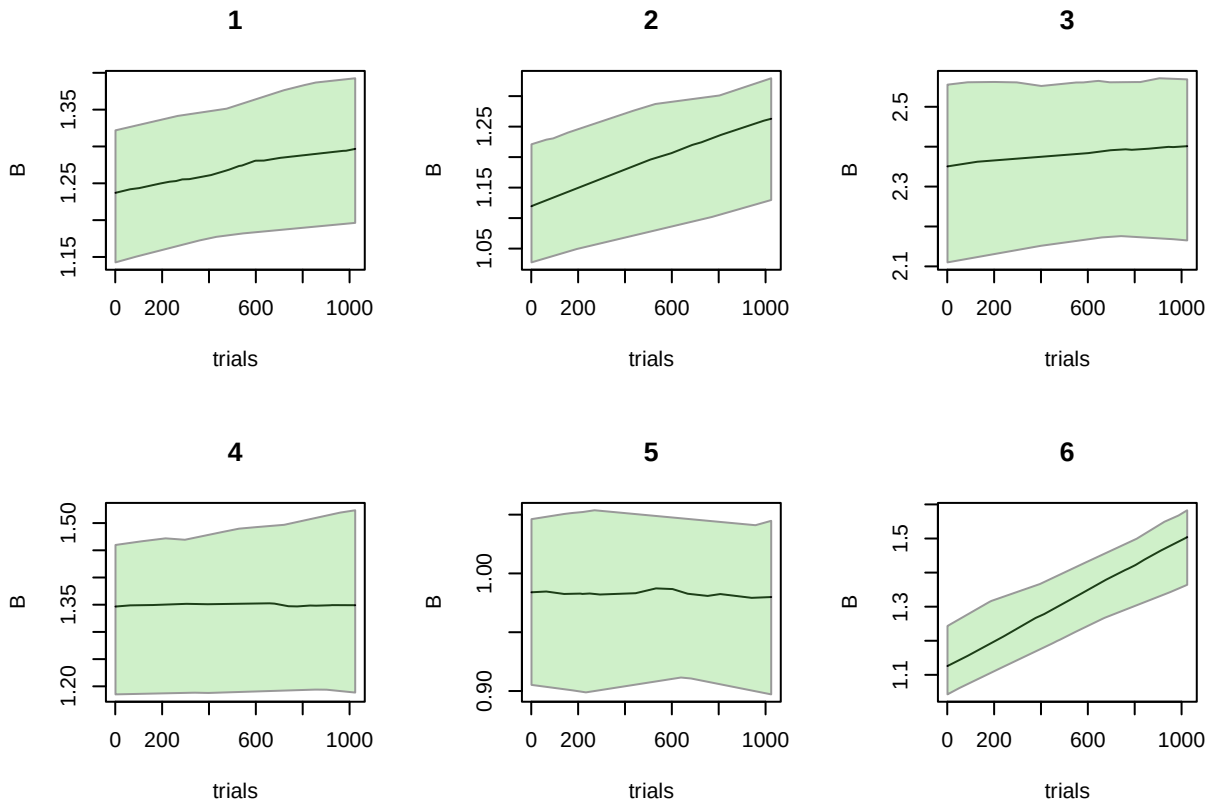



```
credint(emc)
```

```
$mu
         2.5%    50%  97.5%
B       1.122  1.459  1.913
v       0.389  0.696  1.001
v_lMd   0.978  1.536  2.088
t0     -1.951 -1.656 -1.428
B.w    -0.062  0.070  0.171
```

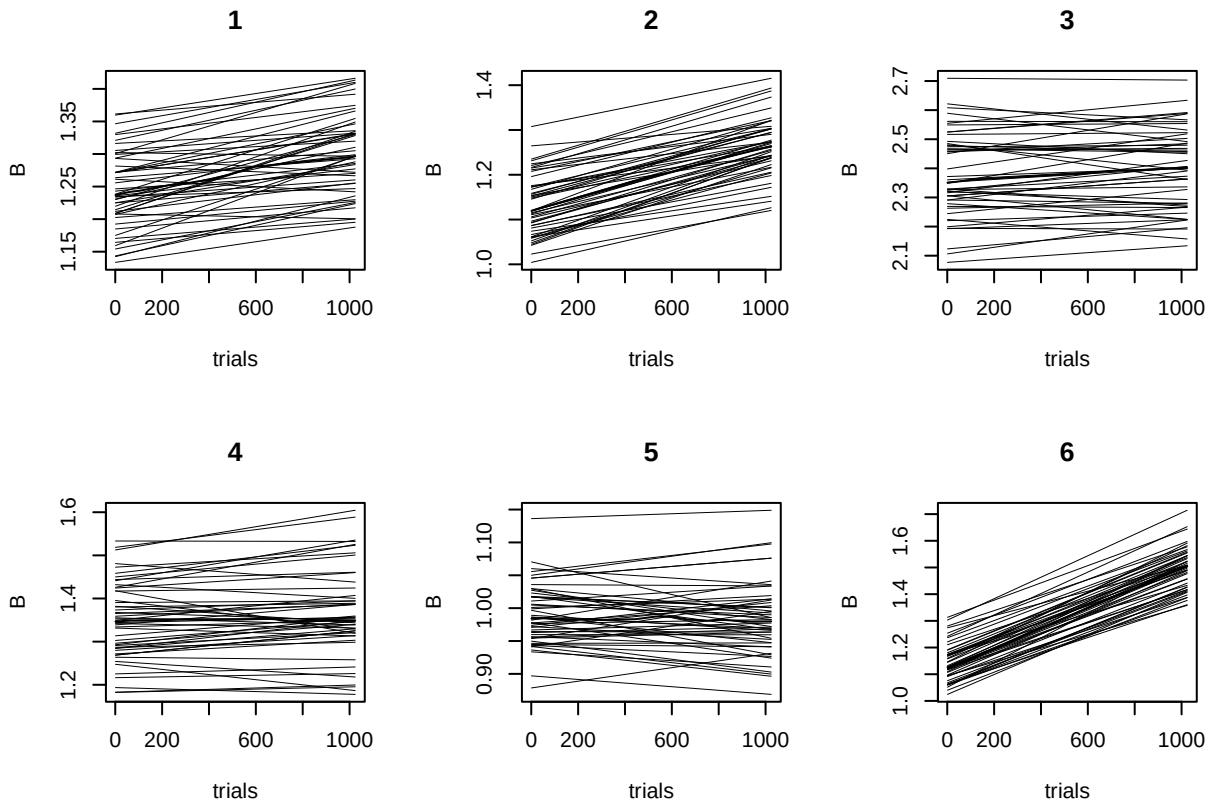
On a group-level, we find a small (not credible) positive `B.w` – on average, in this dataset, thresholds appear to increase by ~0.070 over the course of 1000 trials. We can now generate posterior predictives, and while doing that, ask *EMC2* to return the mapped and transformed parameters, which will allow us to visualize the fitted thresholds over trials:

```
# By setting return_trialwise_parameters to TRUE, we obtain the parameters
# of each iteration as an attribute in the returned object (pp).
pp <- predict(emc, n_cores=10, return_trialwise_parameters=TRUE)
par(mfrow=c(2,3))
for(subject in 1:6) {
  plot_trend(attr(pp, 'trialwise_parameters'), emc=emc,
             par_name='B', subject=subject, lR_filter='odd', main=subject)
}
```



Here, the green shaded area indicated the 95% credible interval of the posterior on the threshold for each trial. We can also plot the individual thresholds of each posterior predictive separately as lines:

```
par(mfrow=c(2,3))
for(subject in 1:6) {
  plot_trend(attr(pp, 'trialwise_parameters'), emc=emc,
             par_name='B', subject=subject, lR_filter='odd',
             main=subject, pp_shaded=FALSE)
}
```



Exponential and power laws

Practice effects tend to be large initially and then gradually decay. Asymptotic functions like exponential or power functions can be used to model such effects. *EMC2* offers increasing and decreasing kernels in both cases. To demonstrate these functional forms:

```
trend_exp_incr <- make_trend(par_names='B', cov_names = 'trials2',
                             kernels = 'exp_incr', bases = 'lin')
trend_exp_decr <- make_trend(par_names='B', cov_names = 'trials2',
                             kernels = 'exp_decr', bases = 'lin')
design_exp_incr <- design(model=RDM,
                          data=dat,
                          contrast=list(lM=ADmat),
                          covariates='trials2',
                          matchfun=function(d) d$S==d$lR,
                          transform=list(func=c('B'='identity')),
                          formula=list(B ~ 1, v ~ lM, t0 ~ 1),
                          trend=trend_exp_incr)
```

```
 Sampled Parameters:
[1] "B"      "v"      "v_lMd"  "t0"     "B.w"    "B.d_ei"

 Design Matrices:
$B
 B
```

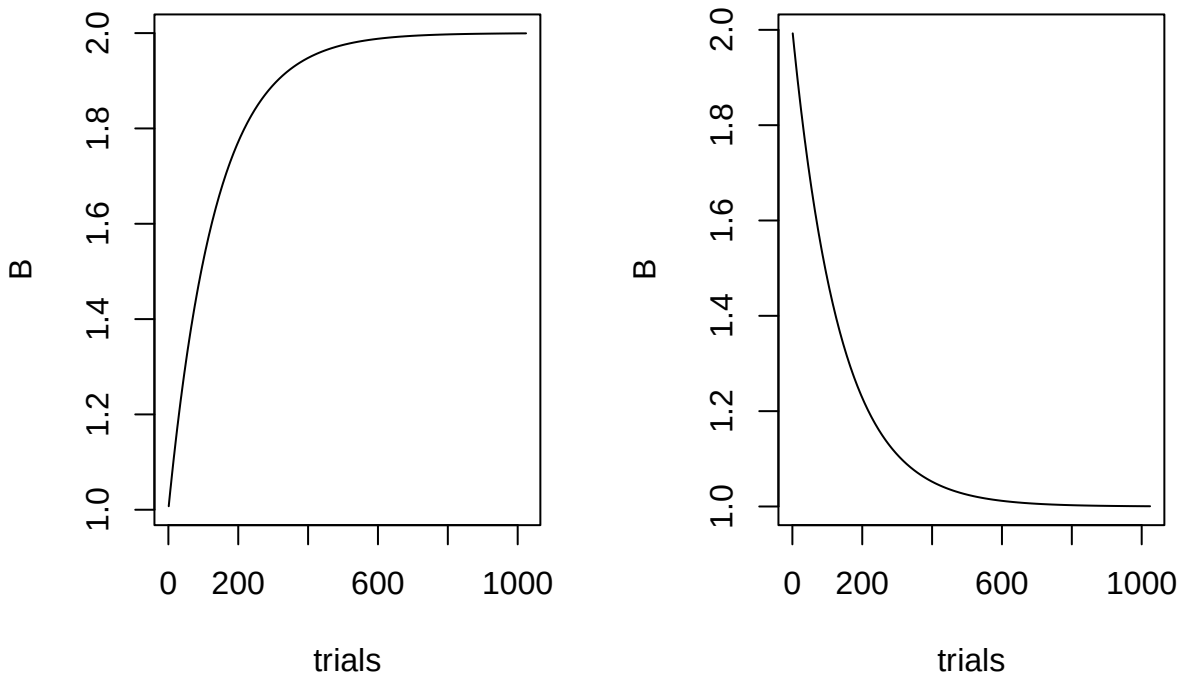
```
 1

$v
    lM v v_lMd
  TRUE 1   0.5
 FALSE 1  -0.5

$t0
 t0
  1

$B.w
 B.w
   1

$B.d_ei
 B.d_ei
      1

$A
 A
 1

$s
 s
 1
```

```
design_exp_decr <- design(model=RDM,
                          data=dat,
                          contrast=list(lM=ADmat),
                          covariates='trials2',
                          matchfun=function(d) d$S==d$lR,
                          transform=list(func=c('B'='identity')),
                          formula=list(B ~ 1, v ~ lM, t0 ~ 1),
                          trend=trend_exp_decr)
```

```
 Sampled Parameters:
[1] "B"      "v"      "v_lMd"  "t0"     "B.w"    "B.d_ed"

 Design Matrices:
$B
 B
 1

$v
    lM v v_lMd
  TRUE 1   0.5
 FALSE 1  -0.5

$t0
 t0
  1
```

```
$B.w
 B.w
   1

$B.d_ed
 B.d_ed
      1

$A
 A
 1

$s
 s
 1
```

```r
emc_incr <- make_emc(dat, design_exp_incr)
emc_decr <- make_emc(dat, design_exp_decr)

p_vector_incr <- c('B'=1, 'v'=1, 'v_lMd'=1, 't0'=0.1, 'B.w'=1, 'B.d_ei'=2)
p_vector_decr <- c('B'=1, 'v'=1, 'v_lMd'=1, 't0'=0.1, 'B.w'=1, 'B.d_ed'=2)

par(mfrow=c(1,2))
plot_trend(p_vector_incr, emc=emc_incr,
           par_name='B', subject=1, lR_filter='odd')
plot_trend(p_vector_decr, emc=emc_decr,
           par_name='B', subject=1, lR_filter='odd')
```



Note that the interpretation of the exponent (`B.e_ei` or `B.e_ed`) depends on the direction: In the increasing case, it corresponds to the asymptote; in the decreasing case, to the intercept.

Advanced note: Enforcing a direction

Having both increasing and decreasing kernels may appear redundant, since the direction of the effect can also be flipped by the `w` parameter in the `lin` base. However, having both increasing and decreasing kernels facilitates enforcing a directional effect. For example, one might hypothesize that `v_lMd` increases sharply over the first few trials due to practice effects, and then stabilizes. An increase could be implemented with `exp_incr`, but if the sampler converges on negative values of `w` of the base, the resulting trend is actually decreasing rather than increasing.

We can force the sampler to sample only increasing trends by restricting the range of `w`. This can be done by sampling `w` on the log-scale, and transforming it to the natural scale. Note that transformations applied to parameters relating to the trends are always applied prior to estimating the trend.

```
trend_exp_incr <- make_trend(par_names='v_lMd', cov_names = 'trials2',
                             kernels = 'exp_incr', bases = 'lin')
design_exp_incr <- design(model=RDM,
                          data=dat,
                          contrast=list(lM=ADmat),
                          covariates='trials2',
                          matchfun=function(d) d$S==d$lR,
                          transform=list(func=c('v'='identity',
                                                'v_lMd.w'='exp')),
                          formula=list(B ~ 1, v ~ lM, t0 ~ 1),
                          trend=trend_exp_incr)
```

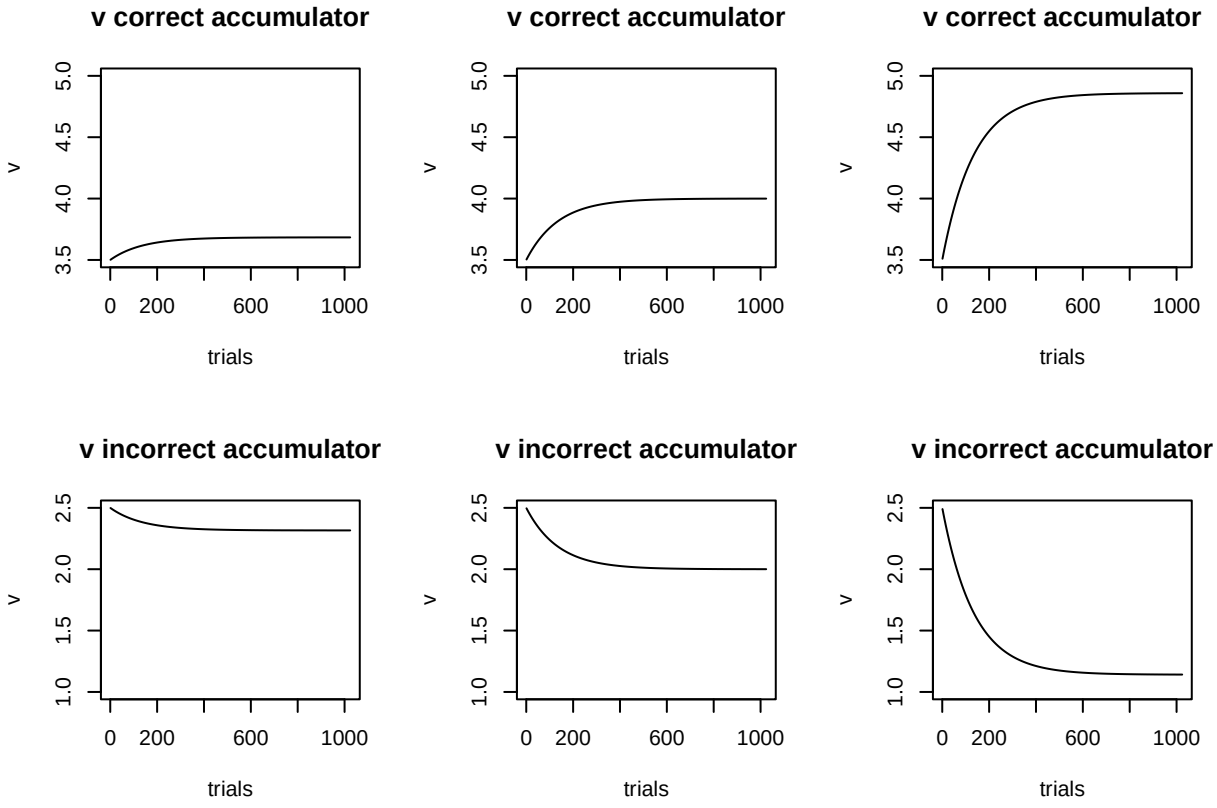
```
 Sampled Parameters:
[1] "B"          "v"          "v_lMd"      "t0"         "v_lMd.w"
[6] "v_lMd.d_ei"

 Design Matrices:
$B
 B
 1

$v
    lM v v_lMd
  TRUE 1   0.5
 FALSE 1  -0.5

$t0
 t0
  1

$v_lMd.w
 v_lMd.w
       1

$v_lMd.d_ei
 v_lMd.d_ei
          1

$A
 A
 1

$s
 s
 1
```

```
mapped_pars(design_exp_incr)
```

```
$v
  lM
 TRUE   : v + 0.5 * v_lMd_t
 FALSE  : v - 0.5 * v_lMd_t
 Trends:
 v_lMd_t = v_lMd + exp(v_lMd.w) * (1 - exp(-exp(v_lMd.d_ei) * trials2))
```

```
emc_incr <- make_emc(dat, design_exp_incr)
p_vector1 <- c('B'=1, 'v'=3, 'v_lMd'=1, 't0'=0.1, 'v_lMd.w'=-1, 'v_lMd.d_ei'=2)
p_vector2 <- c('B'=1, 'v'=3, 'v_lMd'=1, 't0'=0.1, 'v_lMd.w'=0, 'v_lMd.d_ei'=2)
p_vector3 <- c('B'=1, 'v'=3, 'v_lMd'=1, 't0'=0.1, 'v_lMd.w'=1, 'v_lMd.d_ei'=2)

par(mfcol=c(2,3))
plot_trend(p_vector1, emc=emc_incr, par_name='v',
           subject=1, lM_filter=TRUE, main='v correct accumulator', ylim=c(3.5,5))
plot_trend(p_vector1, emc=emc_incr, par_name='v',
           subject=1, lM_filter=FALSE, main='v incorrect accumulator', ylim=c(1, 2.5))
```

```
plot_trend(p_vector2, emc=emc_incr, par_name='v',
           subject=1, lM_filter=TRUE, main='v correct accumulator', ylim=c(3.5,5))
plot_trend(p_vector2, emc=emc_incr, par_name='v',
           subject=1, lM_filter=FALSE, main='v incorrect accumulator', ylim=c(1, 2.5))
plot_trend(p_vector3, emc=emc_incr, par_name='v',
           subject=1, lM_filter=TRUE, main='v correct accumulator', ylim=c(3.5,5))
plot_trend(p_vector3, emc=emc_incr, par_name='v',
           subject=1, lM_filter=FALSE, main='v incorrect accumulator', ylim=c(1, 2.5))
```



In this set-up, no matter the value of `v_lMd.w` the between-accumulator difference in drift rates increases over time. Using the same logic, we can also enforce a decrease by using `exp_decr` as a kernel and applying an exponential `transform` to `w`.

So far, we only inspected time on task as defined by trial number. Other options include time on task within block (e.g., short breaks between trials could cause an increased threshold for the first few trials in each block), or the number of times a stimulus type has been shown. Furthermore, the parameters describing the trend could differ between blocks and stimulus types. To test such hypotheses, we can allow the trend parameters themselves to vary with experimental factors, as defined in `formula` in `design()`. The example data does not have stimulus type recorded, and all trials were run in a single block, but for demonstrative purposes we can simulate such effects:

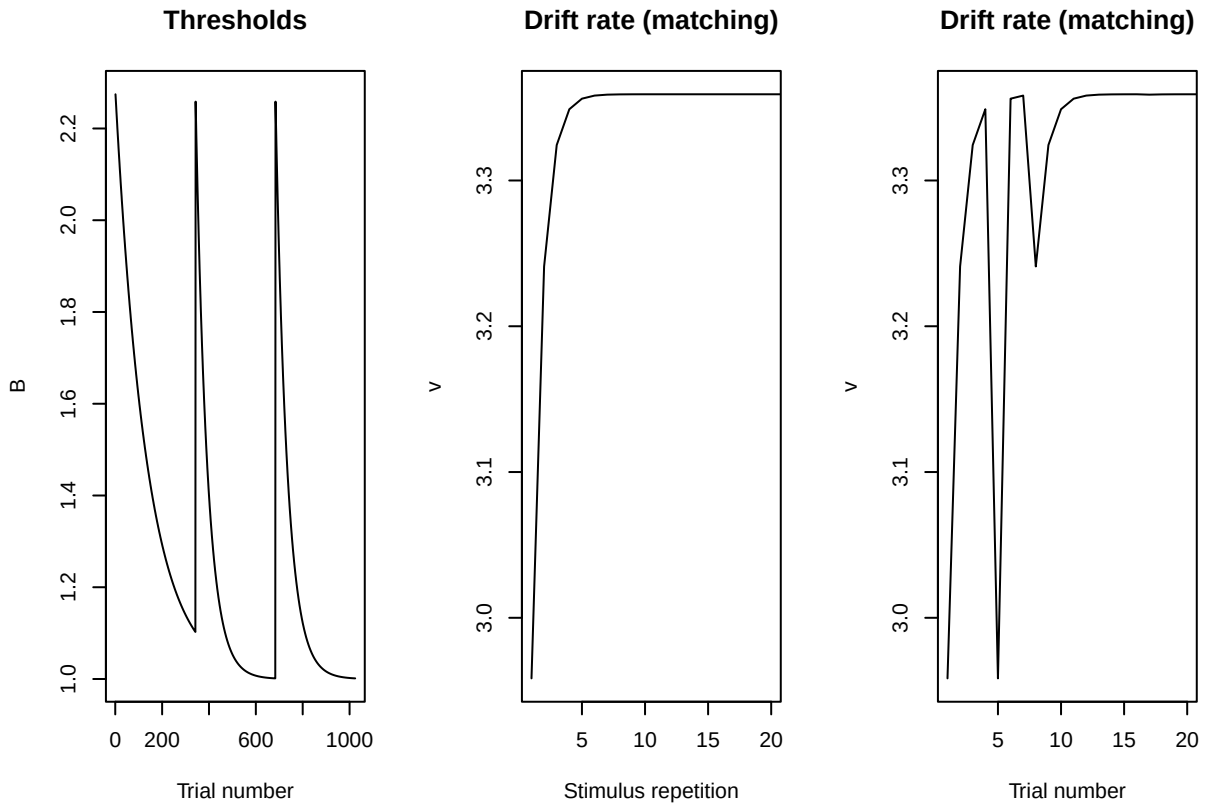
```
# simulate block number (assume 3 blocks)
dat$block <- as.factor(as.numeric(cut(dat$trials, breaks=3)))
for(subject in unique(dat$subjects)) {
  dat[dat$subjects==subject,'trial_in_block'] <- ave(
```

```
    seq_along(dat[dat$subjects==subject,'block']),
    dat[dat$subjects==subject,'block'], FUN = seq_along)/1000
}
# simulate presented digit
dat$stimulus_repetition <- ave(seq_along(dat$S),
                               dat$S, FUN = seq_along)

# combine two trends
trends <- make_trend(par_names=c('B', 'v_lMd'),
                     cov_names=c('trial_in_block', 'stimulus_repetition'),
                     kernels = c('exp_decr', 'exp_incr'),
                     bases = c('lin', 'lin'), )
design_multitrend <- design(model=RDM,
                         data=dat,
                         contrast=list(lM=ADmat),
                         covariates=c('trial_in_block', 'stimulus_repetition'),
                         matchfun=function(d) d$S==d$lR,
                         transform=list(func=c('v_lMd.w'='exp',
                                               'B.w'='exp',
                                               'v'='identity',
                                               'B'='identity')),
                         formula=list(B ~ 1, v ~ lM, t0 ~ 1, `B.d_ed`~block),
                         trend=trends)
```

```
 Sampled Parameters:
 [1] "B"             "v"             "v_lMd"         "t0"
 [5] "B.d_ed"        "B.d_ed_block2" "B.d_ed_block3" "B.w"
 [9] "v_lMd.w"       "v_lMd.d_ei"

 Design Matrices:
$B
 B
 1

$v
    lM v v_lMd
  TRUE 1   0.5
 FALSE 1  -0.5

$t0
 t0
  1

$B.d_ed
 block B.d_ed B.d_ed_block2 B.d_ed_block3
     1      1             0             0
     2      1             1             0
     3      1             0             1

$B.w
 B.w
   1

$v_lMd.w
```

```
 v_lMd.w
       1

$v_lMd.d_ei
 v_lMd.d_ei
          1

$A
 A
 1

$s
 s
 1
```

```
emc_multitrend <- make_emc(dat, design_multitrend)

# thresholds
p_vector <- c('B'=1, 'v'=1, 'v_lMd'=2, 't0'=0.1,
              'B.d_ed'=2, 'B.d_ed_block2'=1, 'B.d_ed_block3'=1, 'B.w'=.25,
              'v_lMd.w'=1, 'v_lMd.d_ei'=.2)

par(mfcol=c(1,3))
plot_trend(p_vector, emc=emc_multitrend, par_name='B',
           subject=1, lR_filter='odd', main='Thresholds',
           xlab='Trial number')
plot_trend(p_vector, emc=emc_multitrend, par_name='v',
           subject=1, lM_filter=TRUE, main='Drift rate (matching)',
           on_x_axis='stimulus_repetition',
           xlab='Stimulus repetition', xlim=c(1,20))
plot_trend(p_vector, emc=emc_multitrend, par_name='v',
           subject=1, lM_filter=TRUE, main='Drift rate (matching)',
           xlab='Trial number', xlim=c(1,20))
```




```
mapped_pars(design_multitrend)
```

```
$B

 intercept  : B_t
 Trends:
 B_t = B + exp(B.w) * exp(-exp(B.d_ed) * trial_in_block)

$v
  lM
 TRUE   : v + 0.5 * v_lMd_t
 FALSE  : v - 0.5 * v_lMd_t
 Trends:
 v_lMd_t = v_lMd + exp(v_lMd.w) * (1 - exp(-exp(v_lMd.d_ei) * stimulus_repetition))

$B.d_ed
  block
 1      : exp(B.d_ed)
 2      : exp(B.d_ed + B.d_ed_block2)
 3      : exp(B.d_ed + B.d_ed_block3)
```

The left panel illustrate how the thresholds decrease within each block (at different rates), but then reset to the same asymptote after every break. The middle panel shows the drift rate for the correct accumulator as a function of how often a stimulus has been presented. Note that, when plotted against trial number (right panel), this leads to strong variability in rates across trials.

Polynomials

When a researcher has no strong prior assumptions about the shape of the trend but wishes to capture gradual changes, it is possible to fit a set of basis functions, such as polynomials. EMC2 currently includes second-, third-, and fourth-order polynomials as available kernels. This approach follows the logic of a Taylor expansion: any function can be approximated with increasing accuracy by summing polynomials of increasing order. However, the increased accuracy comes at a cost of increased sampling uncertainty, since more parameters need to be estimated. As such, it may be possible to estimate only a few polynomials.

```
trend_help(kernel='poly4')
```

```
Description:
Quartic polynomial: k = d1 * c + d2 * c^2 + d3 * c^3 + d4 * c^4

Default transformations (in order):
list(d1 = "identity", d2 = "identity", d3 = "identity", d4 = "identity")

Available bases, first is the default:
add, identity
```

Note that the kernel parameters (`d1, d2, d3, d4`) already weigh the contributions of the individual polynomial terms. As such, this kernel should not be combined with a linear base (`base = 'lin'`), but instead with an additive base (`base = 'add'`). Additionally, we should center the Taylor expansion around the mean of the covariate. The following demonstrates the flexibility of a 4th order polynomial with a few randomly chosen parameter sets:

```
dat$tctrd <- dat$trials2-mean(dat$trials2)
trend_poly <- make_trend(par_names='B',
                     cov_names='tctrd',
                     kernels='poly4',
                     bases='add')
design_poly <- design(model=RDM,
                          data=dat,
                          contrast=list(lM=ADmat),
                          covariates='tctrd',
                          matchfun=function(d) d$S==d$lR,
                          transform=list(func=c('B'='identity')),
                          formula=list(B ~ 1, v ~ lM, t0 ~ 1),
                          trend=trend_poly)
```

```
 Sampled Parameters:
[1] "B"     "v"     "v_lMd" "t0"    "B.d1"  "B.d2"  "B.d3"  "B.d4"

 Design Matrices:
$B
 B
 1

$v
    lM v v_lMd
  TRUE 1   0.5
 FALSE 1  -0.5
```

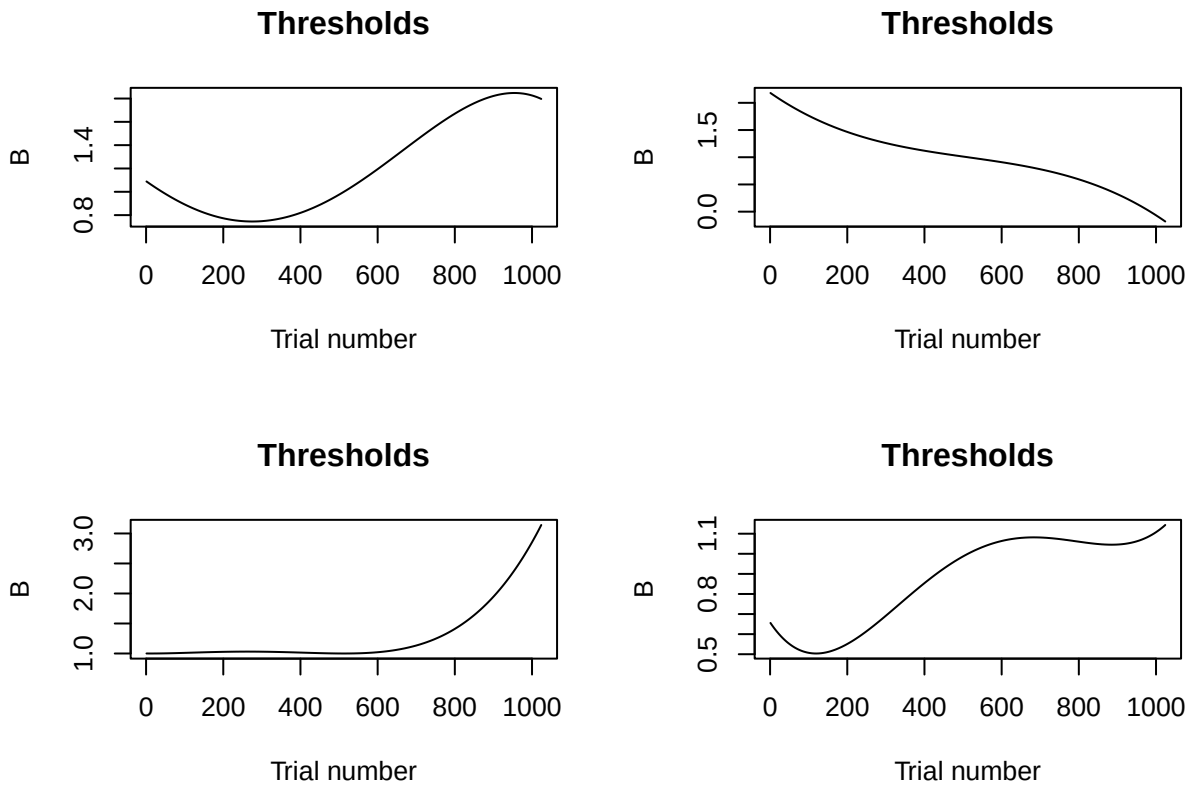
```
$t0
 t0
  1

$B.d1
 B.d1
    1

$B.d2
 B.d2
    1

$B.d3
 B.d3
    1

$B.d4
 B.d4
    1

$A
 A
 1

$s
 s
 1
```

```
mapped_pars(design_poly)
```

```
$B

 intercept  : B_t
 Trends:
 B_t = B + (B.d1 * tctrd + B.d2 * tctrd^2 + B.d3 * tctrd^3 + B.d4 * tctrd^4)

$v
  lM
 TRUE   : exp(v + 0.5 * v_lMd)
 FALSE  : exp(v - 0.5 * v_lMd)
```

```
emc_poly <- make_emc(dat, design_poly)

p_vector1 <- c('B'=1, 'v'=1, 'v_lMd'=2, 't0'=0.1,
               'B.d1'=2, 'B.d2'=3, 'B.d3'=-5, 'B.d4'=-5)
p_vector2 <- c('B'=1, 'v'=1, 'v_lMd'=2, 't0'=0.1,
               'B.d1'=-1, 'B.d2'=0, 'B.d3'=-5, 'B.d4'=0)
p_vector3 <- c('B'=1, 'v'=1, 'v_lMd'=2, 't0'=0.1,
               'B.d1'=0, 'B.d2'=2, 'B.d3'=8, 'B.d4'=8)
p_vector4 <- c('B'=1, 'v'=1, 'v_lMd'=2, 't0'=0.1,
               'B.d1'=1, 'B.d2'=-3, 'B.d3'=-2, 'B.d4'=10)
```

```r
par(mfrow=c(2,2))
plot_trend(p_vector1, emc=emc_poly, par_name='B',
           subject=1, lR_filter='odd', main='Thresholds',
           xlab='Trial number')
plot_trend(p_vector2, emc=emc_poly, par_name='B',
           subject=1, lR_filter='odd', main='Thresholds',
           xlab='Trial number')
plot_trend(p_vector3, emc=emc_poly, par_name='B',
           subject=1, lR_filter='odd', main='Thresholds',
           xlab='Trial number')
plot_trend(p_vector4, emc=emc_poly, par_name='B',
           subject=1, lR_filter='odd', main='Thresholds',
           xlab='Trial number')
```



Mechanisms of dynamics: Stimulus memory

Models with formal mechanisms of dynamics rely on delta rules. To implement delta rules, we follow the same general logic as above—the delta rule is implemented as a kernel in `make_trend`. Let's implement the stimulus memory mechanism proposed by Miletic et al. (2025). Here, the decision maker keeps track of the probability that a stimulus is even (or odd). The covariate in this case is the stimulus, which we can recode as 1 (even) and -1 (odd). With the delta rule:

$$Q_{SM,t+1} = Q_{SM,t} + \alpha_{SM} \cdot (S_t - Q_{SM,t}), \quad S \in \{-1, 1\}$$

The delta rule introduces two extra parameters: a learning rate α_{SM}, and a value for Q at the start of the experiment $Q_{SM,0}$. The latter is hard to estimate, and we tend to fix it to a constant—for this rule, a constant of 0 implies equal beliefs in both stimuli.

For now, let us assume that stimulus memory influences the relative thresholds: the even threshold increases, and the odd threshold decreases (by the same amount). To achieve this, we need to parameterize thresholds with the same mean-difference parameterization as we used for drift rates. That is,

$$B_{odd} = B_{mean} + B_{lRd}$$
$$B_{even} = B_{mean} - B_{lRd}$$

The B_{lRd} term is the parameter influenced by the updated covariate—however, since we do not necessarily expect an across-trial *mean* difference between thresholds, we set B_{lRd} as a constant to 0.

Using a linear base, we then allow the threshold difference to vary on a trial-by-trial basis with

$$B_{lRd,t} = B_{lRd} + w_{SM} \cdot Q_{SM,t}$$

Note the extra parameter here: w_{SM}.

```
dat$Stim1 <- ifelse(dat$S=='even', 1, -1)
SM_trend <- make_trend(cov_names=c('Stim1'),
                       kernels = 'delta',
                       par_names='B_lRd',
                       bases='lin',
                       premap=TRUE,
                       filter_lR=TRUE)
        # see advanced notes for an explanation filter_lR
design_RDM_SM <- design(model=RDM,
                        data=dat,
                        contrast=list(lM=ADmat, lR=ADmat),
                        covariates='Stim1',
                        matchfun=function(d) d$S==d$lR,
                        formula=list(B ~ lR, v ~ lM, t0 ~ 1),
                        trend=SM_trend,
                        constants=c('B_lRd'=0, 'B_lRd.q0'=0))
```

```
 Sampled Parameters:
[1] "B"           "v"           "v_lMd"       "t0"          "B_lRd.w"
[6] "B_lRd.alpha"

 Design Matrices:
$B
   lR B B_lRd
 even 1  -0.5
  odd 1   0.5

$v
    lM v v_lMd
  TRUE 1   0.5
 FALSE 1  -0.5

$t0
```

```
 t0
  1

$B_lRd.w
 B_lRd.w
       1

$B_lRd.q0
 B_lRd.q0
        1

$B_lRd.alpha
 B_lRd.alpha
           1

$A
 A
 1

$s
 s
 1
```

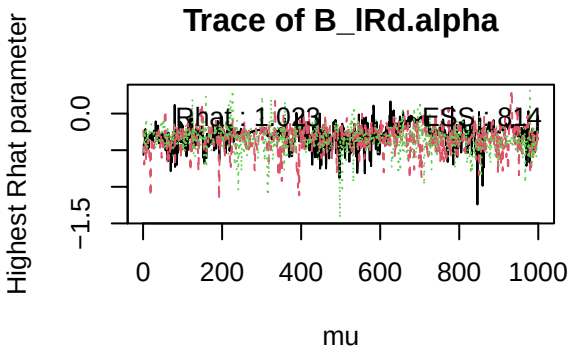
We can now apply a similar prior to B as above, and fit the model. Note that data compression is turned off automatically, since the delta rule needs to be applied to the covariate on each trial individually.

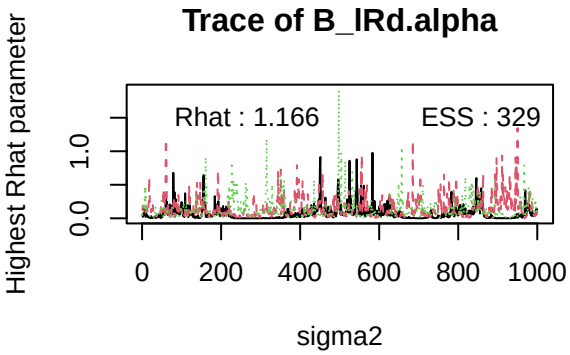
```
prior_SMB <- prior(design_RDM_SM, mu_mean=c(B=2), mu_sd=c(B=0.5))
emc <- make_emc(dat, design=design_RDM_SM, prior_list = prior_SMB, compress=FALSE)
```

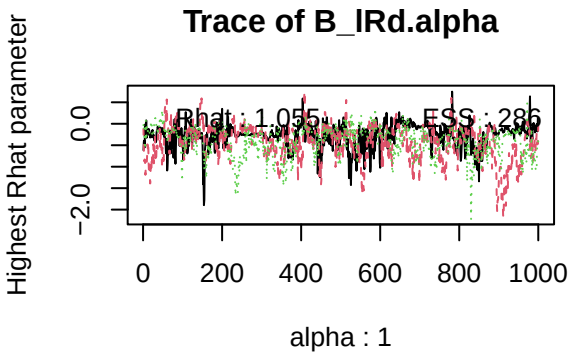
```
emc <- fit(emc, cores_per_chain=6, cores_for_chains=3,
               fileName='./samples/ds1_SMB.RData')
```

```
check(emc)
```

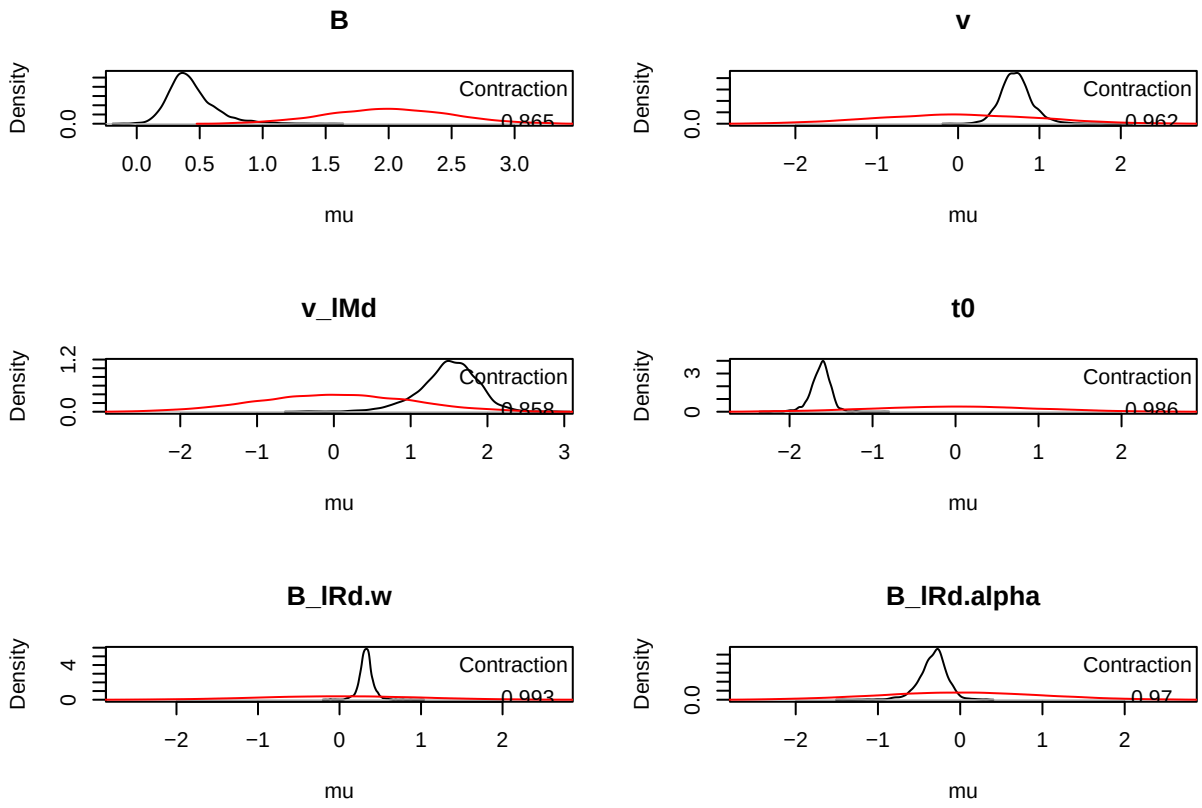
```
Iterations:
     preburn burn adapt sample
[1,]       0    0     0   1000
[2,]       0    0     0   1000
[3,]       0    0     0   1000

 mu
            B        v    v_lMd       t0 B_lRd.w B_lRd.alpha
Rhat    1.006    1.004    1.005    1.003   1.008       1.023
ESS  1220.000 1747.000 1668.000 1664.000 962.000     814.000

 sigma2
          B        v    v_lMd       t0 B_lRd.w B_lRd.alpha
Rhat   1.005    1.007    1.005    1.004   1.028       1.166
ESS  942.000 1265.000 1401.000 1553.000 795.000     329.000

 alpha highest Rhat : 1
            B       v   v_lMd       t0 B_lRd.w B_lRd.alpha
Rhat    1.004   1.023   1.019    1.003   1.052       1.055
ESS  1543.000 806.000 812.000 1863.000 247.000     286.000
```








```
plot_pars(emc)
```



We find excellent convergence properties. Note that learning rates are by default sampled on the probit scale, so if we want to know the mean learning rate, we need to map and transform the parameters first:

```
credint(emc, map=TRUE)
```

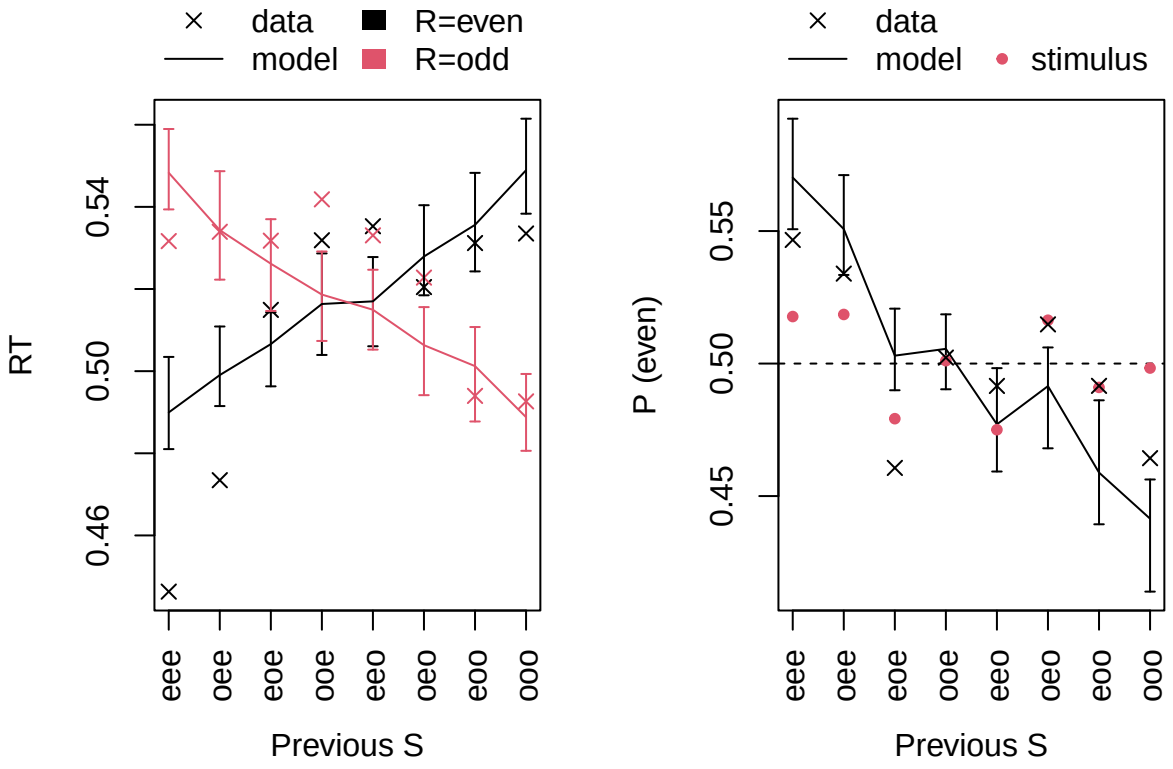
```
$mu
             2.5%   50% 97.5%
B_lReven    1.162 1.495 2.445
B_lRodd     1.162 1.495 2.445
v_lMTRUE    3.420 4.333 5.250
v_lMFALSE   0.499 0.947 2.226
t0          0.151 0.199 0.241
B_lRd.w     0.153 0.325 0.495
B_lRd.alpha 0.237 0.377 0.489
```

Hence, we find a group-wise mean learning rate of ~0.38, and the threshold of the odd accumulator is 0.323 lower than the threshold for the even accumulator if the participant has a Q_{SM}-value of 1.

We can now check whether the SM mechanism can explain stimulus-history effects by plotting the RTs (by choice) and the probability of choosing left as a function of the previous three presented stimuli. First, generate posterior predictives:

```
pp <- predict(emc, n_cores=10,
              # for plotting purposes below, these arguments are added:
              return_trialwise_parameters=TRUE, return_covariates=TRUE)
save(pp, file='./ds1_SMB_pps.RData')
```

```
load('./ds1_SMB_pps.RData')
plot_history_effects(dat, pp, n_hist=3)
```



The left panel of this plot shows the mean RT as a function of the three previous trials' stimuli (ooo = odd, odd, odd; ooe = odd, odd, even; etc. Note that the right-most letter in the string indicates the most recent trial). The data show a cross-over pattern, such that if the previous three stimuli were odd, 'odd' responses are faster than 'even' responses; and vice versa. The model tends to capture this cross-over pattern fairly well, with some misfits mostly due to an apparent asymmetry in the cross-over in the data.

The right panel shows the probability of choosing even (black) and the probability of the stimulus being even (red) as a function of local trial history. The black crosses demonstrates that participants are more likely (~.55) to choose 'even' if the previous three trials were 'even'. The red circles are important as a quality check. If the stimuli are generated truly randomly, the red circles should be approximately at 0.5 in all cases. However, some experiments use pseudorandomization, which can lead to negative correlations in local stimulus histories. This can confound any stimulus history effects present in the human data. True stimulus history effects are those that are larger than the ones present in the data, e.g., visible as black crosses deviating stronger from 0.5 compared to red circles.

Advanced note: filter_lR

Under the hood, EMC2 works with data-augmented design matrices (`dadms`). A `dadm` contains one row per accumulator per trial – so for race models applied to two-alternative forced choice tasks, this will typically result in two rows per trial. We can inspect the `dadm` for the first subject in the emc object to see that the covariate `Stim1` is indeed the same across accumulators in each trial:

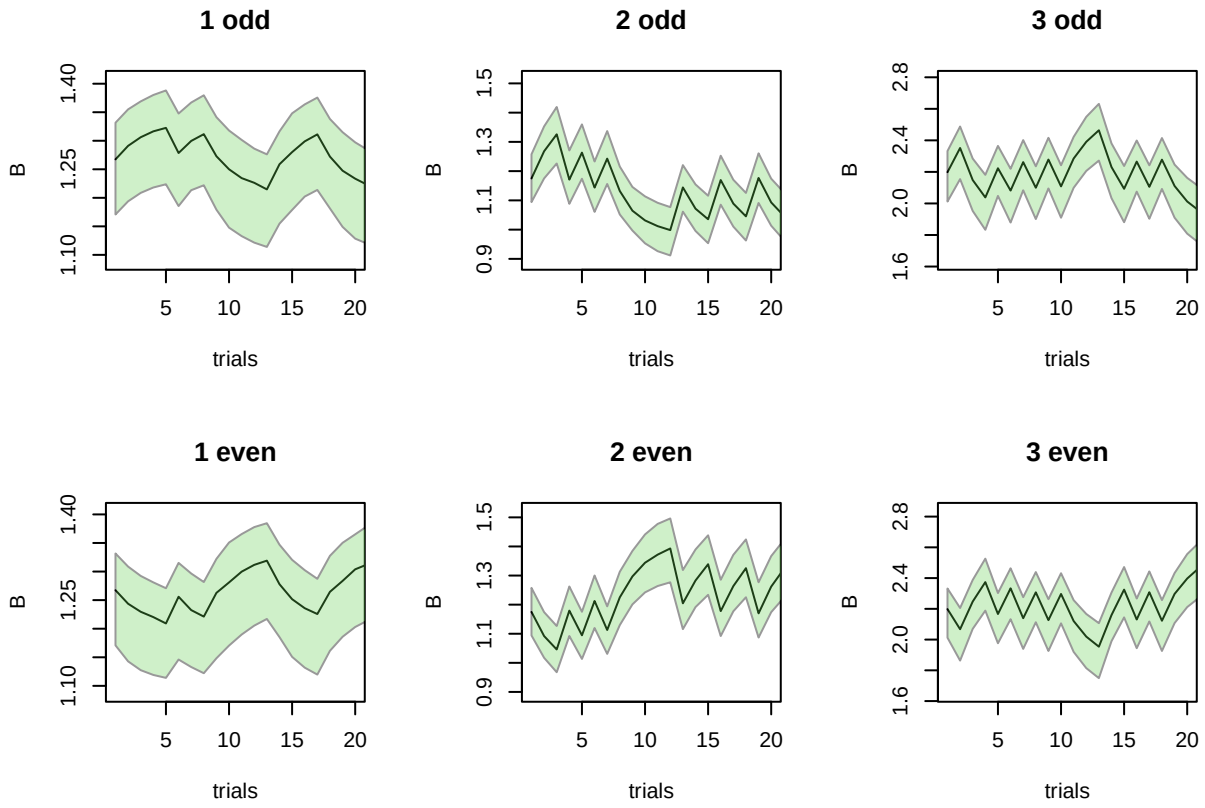
```
head(emc[[1]]$data[[1]])
```

```
  subjects    S    R   rt trials trials2 block trial_in_block
1        1 even even 0.54      1   0.001     1          0.001
2        1 even even 0.54      1   0.001     1          0.001
3        1 even even 0.42      2   0.002     1          0.002
4        1 even even 0.42      2   0.002     1          0.002
5        1 even even 0.50      3   0.003     1          0.003
6        1 even even 0.50      3   0.003     1          0.003
  stimulus_repetition   tctrd Stim1   lR    lM winner Stim1_lRfiltered
1                   1 -0.5115     1 even  TRUE   TRUE                1
2                   1 -0.5115     1  odd FALSE  FALSE               NA
3                   2 -0.5105     1 even  TRUE   TRUE                1
4                   2 -0.5105     1  odd FALSE  FALSE               NA
5                   3 -0.5095     1 even  TRUE   TRUE                1
6                   3 -0.5095     1  odd FALSE  FALSE               NA
```

The delta rule is applied to the covariate as it is specified in the `dadm`. Since, in the examples in this tutorial, the covariate is the same for both accumulators, blindly applying a delta rule to the covariate in the `dadm` would lead to *two* updates every trial: One for the first accumulator, and then another one for the second accumulator. The Q-values would then also differ between accumulators. The correct behavior is to update only once every trial (i.e., for the first accumulator), and then feed forward the updated Q-value to every second (and third, fourth, ...) accumulator. This can be done by setting the covariate for every second, third, fourth, etc accumulator to NA, since NA-values are ignored when updating. `filter_lR=TRUE` does this: it creates a new column `covariate_lRfiltered`, and applies the delta rule to that column, feeding forward updated Q-values when encountering an NA-value.
In many applications when using dynamic EAMs, this is likely the intended behavior. However, there are two exceptions: 1) The DDM only has one accumulator per trial, so no filtering has to be done (or can be done); and 2) when using RL-EAMs, applied to instrumental learning tasks, the covariate can differ between the two accumulators (feedback can be given to one specific option, for example). In such cases, more advanced handling of the covariate is required, and `filter_lR` should not be TRUE.

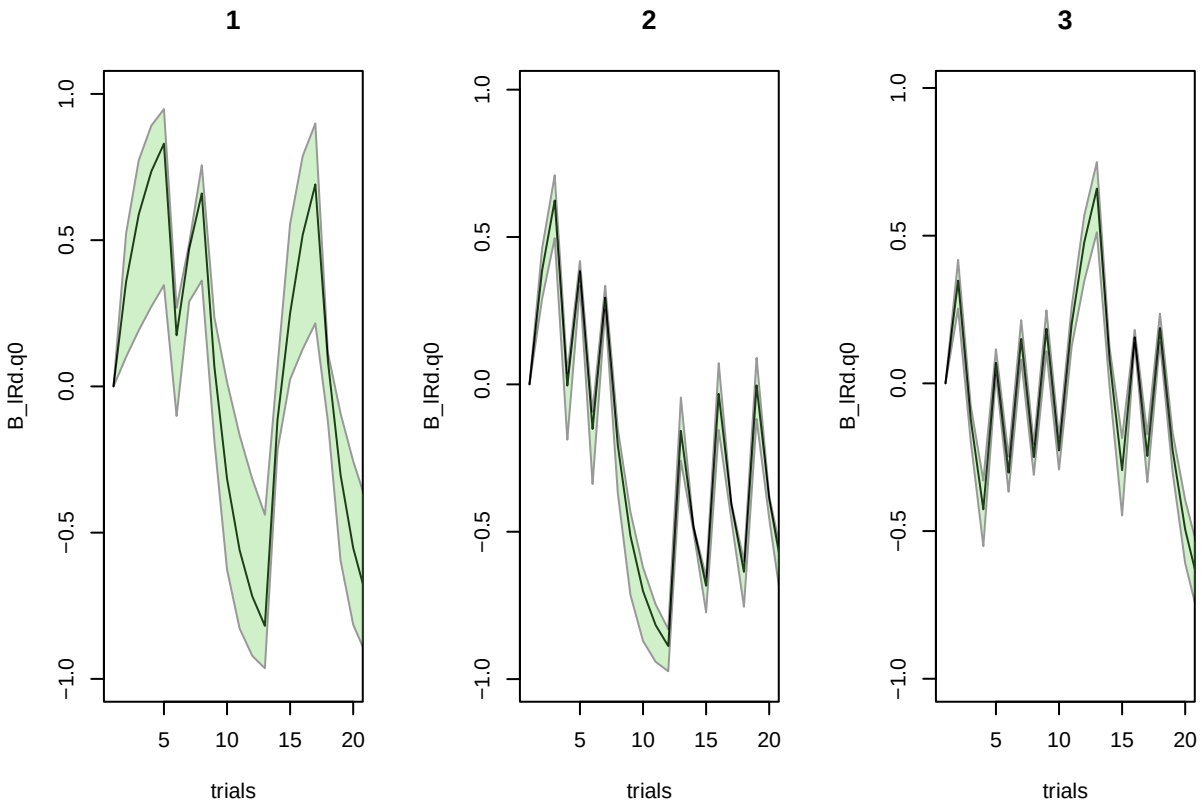
As before, we can plot the trial-wise parameters with `plot_trend()`. Let's zoom in on the even and odd thresholds of the first three subject for the first 20 trials:

```
par(mfcol=c(2,3))
for(subject in 1:3) {
  plot_trend(attr(pp, 'trialwise_parameters'), emc=emc,
             par_name='B', subject=subject, lR_filter='odd', main=paste0(subject, ' odd'),
             xlim=c(1,20))
  plot_trend(attr(pp, 'trialwise_parameters'), emc=emc,
             par_name='B', subject=subject, lR_filter='even', main=paste0(subject, ' even'),
             xlim=c(1,20))
}
```



Alternatively, we can plot the evolution of the Q-values themselves. Note that `predict()` was run with another argument: `return_covariates`, which returns the updated covariates as an attribute. We can use this as follows:

```
par(mfcol=c(1,3))
for(subject in 1:3) {
  plot_trend(attr(pp, 'covariates'), emc=emc,
             par_name='B_lRd.q0', subject=subject,
             main=subject,
             xlim=c(1,20))
}
```



Repetition/alternation memory

The stimulus-memory mechanism accounts for what Jones et al. (2014) termed first-degree recency effects: They depend on stimulus identity (e.g., 'odd' or 'even'). There are also second-degree recency effects, which depend on sequential properties - i.e., whether stimuli are repetitions or alternations. It is easy to implement such a mechanism with a delta rule as well. If we code the stimuli as $S_t \in -1, 1$, then multiplying S_t with S_{t-1} tells us whether S_t repeats (outcome 1) or alternates (outcome -1) the previous stimulus. Let's hypothesize that participants keep track of the repetition rate using a repetition memory (RM). That could make use ofthe delta rule:

$$Q_{RM,t+1} = Q_{RM,t} + \alpha_{RM} \cdot (S_{t-1}S_t - Q_{RM,t}), \quad S \in \{-1, 1\}$$

Q_{RM} is a latent representation for the belief that the stimulus will repeat itself. We could hypothesize that, if participants believe that the stimulus is repeating, the threshold corresponding the choice option that repeats the previous stimulus is lower compared to the one that alternates - and vice versa. To implement this, we need to parametrise the thresholds as repeat/alternate:

$$B_{repeat} = B_{mean} + B_{lR2d}$$
$$B_{alternate} = B_{mean} - B_{lR2d}$$

The B_{lR2d} term is the parameter influenced by the updated covariate—however, since we do not necessarily expect an across-trial *mean* difference between thresholds, we set B_{lR2d} as a constant to 0.

Using a linear base, we then allow the threshold difference to vary on a trial-by-trial basis with

$$B_{lR2d,t} = B_{lR2d} + w_{RM} \cdot Q_{RM,t}$$

Note the extra parameter here: w_{RM}.

Let's implement this step by step. First, let's add $S_{t-1}S_t$ to the data, as `dat$Stim2`. This will be our new covariate. It should influence `B_lR2d`. `lR2` can be created by `EMC2` by passing a function.

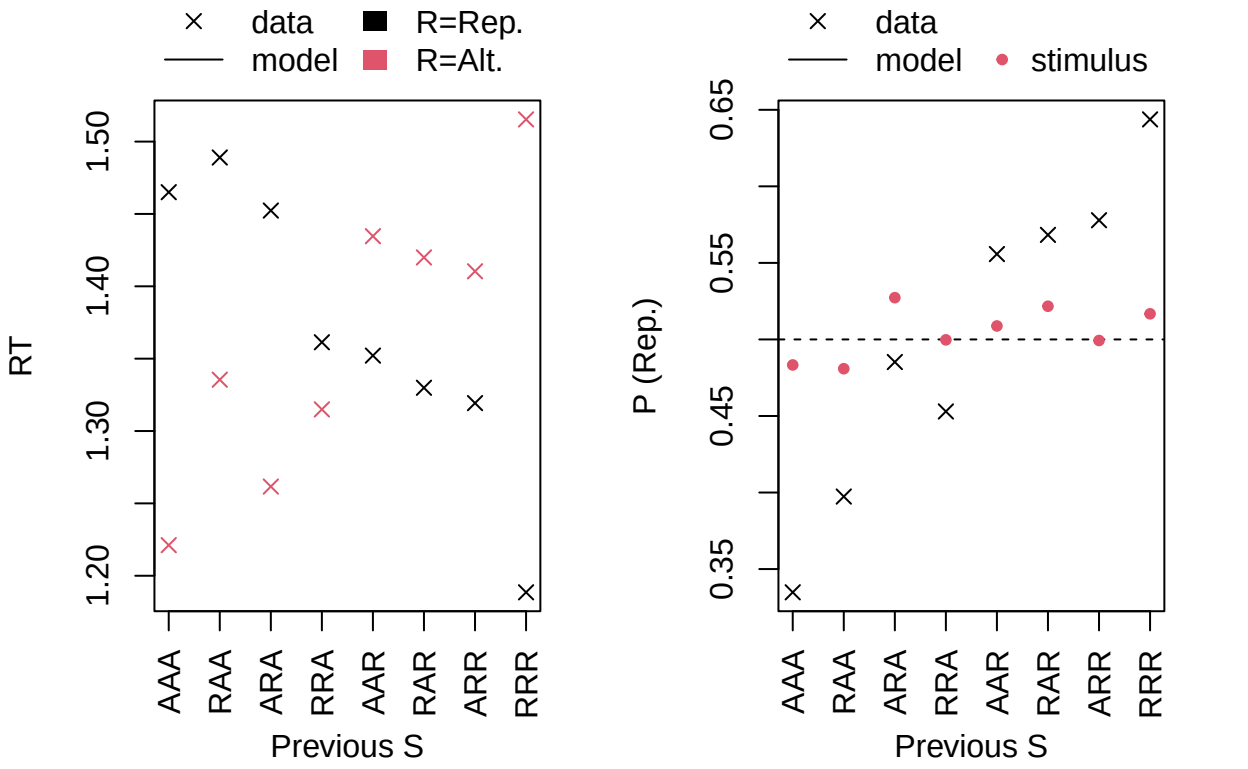
```
dat$Stim1 <- ifelse(dat$S=='even', 1, -1)
dat$Stim2 <- dat$Stim1*Hmisc::Lag(dat$Stim1,1)
RM_trend <- make_trend(cov_names=c('Stim2'),
                       kernels = 'delta',
                       par_names='B_lR2d',
                       bases='lin',
                       premap=TRUE,
                       filter_lR=TRUE)
ADmat <- matrix(c(-.5,.5), ncol=1, dimnames=list(NULL,'d'))

design_RDM_RM <- design(model=RDM,
                        data=dat,
                        contrast=list(lM=ADmat, lR=ADmat, lR2=ADmat),
                        covariates='Stim2',
                        functions=list(lR2=function(x) {
                          lR2 <- as.numeric(x$lR)==as.numeric(Hmisc::Lag(x$S,2))
                          lR2[is.na(lR2)] <- TRUE
                          factor(lR2, levels=c(1,0), labels=c('rep', 'alt'))
                        }),
                        matchfun=function(d) d$S==d$lR,
                        formula=list(B ~ lR2, v ~ lM, t0 ~ 1),
                        trend=RM_trend,
                        constants=c('B_lR2d'=0, 'B_lR2d.q0'=0))
```

```
 Sampled Parameters:
[1] "B"            "v"            "v_lMd"        "t0"           "B_lR2d.w"
[6] "B_lR2d.alpha"

 Design Matrices:
$B
 lR2 B B_lR2d
 rep 1   -0.5
 alt 1    0.5

$v
    lM v v_lMd
  TRUE 1   0.5
 FALSE 1  -0.5

$t0
 t0
  1

$B_lR2d.w
 B_lR2d.w
        1
```

```
$B_lR2d.q0
 B_lR2d.q0
         1

$B_lR2d.alpha
 B_lR2d.alpha
            1

$A
 A
 1

$s
 s
 1
```

Before fitting, we might want to simulate some data to test whether simulated data indeed shows second-degree recency effects.

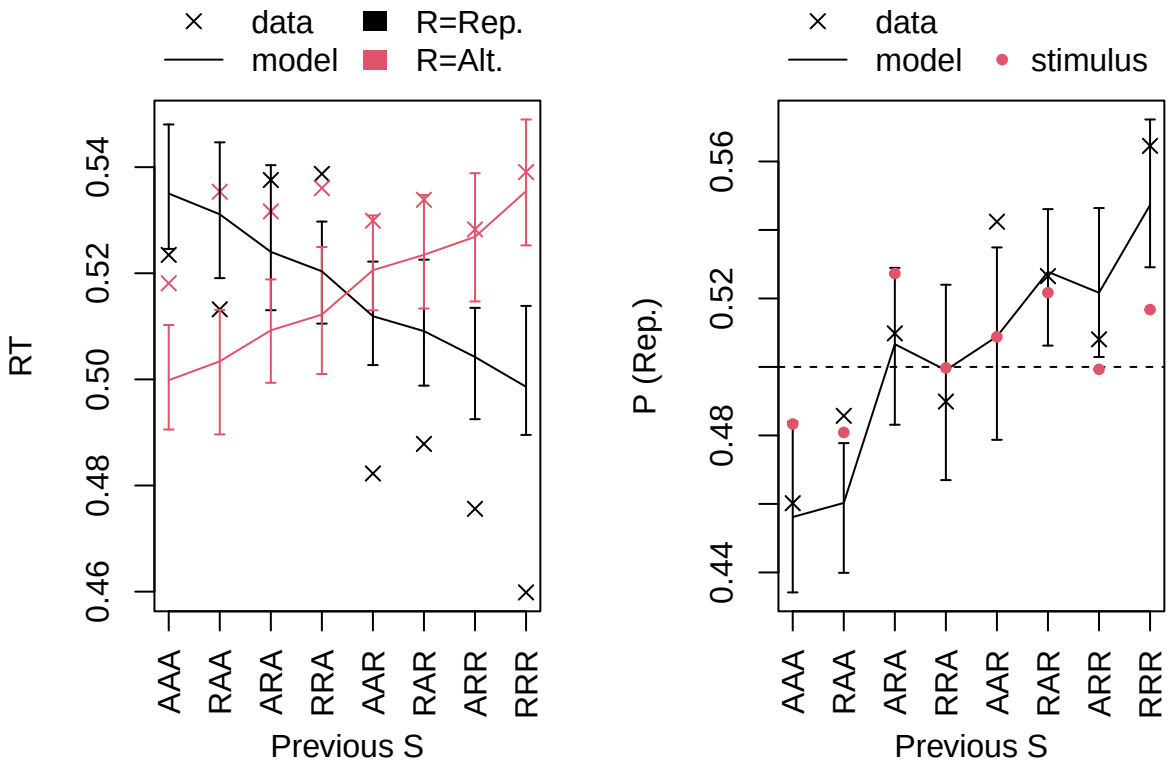
```
p_vector <- sampled_pars(design_RDM_RM)
p_vector[1:6] <- c(log(2), log(1), .5, log(.15), 2, qnorm(0.1))
simDat <- make_data(p_vector, design_RDM_RM, data=dat)
plot_history_effects(dat=simDat, degree=2)
```



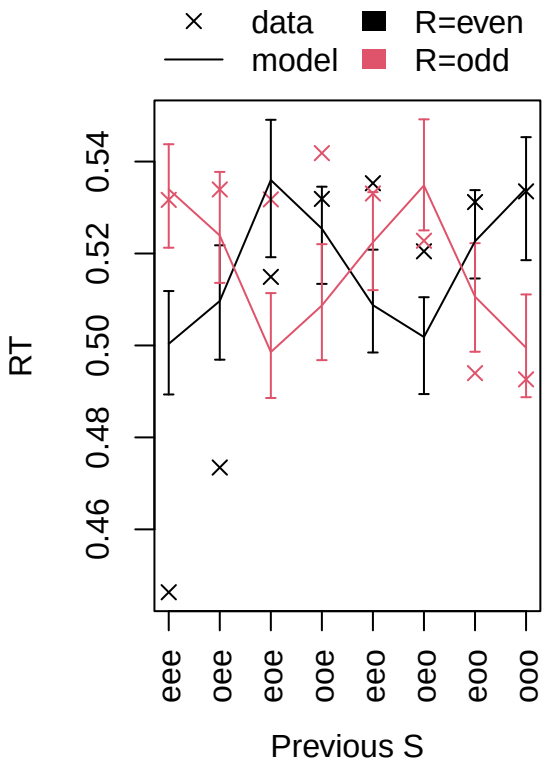
Which generates the expected pattern: if the previous trials were repeats, there is a tendency to repeat on the current trial, and that response will be faster compared to an alternation trial. Now let's see what it looks like in the real data:

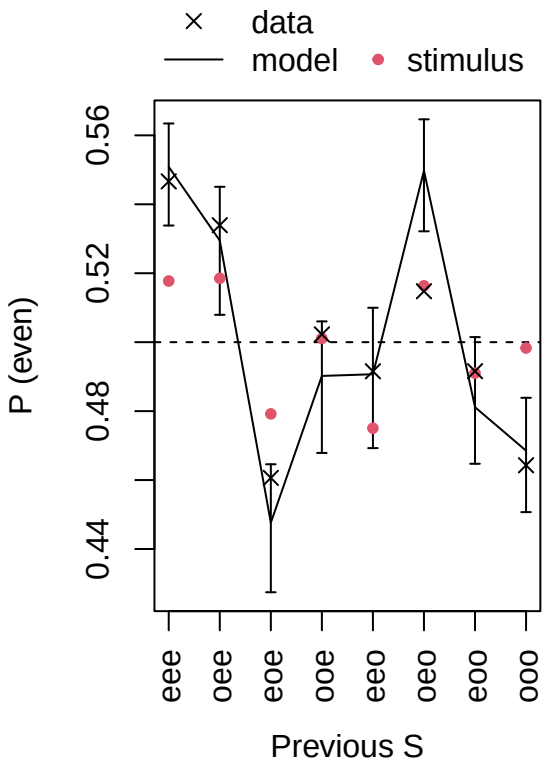
```
emc <- make_emc(dat, design=design_RDM_RM, compress=FALSE)
emc <- fit(emc, cores_per_chain=6, cores_for_chains=3,
           fileName='./samples/ds1_RMB.RData')
pp <- predict(emc, n_cores=10)
save(pp, file='./samples/ds1_RMB_pps.RData')
```

```
plot_history_effects(dat, pp, degree=2)
```




```
plot_history_effects(dat, pp, degree=1)
```

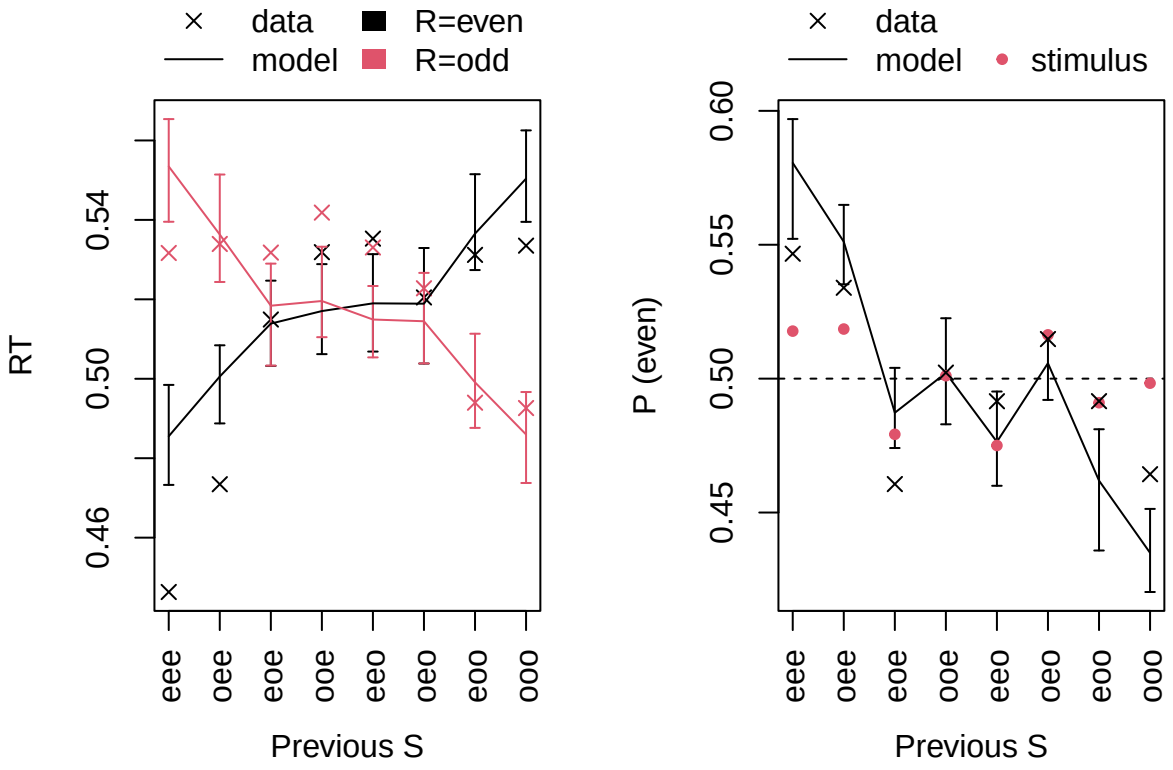




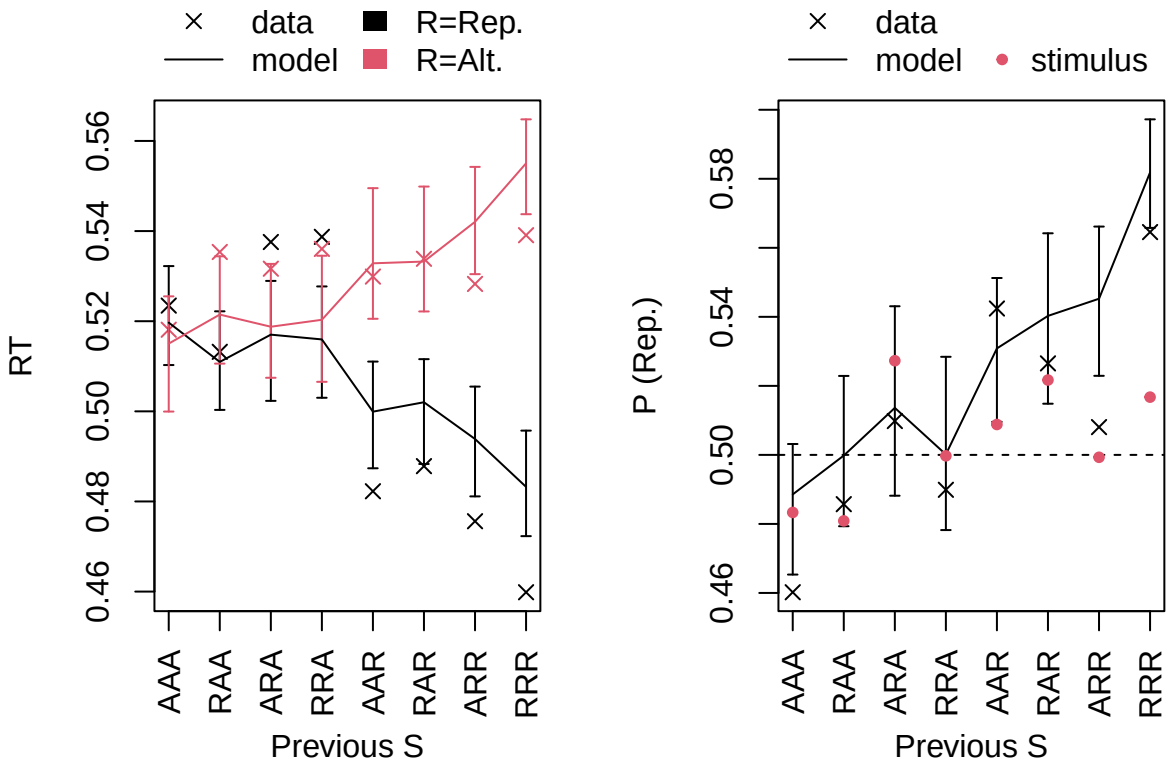

```r
dat$Stim1 <- ifelse(dat$S=='even', 1, -1)
dat$Stim2 <- dat$Stim1*Hmisc::Lag(dat$Stim1,1)
SM_RM_trend <- make_trend(cov_names=c('Stim1', 'Stim2'),
                          kernels = c('delta', 'delta'),
                          par_names=c('B_lRd', 'B_lR2d'),
                          bases=c('lin', 'lin'),
                          premap=TRUE,
                          filter_lR=TRUE)

design_RDM_RM <- design(model=RDM,
                        data=dat,
                        contrast=list(lM=ADmat, lR=ADmat, lR2=ADmat),
                        covariates=c('Stim1', 'Stim2'),
                        functions=list(lR2=function(x) {
                          lR2 <- as.numeric(x$lR)==as.numeric(Hmisc::Lag(x$S,2))
                          lR2[is.na(lR2)] <- TRUE
                          factor(lR2, levels=c(1,0), labels=c('rep', 'alt'))
                        }),
                        matchfun=function(d) d$S==d$lR,
                        formula=list(B ~ lR2+lR, v ~ lM, t0 ~ 1),
                        trend=SM_RM_trend,
                        constants=c('B_lR2d'=0, 'B_lR2d.q0'=0,
                                    'B_lRd'=0, 'B_lRd.q0'=0))

emc <- make_emc(dat, design=design_RDM_RM, compress=FALSE)
```

```
emc <- fit(emc, cores_per_chain=6, cores_for_chains=3,
           fileName='./samples/ds1_SMB_RMB.RData')
pp <- predict(emc, n_cores=10)
save(pp, file='./samples/ds1_SMB_RMB_pps.RData')
```

```
plot_history_effects(dat, pp, degree=1)
```




```
plot_history_effects(dat, pp, degree=2)
```



While the alternation effects are relatively small in this dataset, formal model comparison suggests that the inclusion of the RM mechanism sufficiently improves the quality of fit to warrant the additional model complexity:

```
emc_SMB <- get(load('./samples/ds1_SMB.RData'))
emc_SMB_RMB <- get(load('./samples/ds1_SMB_RMB.RData'))
compare(list('SM'=emc_SMB, 'SM+RM'=emc_SMB_RMB), BayesFactor=FALSE)
```

```
        DIC wDIC  BPIC wBPIC EffectiveN meanD Dmean  minD
SM    -6995    0 -6895     0        100 -7096 -7121 -7196
SM+RM -7085    1 -6995     1         90 -7175 -7194 -7265
```

Stimulus memory: DDM and drift rate effects

The examples above mimick Miletic et al. (2025) by using an RDM and assuming that stimulus memory causes a threshold (start point) bias. *EMC2* is flexibly designed to implement your own hypotheses. For example, we can propose a DDM instead and hypothesize that drift rates rather than start points are biased. This can be implemented as follows:

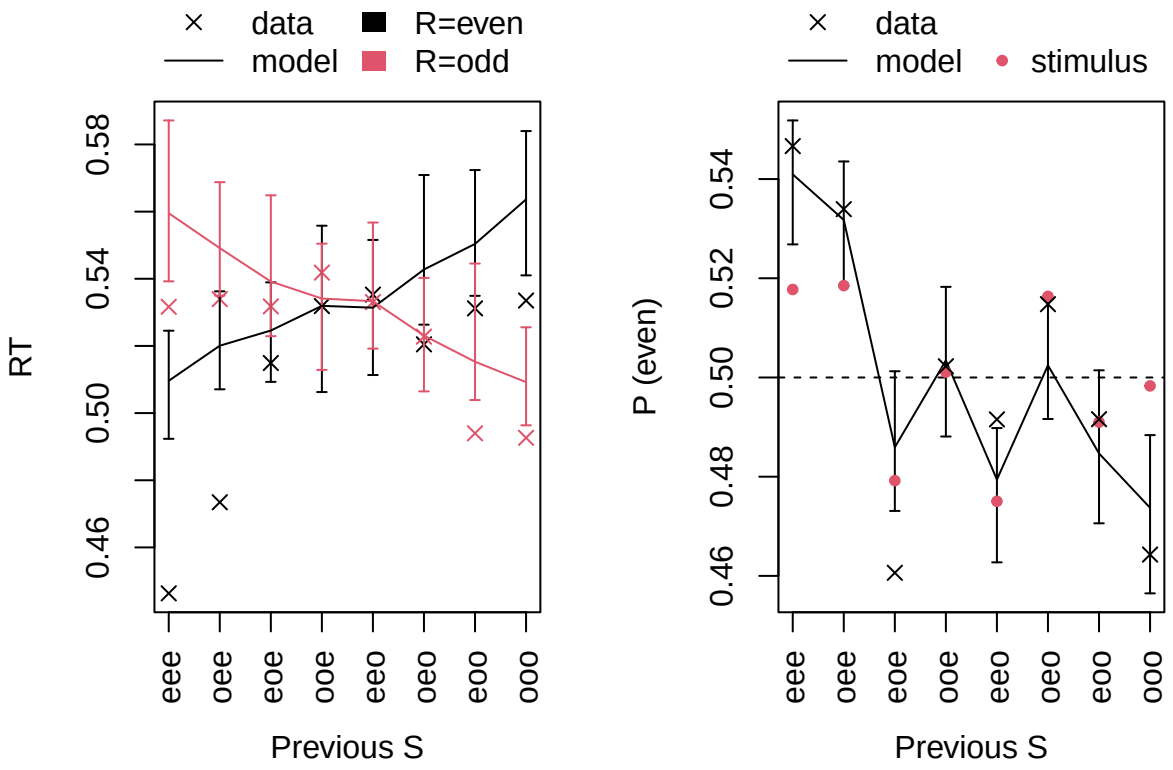
```
trend_SM_DDM <- make_trend(cov_names='Stim1',
                           kernels = 'delta',
                           par_names='v',
                           bases='lin',
                           premap=TRUE)
Smat <- matrix(c(-1,1), nrow = 2,dimnames=list(NULL,"dif"))
```

```
design_DDM <- design(model=DDM,
                     data=dat,
                     covariates=c('Stim1'),
                     contrasts=list(S=Smat),
                     formula=list(v ~ S, a~1, t0 ~ 1),
                     constants=c('v.q0'=0, 'v'=0),
                     trend=trend_SM_DDM)

# since v is estimated on the natural scale (in case of the DDM),
# we do not need to change the transformations, and we can use the default priors.
emc <- make_emc(dat, design=design_DDM, compress=FALSE)
emc <- fit(emc, cores_per_chain=6, cores_for_chains=3,
           fileName='./samples/ds1_SMv_DDM.RData')
```

However, this does not appear to fit as well qualitatively and loses quantitative model comparisons:

```
pp_ddm <- predict(emc, n_cores=10)
save(pp_ddm, file='./samples/ds1_SMv_DDM_pps.RData')
```

```
plot_history_effects(dat, pp_ddm)
```




```
#compare(sList=list(rdm=get(load('./samples/ds1_SMB.RData')),
#                   ddm=get(load('./samples/ds1_SMv_DDM.RData'))))
```

Mechanisms of dynamics: Accuracy memory

Accuracy memory follows the same logic as above, with a different covariate (errors) and a different parameter (urgency, v). Errors are the inverse of accuracy; they are 0 when the response matches the stimulus, and 1 otherwise. Unlike the presented stimulus, accuracy/error is not a static property of the experimental design but depends on the observed behavior. This also means that, when simulating behavior, accuracy needs to be re-calculated based on the newly simulated data.

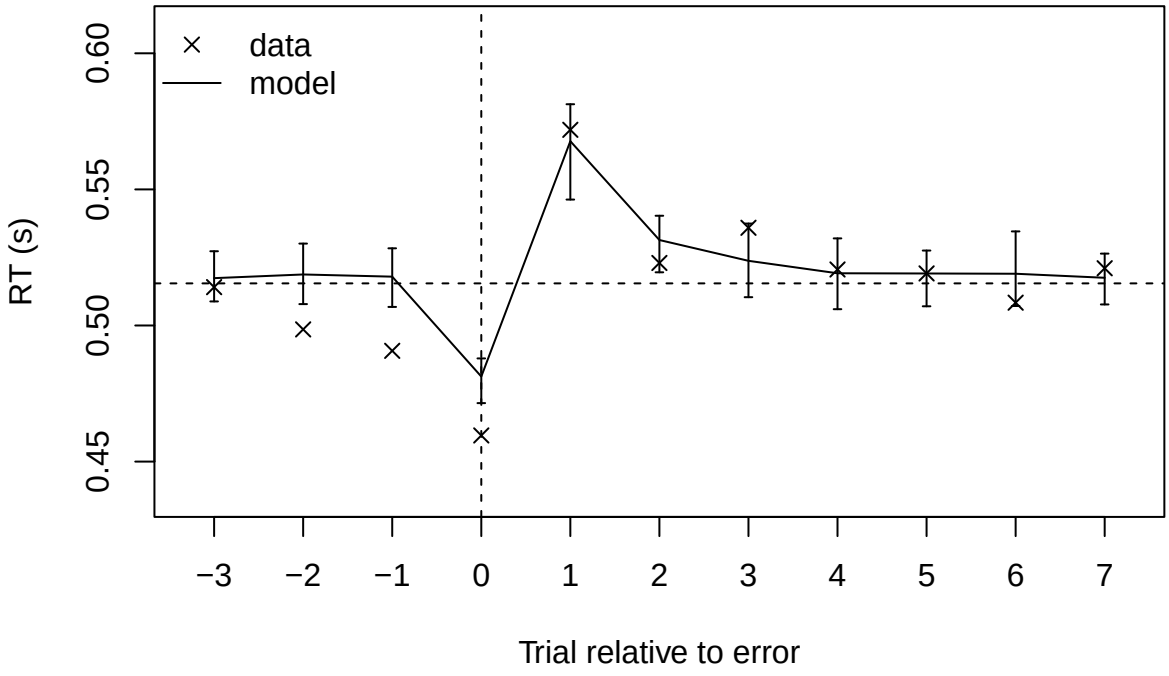
For this reason, it is useful to have *EMC2* calculate accuracy with a function rather than specifying it in the dataframe (in which case it would be treated as a static experimental factor and assumed fixed across simulations). The same function is applied when generating posterior predictives, so the accuracy is correctly determined in that phase.

```
AM_trend <- make_trend(cov_names='error',
                       kernels = 'delta',
                       par_names='v',
                       bases='lin',
                       premap=TRUE,
                       filter_lR=TRUE)
design_RDM_AM <- design(model=RDM,
                        data=dat,
                        contrast=list(lM=ADmat, lR=ADmat),
                        functions=list(error=function(x) x$S!=x$R),
                        covariates='error',
                        matchfun=function(d) d$S==d$lR,
                        formula=list(B ~ 1, v ~ lM, t0 ~ 1, s~lM),
                        trend=AM_trend,
                        constants=c('v.q0'=0, 's'=1))
# note that 1) we allow within-trial noise s to vary with accumulator
# match (correct/incorrect), which will allow us to capture the
# relative speed of errors.

# note that 2) in this case, we are not estimating v on the natural scale:
mapped_pars(design_RDM_AM)
# This is because the AM mechanism can push drift rates on individual trials
# to negative values when applied on the natural scale.
# Since the RDM has no known analytic likelihood for negative drift rates,
# this can lead to difficulties sampling. Hence, the effect of errors on
# urgency as we implement it here, is not strictly linear -- it is linear on
# the log-scale, and then exponentiated.
# The cognitive interpretation is similar.

emc <- make_emc(dat, design=design_RDM_AM, compress=FALSE)
emc <- fit(emc, cores_per_chain=6, cores_for_chains=3,
               fileName='./samples/ds1_AM.RData')
credint(emc)
```

Accuracy memory can help explain error-related effects, like post-error slowing. To visualize, we can generate posterior predictives and then use a convenience function to plot mean RT as a function of trial number relative to an error. Note that, since the covariate in this case depends on observed behavior, we need to simulate data and determine the covariate one trial at a time. *EMC2* detects whether the covariate is `rt`, `R`, or the output of one of the functions provided to design. If it is, it automatically simulates data trial-by-trial. As vectorisation is not possible in this case, this way of simulating data is slower.

```
emc <- get(load('./samples/ds1_AM.RData'))

pp <- predict(emc, n_cores=10, conditional_on_data=FALSE)
save(pp, file='./samples/ds1_AM_pps.RData')
```

```
plot_error_effects(dat, pp)
```



Like in the case of stimulus memory, the accuracy memory mechanism can be adjusted to other EAMs like the DDM, and it can affect other parameters. The DDM does not have an urgency mechanism, but we could, for example, allow AM to affect thresholds. This trend can be specified as post-transform since the thresholds are the same across all design cells. So in this case, we can sample thresholds on the log scale, then transform to the natural scale, and then apply the trend.

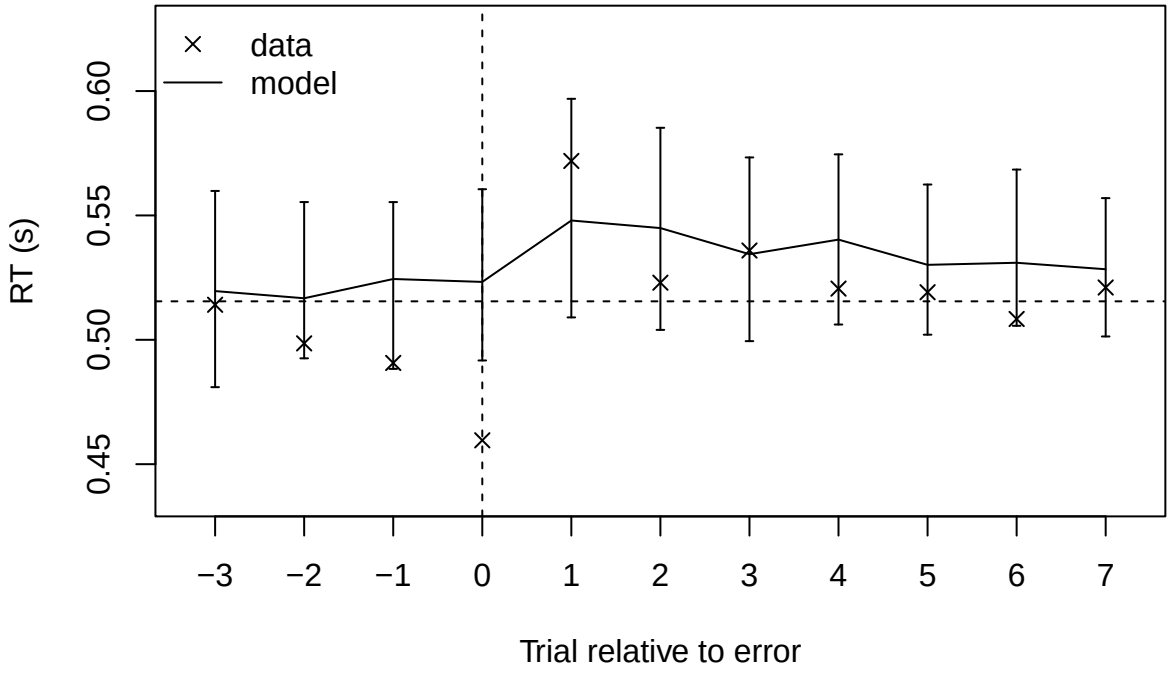
```
trend_AM_DDM <- make_trend(cov_names='error',
                           kernels = 'delta',
                           par_names='a',
                           bases='lin',
                           premap=FALSE, pretransform=FALSE)
Smat <- matrix(c(-1,1), nrow = 2,dimnames=list(NULL,"dif"))
design_DDM <- design(model=DDM,
                     data=dat,
                     functions=list(error=function(x) x$S!=x$R),
                     covariates='error',    # specify relevant covariate columns
                     contrasts=list(S=Smat),
                     formula=list(v ~ S, a~1, t0 ~ 1),
                     constants=c('a.q0'=0),
```

```
                  trend=trend_AM_DDM)

emc <- make_emc(dat, design=design_DDM, compress=FALSE)
emc <- fit(emc, cores_per_chain=6, cores_for_chains=3,
                fileName='./samples/ds1_AMa_DDM.RData')
```

However, this again does not appear to fit as well qualitatively, and loses model comparisons:

```
emc <- get(load('./samples/ds1_AMa_DDM.RData'))
pp_ddm <- predict(emc, n_cores=10, conditional_on_data=FALSE)
save(pp_ddm, file='./samples/ds1_AMa_DDM_pps.RData')
```

```
plot_error_effects(dat, pp_ddm)
```




```
# and loses model comparisons again
#compare(sList=list(rdm=get(load('./samples/ds1_AM.RData')),
#                   ddm=get(load('./samples/ds1_AMa_DDM.RData'))))
```

Special case: Fluency memory

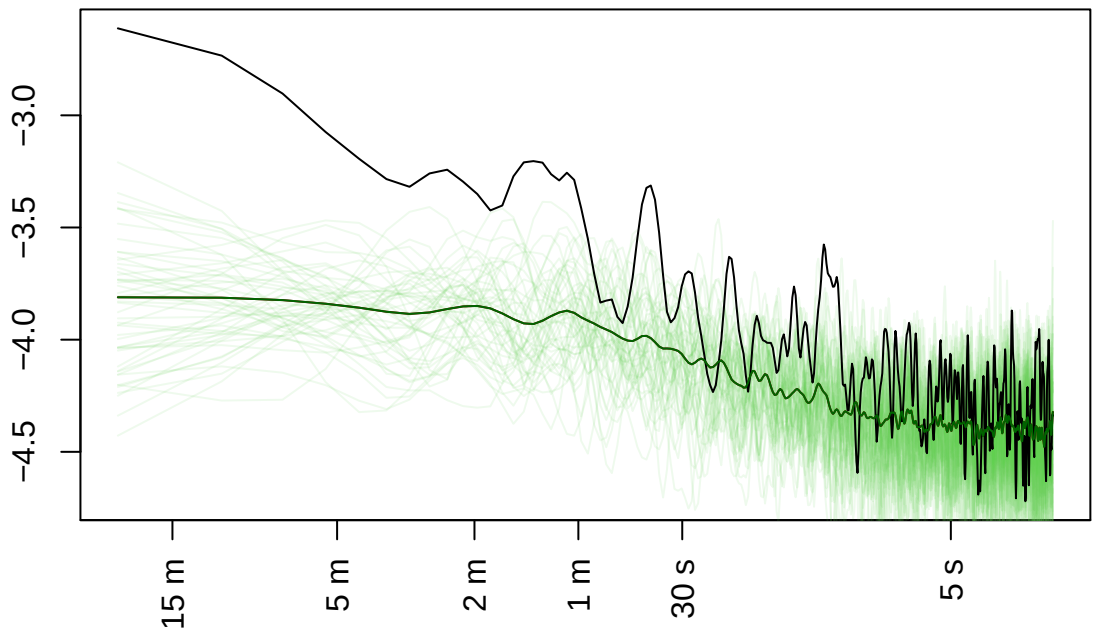
Fluency memory is a formalization of the idea that participants adjust their behavior according to perceived difficulty. To estimate perceived difficulty, we rely on a few simplifying assumptions: participants can quantify (1) their thresholds, and (2) their response time. In race models, the ratio of threshold over response time constitutes a measure of mean speed, which can be used as a proxy for difficulty. After all, we expect processing efficiency to be lower for difficult trials than for easy trials. Formally, $Q_{FM,t+1} =$

$Q_{FM,t} + \alpha_{FM}(b_t/rt_t - Q_{FM,t})$. Contrary to stimulus and accuracy memory, fluency memory is a very specific effect, and its implementation is not very flexible. It can only work for race models; it must influence thresholds; and the thresholds should be estimated on the natural scale. *EMC2* implements it as a separate kernel `deltab` with covariate `rt`:

```
trend_FM=make_trend(kernel='deltab',
                    base='lin',
                    cov_names ='rt',
                    par_names='B', premap = TRUE, filter_lR=TRUE)

design_FM <- design(model=RDM,
                  data=dat,
                  contrast=list(lM=ADmat),
                  matchfun=function(d) d$S==d$lR,
                  formula=list(B ~ 1, v ~ lM, t0 ~ 1, s~lM),
                  transform=list(func=c(B='identity'),          # should also update prior in this case
                                 lower=c(B.alpha=0.05)),
                  constants=c('B.q0'=3, 's'=log(1)),            # Why 3? When fitting a static RDM, the
                  trend=trend_FM)
prior_FM <- prior(design_FM, mu_mean=c('B'=2), mu_sd=c('B'=0.5))
emc <- make_emc(dat, design=design_FM, compress=FALSE, prior_list=prior_FM)

emc <- fit(emc, cores_per_chain=6, cores_for_chains=3,
                fileName='samples/ds1_FM.RData')
check(emc)
pp <- predict(emc, n_cores=10, conditional_on_data=FALSE)
save(pp, file='./samples/ds1_FM_pps.RData')
```

Fluency memory provides an explanation for the slower fluctuations in the observed response times. One useful way to visualize these is by creating a power spectrum of the RT data. We provide a convenience function that does this for the data and the posterior predictives of the estimated model:

```
# convenience function to plot the spectrum
plotSpectrum(dat=dat, pp=pp, trial_duration=1.265)  # mean trial duration of this task was 1.265 s, wh
```



which demonstrates that fluency can account for the fluctuations that are moderate to fast, but not the slowest fluctuations.