Studying the variations in reaction norms using the Reacnorm package

Pierre de Villemereuil

October 1, 2024

Contents

1 Summary and aim of the package

1.1 The Reacnorm package

1.2 The dragon dataset

1.3 Packages and seed used in this tutorial

The tutorial assumes that the *tidyverse* meta-package (containing e.g. `tidyr`, `dplyr`, `purrr`, `forcats` and `ggplot2`, that we'll be using) has been loaded. To complement `ggplot2`, and be able to compose plots, we will use the `patchwork` package. For the statistical modelling, we will use the Bayesian package `brms`. There are two reasons for this choice. First, by using a Bayesian method, we can easily compute the uncertainty surrounding our `Reacnorm` estimates, by computing a value for each iteration of the MCMC chain. Second, `brms` is a very versatile, and thus we can use it to implement all of the models (including non-linear models) we will be using in this tutorial. Finally, to work with the MCMC output of `brms`, we will be using the packages *posterior* and `bayesplot`. The tutorial assumes that all of those packages are loaded.

Another thing is that we will set a "seed" for the whole tutorial. This seed will allow for the reproducibility of the analysis across computers.

1.4 About Bayesian statistics and brms

We will be using Bayesian statistics...

2 Overview of the theory

Coming soon, a summary of the theoretical bases of the Reacnorm package. In the meantime, users can refer to the companion paper of the package.

3 Studying reaction norms in a discretised environment

3.1 A fully quadratic reaction norm

3.1.1 Overview of the data on aggressiveness

Let's start by looking at the data, shipped directly when loading the Reacnorm package:

```
head(dragon_discrete)
```

```
  Name_Env Temp Individual Aggressiveness Performance
1   Env_01   -2     Ind_01        -2.1600     -0.0234
2   Env_01   -2     Ind_02        -3.0300      0.0564
3   Env_01   -2     Ind_03         0.0278      0.0565
4   Env_01   -2     Ind_04        -1.3200      0.0744
5   Env_01   -2     Ind_05        -3.6800      0.0515
6   Env_01   -2     Ind_06        -2.7200     -0.0668
```

Another option is to look at the description of the dataset using `?dragon_discrete`. The dataset contains measures of phenotypic assays collected on dragons[1] kept in a (gigantic) thermostatic cage. Aggressiveness is measured using a complex, continuous index based on their behaviour when exposed to an armoured knight provoking them.

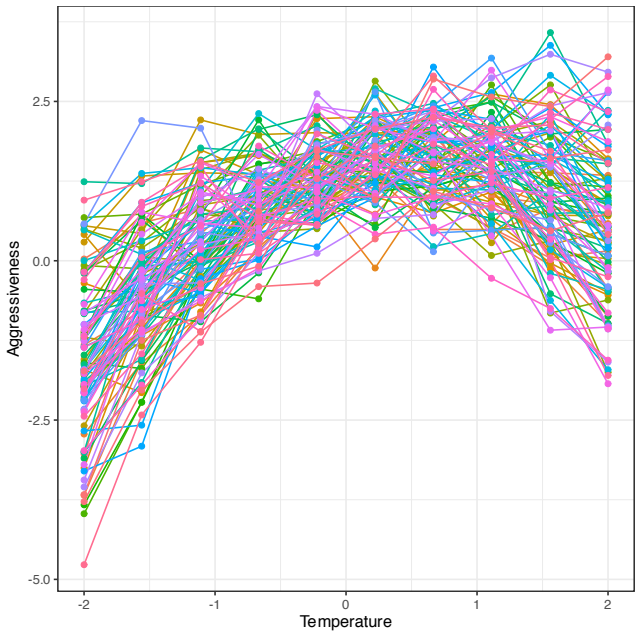

Figure 1: Dragons aggressiveness according to the experimental test temperature

We can have a look at how aggressiveness depends on the experimental temperature:

```
ggplot(dragon_discrete) +
    geom_line(aes(x = Env, y = Aggressiveness, group = Individual, colour = Individual)) +
    geom_point(aes(x = Env, y = Aggressiveness, group = Individual, colour = Individual)) +
    theme(legend.position = "none") +
    xlab("Temperature") + ylab("Aggressiveness")
```

Figure 1 shows the resulting graph, in which we can see that a quadratic curve will probably be a good fit for the reaction norm curve. So, this is what we'll use.

In order to compute a quadratic reaction norm, we have to compute the (mean-centered) squared values of the environment. To be sure to remember that we modified the original `dragon_discrete`, we will create a new dataset (say `tbl_dragon_ds`)

```
tbl_dragon_ds <-
    dragon_discrete |>
    mutate(Env_Sq = (Env - mean(Env))^2)
```

The mean-centering is necessary to have squared values that are not correlated with the direct environmental values[2].

▸ 3.1.2 Fitting a quadratic reaction norm to the data

Running the model We will be using the brms package to study (see subsection 1.4 for more information) to study this quadratic reaction norm. As a reminder, we will run the model for 3000

[1] For readers who have kept their childlike spirit and still believe in dragons, I am sorry to say the data have been simulated.

[2] Although it is a bit useless here, because the mean is already 0, but better be safe than sorry.

iterations in total, discarding the first 1000 iterations considered as lost during the warming-up. Since the NUTS algorithm is particularly efficient to reduce auto-correlation, we will conserve all consecutive iterations:

```
# Number of independent chains
n_chains <- 4
# Total number of iterations
n_iter <- 3000
# Number of iterations that will be discarded for the warm-up
n_warm <- 1000
# Thinning interval
n_thin <- 1
```

To study a quadratic reaction norm, we will use a linear model[3], with two predictors: the temperature and the squared-value of the temperature. We also need to specify to the model that each values of the three parameters (intercept, slope, second-order component) vary between individuals. This will be done with `brms` syntax to specify random effects, which is close to e.g. the `lme4` package:

```
form_quad <- brmsformula(Aggressiveness ~ Temp + Temp_Sq +
                                          (1 + Temp + Temp_Sq | Individual))
```

The function `brmsformula()` generates a formula to pass on the function actually running the model, which is named `brm()`:

```
model_agr <-
  brm(formula   = form_quad,
      data      = tbl_dragon_ds,
      save_pars = save_pars(group = FALSE),
      chains    = n_chains,
      cores     = n_chains,
      seed      = seed,
      iter      = n_iter,
      warmup    = n_warm,
      thin      = n_thin)
```

To explain what is happening here: we ask `brm()` to run a model using the formula `form_rn`, collecting data from the `tbl_dragon_ds` data.frame. We provide the characteristics of the chains we want `brms` to run. Note that we provide the `seed` to the function, so that the output is reproducible.

Checking the model We can have a look at the output of the model using the `summary()` function:

```
summary(model_agr)
```

```
 Family: gaussian
  Links: mu = identity; sigma = identity
Formula: Aggressiveness ~ Temp + Temp_Sq + (1 + Temp + Temp_Sq | Individual)
   Data: tbl_dragon_ds (Number of observations: 1000)
  Draws: 4 chains, each with iter = 3000; warmup = 1000; thin = 1;
         total post-warmup draws = 8000

Multilevel Hyperparameters:
~Individual (Number of levels: 100)
```

[3]Yes, the model itself is linear, even though the reaction norm is quadratic, because "linear" here must be understood as "linear in its parameters", which is the case of polynomial functions.

```
                     Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS
sd(Intercept)            0.28      0.04     0.21     0.35 1.00     4031
sd(Temp)                 0.42      0.03     0.36     0.49 1.00     2350
sd(Temp_Sq)              0.18      0.02     0.14     0.22 1.00     2390
cor(Intercept,Temp)     -0.21      0.13    -0.46     0.05 1.00      751
cor(Intercept,Temp_Sq)  -0.04      0.16    -0.34     0.28 1.00     1347
cor(Temp,Temp_Sq)        0.08      0.12    -0.16     0.32 1.00     2662
                     Tail_ESS
sd(Intercept)            5617
sd(Temp)                 4027
sd(Temp_Sq)              4029
cor(Intercept,Temp)      1588
cor(Intercept,Temp_Sq)   2467
cor(Temp,Temp_Sq)        4250

Regression Coefficients:
          Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept     1.48      0.04     1.41     1.55 1.00     6669     6685
Temp          0.53      0.04     0.44     0.61 1.00     2196     3530
Temp_Sq      -0.49      0.02    -0.53    -0.45 1.00     3527     5341

Further Distributional Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sigma     0.49      0.01     0.46     0.52 1.00     5206     6387

Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
and Tail_ESS are effective sample size measures, and Rhat is the potential
scale reduction factor on split chains (at convergence, Rhat = 1).
```

Beyond the classical values of point estimate, standard error and 95% CI provided for each parameter of the value, we get values to assess whether the algorithm went well (**vehtari_ranknormalization_2021**). Notably, $\hat{R}$ tests for convergence (i.e. whether the chains reached stationary state) and should near 1 (recommended values are $\hat{R} \leq 1.01$) The Bulk and Tail effective sample sizes (ESS) provide information regarding whether the chains were long enough to obtain precise estimates or not. Schematically, the ESS of a chain is the equivalent number of pure Monte Carlo sampling yielding the same amount of information. In other words, if you had 1000 iterations, but an ESS of 40, it is *as if* you drew only 40 independent samples from the posterior distribution of the parameter. The reason for this discrepancy comes from the fact that consecutive iterations in the chains are not independent (there is auto-correlation). While Bulk ESS provides information on how well we sampled around the mean (so, how well it is estimated), Tail ESS provides information on how well we sampled the tail (so, how well the variance is estimated). Both ESS should be above at least 400 for all parameters (**vehtari_ranknormalization_2021**).

We can also have a graphical look at the model, to see the traces (values of the parameters along the iterations, to check for convergence) and posterior distributions of the parameters (see **??**):

```
plot(model_agr)
```

To have a better look at how the model fits the data, we can have a look at the average reaction norm predicted by the model:

```
tbl_agr_mod <-
    tbl_dragon_ds |>
```

```
    mutate(Predict = predict(model_agr, re_formula = NA) |>
                     as_tibble()) |>
    unpack(Predict) |>
    select(Temp,
           Predict = Estimate,
           Predict_Low = Q2.5,
           Predict_Up  = Q97.5) |>
    summarise(across(starts_with("Predict"), mean),
              .by = Temp)

p_rn_agr <-
    p_aggr +
    geom_ribbon(data = tbl_agr_mod,
                mapping = aes(x = Temp, ymin = Predict_Low, ymax = Predict_Up),
                alpha = 0.3) +
    geom_line(data = tbl_agr_mod,
              mapping = aes(x = Temp, y = Predict),
              linewidth = 1)
```

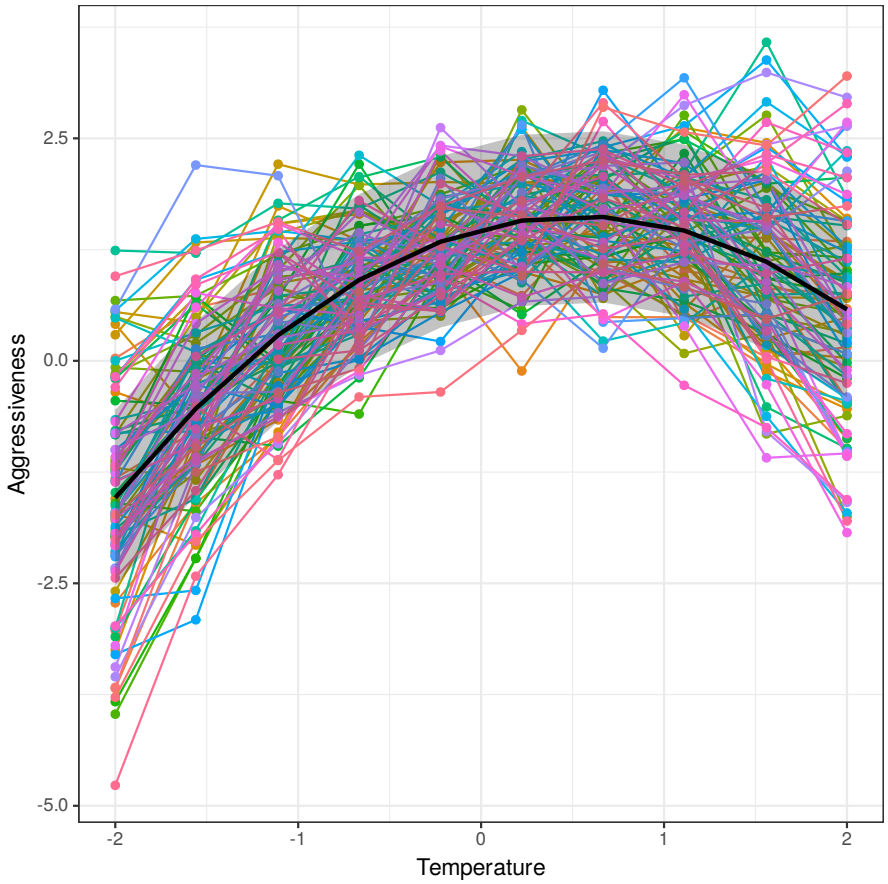

Figure 2: Aggressiveness individual data, with the average reaction norm predicted by the `mod_agr` model.

▸ *3.1.3 Decomposing the variance based on point estimates*

Getting point estimates In order to perform the variance decomposition using the `Reacnorm` package, we need first to extract the point estimates of key parameters in the model. The first thing we will need are the estimates of the quadratic coefficients of the model ($\bar{\theta}$ in the theoretical overview above). To do so, we will use the `fixef()` function:

```
theta_agr <- fixef(model_agr, robust = TRUE)[ , "Estimate"]
names(theta_agr) <- c("a", "b", "c")
theta_agr
```

```
         a          b          c
 1.4808551  0.5293001 -0.4903728
```

Similarly, we can extract the variance-covariance of the fitted random effects:

```
G_agr <-
    VarCorr(model_agr, robust = TRUE)[["Individual"]][["cov"]][ , "Estimate", ]
rownames(G_agr) <- colnames(G_agr) <- names(theta_agr)
G_agr
```

```
             a            b            c
a  0.075549554 -0.023922259 -0.002332428
b -0.023922259  0.171736042  0.005871739
c -0.002332428  0.005871739  0.031550777
```

Note that we used the `robust = TRUE` argument. This outputs the posterior median, rather than the more classical posterior mean, as a point estimate. In general, if the posterior distribution is symmetrical (see "b" prefixed panels in ??), both point estimates should be comparable. But for standard-deviations or variances of the random effects, posterior distributions tend to be strongly to slightly asymmetrical, in which case the posterior median is a better point estimate. We thus use `robust = TRUE` everywhere for consistency. We will also require the uncertainty around the $\bar{\theta}$ point estimates, i.e. the S matrix (see theoretical overview):

```
S_theta_agr <- vcov(model_agr)
rownames(S_theta_agr) <- colnames(S_theta_agr) <- c("a", "b", "c")
S_theta_agr
```

```
              a             b             c
a  0.0013203047 -0.0003068121 -0.0002255506
b -0.0003068121  0.0019102726  0.0000669430
c -0.0002255506  0.0000669430  0.0004393123
```

Design matrix The last ingredient we will require to use the `Reacnorm` package is the design matrix X is the linear model. Unfortunately, `brms` objects do not contain such matrix, but we can "reconstruct" it based on the formula of the model, using the `model.matrix()` function:

```
design_mat <- model.matrix(Aggressiveness ~ Temp + Temp_Sq, data = tbl_dragon_ds)
head(design_mat)
```

```
  (Intercept) Temp Temp_Sq
1           1   -2       4
2           1   -2       4
3           1   -2       4
4           1   -2       4
5           1   -2       4
6           1   -2       4
```

Getting the variance of average reaction norm and its decomposition In order to obtain the variance of the average reaction norm (V_{Plas}) and its decomposition, the simplest and quickest way is to use the `rn_phi_decomp()` function :

```
plas_agr <-
    rn_phi_decomp(theta = theta_agr, X = design_mat, S = S_theta_agr)
plas_agr
```

```
     V_Plas     Phi_b     Phi_c      Phi_b_c
1 0.9497063 0.4781417 0.5218583 7.671419e-17
```

Since the true reaction norm is quadratic, we know that the φ- and π-decomposition are equal, and thus, here we have $\varphi_b = \pi_{\text{Sl}}$ and $\varphi_c = \pi_{\text{Cv}}$. Hence, the function performing the π-decomposition would yield (approximately) the same result. However, because it requires performing numerical integration, it would take longer (roughly 200 times longer, but still instant here) and be slightly less exact:

```
plas_agr_pi <-
    rn_pi_decomp(th eta   = theta_agr,
                 G_theta = G_agr,
                 env     = tbl_dragon_ds[["Temp"]] |> unique(),
                 shape   = expression(a + b * x + c * x^2))
plas_agr_pi
```

```
     V_Plas    Pi_Sl     Pi_Cv
1 0.9537137 0.478577 0.5205334
```

There are two reasons for why the two functions slightly differ. The first is that, while `rn_phi_decomp()` accounts for the uncertainty in $\bar{\theta}$ using the S matrix, the `rn_pi_decomp()` function cannot do it. If we were to set S to a null matrix when calling `rn_phi_decomp()`, the results would be even close to `rn_pi_decomp()`:

```
rn_phi_decomp(theta = theta_agr, X = design_mat, S = 0 * diag(3))
```

```
     V_Plas     Phi_b     Phi_c      Phi_b_c
1 0.9537309 0.4793928 0.5206072 7.639302e-17
```

The second reason is that `rn_phi_decomp()` uses exact matrix computation, while `rn_pi_decomp()` is based on numerical integration, which is (slightly) more approximative. In the end, we can claim that $V_{\text{Plas}} = 0.95$, with $\pi_{\text{Sl}} = 0.48$ and $\pi_{\text{Cv}} = 0.52$. The variance V_{Plas} is the variance arising from variation along the black line in Figure 2. Slightly more of this variance is coming from the curvature of this line ($\pi_{\text{Cv}} = 0.52$) than from its average slope ($\pi_{\text{Sl}} = 0.48$), although these contributions are close to equality.

Getting the additive genetic variances and their decomposition To compute the additive genetic variance of the reaction norm (V_{Add}) and its γ-decomposition; the marginal additive genetic variance of the trait (V_{A}); and the additive genetic variance of plasticity ($V_{\text{A}\times\text{E}}$) and its ι-decomposition.

```
gen_agr <-
    rn_gen_decomp(theta = theta_agr, G_theta = G_agr, X = design_mat)
gen_agr
```

```
     V_Add       V_A     V_AxE   Gamma_a   Gamma_b   Gamma_c Gamma_a_b Gamma_b_c
1 0.4973828 0.1519671 0.3454157 0.1518942 0.5634872 0.2999246         0         0
    Gamma_a_c Iota_a    Iota_b    Iota_c Iota_a_b Iota_a_c Iota_b_c
1 -0.01530597      0 0.8113958 0.1886042        0        0        0
```

The additive genetic variance of the reaction is thus $V_{\text{Add}} = 0.50$), so roughly twice as low as V_{Plas}. It is composed for a third by the marginal additive genetic variance of the trait ($V_{\text{A}} = 0.15$) and

for two-thirds by the additive genetic variance of plasticity ($V_{A\times E} = 0.35$). This seems to suggest that there is a considerable amount of adaptive potential in the plasticity of aggressiveness. Most of the additive genetic variation in the reaction norm comes from variation in the slopes ($\gamma_b = 0.56$). Regarding genetic variation in plasticity itself, it is even more the case that most of the variation (thus adaptive potential) comes from the slope ($\iota_b = 0.81$). Note that, in this simple case, most of the covariance terms (e.g. $\gamma_{a,b} = 0$ or $\iota_{b,c} = 0$). For the sake of security, the `Reacnorm` function will always yield all components even if they are null. In the rest of this tutorial, we will remove such null elements by imposing a threshold. For this, we will use the `select()` function from `dplyr`:

```
rn_gen_decomp(theta = theta_agr, G_theta = G_agr, X = design_mat) |>
    select(where( \(col_) { abs(mean(col_)) > 10^-5 }) )
```

```
      V_Add       V_A     V_AxE   Gamma_a   Gamma_b   Gamma_c   Gamma_a_c
1 0.4973828 0.1519671 0.3454157 0.1518942 0.5634872 0.2999246 -0.01530597
     Iota_b    Iota_c
1 0.8113958 0.1886042
```

Less cluttered, uh?

▸ 3.1.4 Decomposing the variance using the full posterior distribution

Getting the posterior distributions of the parameters Getting estimates from the point estimates of the model is a nice first thing, but it is not the best (Bayesian) way to obtain our variance decomposition. It is better to compute the above parameter from each iteration of our model's chains. In order to do so, we will first have to collect the values of our parameters for each iterations of the chain. We will do so by setting the argument `summary = FALSE` in the functions that we used above:

```
theta_post_agr <- fixef(model_agr, summary = FALSE)
colnames(theta_post_agr) <- c("a", "b", "c")
head(theta_post_agr)
```

```
    variable
draw        a         b          c
   1 1.488025 0.4744503 -0.5005814
   2 1.511379 0.4318258 -0.5009057
   3 1.521973 0.4587290 -0.4994489
   4 1.532942 0.4816476 -0.5069072
   5 1.537320 0.4673945 -0.4882447
   6 1.501262 0.4572999 -0.5282360
```

```
G_post_agr <-
    VarCorr(model_agr, summary = FALSE)[["Individual"]][["cov"]] |>
    # We use apply() to transform the 3-dimensional array into a list
    apply(1, \(mat_) { mat_ }, simplify = FALSE) |>
    map( \(mat_) { rownames(mat_) <- colnames(mat_) <- c("a", "b", "c"); return(mat_) })
G_post_agr[[1]]
```

```
            a            b           c
a  0.07372066 -0.018482854 0.008082450
b -0.01848285  0.192524723 0.005297588
c  0.00808245  0.005297588 0.028833529
```

To transform those into posterior chains, we will use the package `posterior`:

```r
post_agr <- as_draws_df(theta_post_agr)
post_agr[["G"]] <- G_post_agr
post_agr
```

```
# A draws_df: 2000 iterations, 4 chains, and 4 variables
     a    b     c
1  1.5 0.47 -0.50
2  1.5 0.43 -0.50
3  1.5 0.46 -0.50
4  1.5 0.48 -0.51
5  1.5 0.47 -0.49
6  1.5 0.46 -0.53
7  1.5 0.55 -0.49
8  1.5 0.58 -0.47
9  1.5 0.55 -0.47
10 1.5 0.55 -0.53
                                                                                                 G
1                0.0737, -0.0185, 0.0081, -0.0185, 0.1925, 0.0053, 0.0081, 0.0053, 0.0288
2              0.0649, -0.0061, 0.0103, -0.0061, 0.1964, -0.0015, 0.0103, -0.0015, 0.0233
3                  0.0574, 0.0040, 0.0055, 0.0040, 0.2056, 0.0043, 0.0055, 0.0043, 0.0218
4                0.0811, 0.0080, -0.0034, 0.0080, 0.1684, 0.0061, -0.0034, 0.0061, 0.0306
5      6.7e-02, -8.2e-05, 4.7e-03, -8.2e-05, 1.6e-01, 1.5e-02, 4.7e-03, 1.5e-02, 2.5e-02
6  0.06988, -0.00548, -0.00811, -0.00548, 0.18584, -0.00077, -0.00811, -0.00077, 0.02962
7                0.0625, -0.0058, 0.0085, -0.0058, 0.1668, 0.0107, 0.0085, 0.0107, 0.0419
8                0.0636, -0.0196, 0.0049, -0.0196, 0.2080, 0.0071, 0.0049, 0.0071, 0.0331
9    0.06724, -0.00861, -0.00063, -0.00861, 0.18598, 0.01109, -0.00063, 0.01109, 0.03672
10               0.0907, -0.0192, 0.0024, -0.0192, 0.2508, 0.0134, 0.0024, 0.0134, 0.0387
# ... with 7990 more draws
# ... hidden reserved variables {'.chain', '.iteration', '.draw'}
```

We can agree that this is not the best output format for the G-matrix...

Subsetting the parameters As we can see from the output above, we have 8000 iterations. We could them all, but for the sake of computation time for this tutorial, we will subset to only 1000 iterations of the chains. To do so, we will again use the `posterior` package to "thin" the chains so that we end up with 1000 iterations :

```r
post_agr <- thin_draws(post_agr, thin = nrow(theta_post_agr) / 1000)
```

In order to be able to re-transform the future data.frames that we will generate, we will keep the "meta-information" that the `posterior` package keeps at supplementary columns starting with a dot (`.chain`, `.iteration`, `.draw`):

```r
post_agr_info <- select(post_agr, starts_with("."))
```

Getting the variance of average reaction norm and its decomposition To use the full posterior distribution of the parameters, we need to apply the `rn_phi_decomp()` to each iteration of the chains. To do so, we will use `apply()`:

```r
post_plas_agr <-
    post_agr |>
    select(a, b, c) |>
    apply(1, \(th_) rn_phi_decomp(theta = th_, X = design_mat, S = S_theta_agr)) |>
```

```
    # Collect the output of apply() into a data.frame
    bind_rows()  |>
    select(where( \(col_) { abs(mean(col_)) > 10^-5 })) |>
    # Transform this into a "draws" object using posterior package
    cbind(post_agr_info) |>
    as_draws_df()
summarise_draws(post_plas_agr)
```

```
# A tibble: 3 × 10
  variable  mean median     sd    mad    q5   q95  rhat ess_bulk ess_tail
  <chr>    <dbl>  <dbl>  <dbl>  <dbl> <dbl> <dbl> <dbl>    <dbl>    <dbl>
1 V_Plas   0.957  0.957 0.0834 0.0850 0.831 1.09  1.00     1012.     908.
2 Phi_b    0.480  0.478 0.0485 0.0481 0.405 0.557 0.999    1050.     933.
3 Phi_c    0.520  0.522 0.0485 0.0481 0.443 0.595 0.999    1050.     933.
```

The nice thing with the way we re-created a "draws" object from `posterior` is that we can compute diagnostic values of our parameters (see columns `rhat`, `ess_bulk` and `ess_tail`). The values for V_{Plas} is slightly larger than when we used the point estimates, because by averaging over the posterior distribution, due to the averaging over the posterior distribution[4]. This time, we also obtain information about uncertainty in the estimates, as well as their 95% credible interval. We can also plot graphics of the trace of these derived parameters, as well as their full posterior distribution (see Figure 3) using the `bayesplot` package :

```
mcmc_trace(post_gen_agr)
mcmc_areas(post_gen_agr,
           regex_pars = "^V",
           prob = 0.95,
           area_method = "scaled height") /
    mcmc_areas(post_gen_agr,
               regex_pars = "^[^V]",           # = Not starting with V
               prob = 0.95,
               area_method = "scaled height")
```

Note that we separated the plot into the actual variance on the one hand, and the π-decomposition[5] on the other hand.

Getting the additive genetic variances and their decomposition Again, to compute the additive genetic variances and their decomposition, we again need to execute the same function over all iterations. But this time, since we will need to iterate over the arguments `theta` ($\bar{\theta}$) and `G_theta` (G_θ) of `rn_gen_decomp()`, we need to be able to use several columns at once. To do so, we will first prepare a new column for $\bar{\theta}$ in our posterior draws:

```
post_agr[["theta"]] <-
    post_agr |>
    select(a:c) |>
    apply(1, \(vec_) { vec_ }, simplify = FALSE)
```

[4]Briefly, the issue is that V_{Plas} is a variance over the fixed effects estimates, so by averaging over the posterior distribution, part of the uncertainty in these fixed effects estimates is "absorbed" into V_{Plas}. This time, it is not possible to simply use the S variance-covariance matrix correction, because the influence of the prior distribution is such that we are not sure to be over-correcting or not.

[5]Yes, here we used `rn_phi_decomp()` and `Phi` is printed on the plot, but remember that since the reaction norm is fully quadratic, we have $\pi_{\text{Sl}} = \varphi_b$ and $\pi_{\text{Cv}} = \varphi_c$.

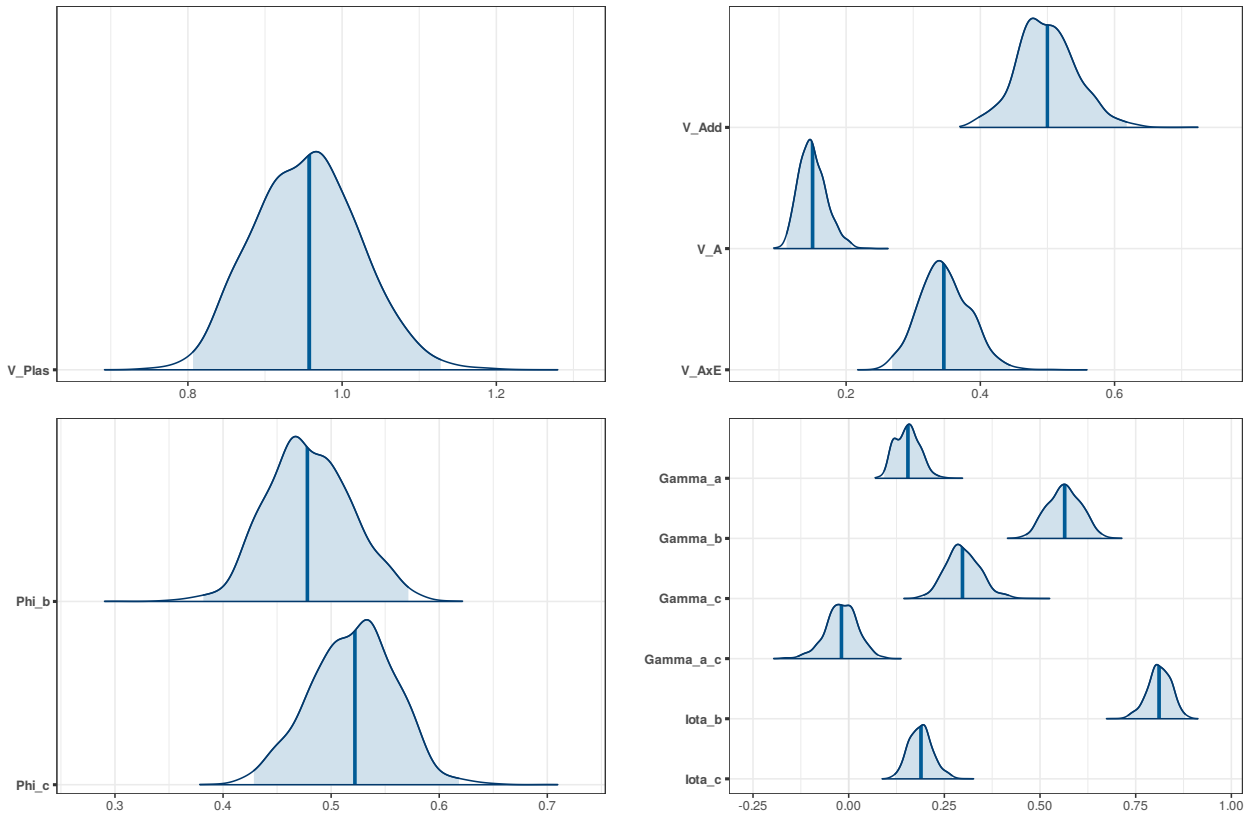

Figure 3: Posterior distribution of the variance decomposition of the reaction norm of aggressiveness, based on a quadratic model.

Now, we can use the function `map2()` from the `purrr` package from the tidyverse, to apply `rn_gen_decomp()` to both columns at once:

```
post_gen_agr <-
    map2(post_agr[["theta"]], post_agr[["G"]],
     \(th_, G_) { rn_gen_decomp(theta = th_,
                                G_theta = G_,
                                X = design_mat |> unique()) },
     .progress = TRUE) |>   # This makes map2() prints a nice progress bar
    bind_rows() |>
    select(where(\(col_) { abs(mean(col_)) > 10^-5 })) |>
    cbind(post_agr_info) |>
    as_draws_df()
summarise_draws(post_gen_agr)
```

```
# A tibble: 9 × 10
  variable     mean  median     sd    mad     q5    q95  rhat ess_bulk ess_tail
  <chr>       <dbl>   <dbl>  <dbl>  <dbl>  <dbl>  <dbl> <dbl>    <dbl>    <dbl>
1 V_Add       0.502   0.500  0.0556 0.0528  0.411 0.599  0.999     933.    1067.
2 V_A         0.153   0.150  0.0257 0.0248  0.116 0.200  1.00      765.     677.
3 V_AxE       0.349   0.346  0.0466 0.0463  0.275 0.427  0.998     897.    1033.
4 Gamma_a     0.156   0.155  0.0383 0.0417  0.100 0.222  0.999    1037.    1035.
5 Gamma_b     0.564   0.564  0.0519 0.0551  0.482 0.647  1.00      815.     933.
6 Gamma_c     0.301   0.297  0.0557 0.0547  0.216 0.403  1.00      834.     947.
7 Gamma_a_c -0.0211 -0.0189 0.0524 0.0482 -0.115 0.0613 1.00      790.    1012.
```

```
8 Iota_b     0.810   0.811  0.0387 0.0388  0.739 0.869  1.00      826.     878.
9 Iota_c     0.190   0.189  0.0387 0.0388  0.131 0.261  1.00      826.     878.
```

Here, again, we can also plot the traces and posterior distributions of these derived parameters (see Figure 3 for the latter):

```
mcmc_trace(post_gen_agr)
mcmc_areas(post_gen_agr,
           regex_pars = "^V",
           prob = 0.95,
           area_method = "scaled height") /
    mcmc_areas(post_gen_agr,
               regex_pars = "^[^V]",
               prob = 0.95,
               area_method = "scaled height")
```

The point estimates are very close to what we obtained with their direct computation from the point estimates from the model, but here, we have the full posterior of these variance decomposition, and can e.g. compute their 95% credible interval.

3.2 Analysing a non-linear reaction norm with a quadratic curve

3.2.1 Overview of the data on performance

The data on performance can be found, yet again, in the `dragon_discrete` dataset shipped with the `Reacnorm package`, that we transformed into `tbl_dragon_ds` (see the `Performance` column):

```
  head(tbl_dragon_ds)
```

```
  Name_Env Temp Individual Aggressiveness Performance Temp_Sq
1   Env_01   -2     Ind_01        -2.1600     -0.0234       4
2   Env_01   -2     Ind_02        -3.0300      0.0564       4
3   Env_01   -2     Ind_03         0.0278      0.0565       4
4   Env_01   -2     Ind_04        -1.3200      0.0744       4
5   Env_01   -2     Ind_05        -3.6800      0.0515       4
6   Env_01   -2     Ind_06        -2.7200     -0.0668       4
```

They are data providing a measure of locomotive performance of the dragons measured at different temperatures. Locomotive performance is measured as the maximum sprint speed attained by individuals, when stimulated with a dummy princess at the end of a very long (thermostatic) corridor.

As for aggressiveness, we can have a look at how thermal performance depends on the experimental temperature:

```
p_tpc <-
    ggplot(tbl_dragon_ds) +
    geom_line(aes(x = Temp, y = Performance, group = Individual, colour = Individual)) +
    geom_point(aes(x = Temp, y = Performance, group = Individual, colour = Individual)) +
    theme(legend.position = "none") +
    xlab("Temperature") + ylab("Performance")
```

Figure 4 shows the resulting graph. Clearly, a quadratic curve will not be a perfect fit in this case. We will, however, make do with a quadratic reaction norm to start with, to be able to understand

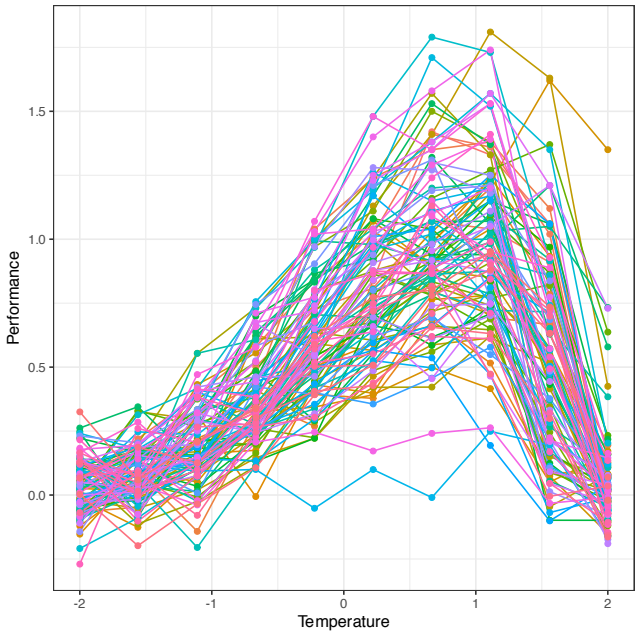

Figure 4: Dragons thermal performance, measured as locomotive performance, according to the experimental test temperature

the average variation in terms of slope and curvature. We will measure the level of error we are making by comparing our model with a more general character-state approach, and by computing the M^2_{Plas} introduced in the companion article.

▸ 3.2.2 Fitting a quadratic reaction norm to the data

Running the model The model is run exactly as in subsubsection 3.1.2, although here we will use the column `Performance` as the response variable:

```
form_quad <- brmsformula(Performance ~ Temp + Temp_Sq +
                                       (1 + Temp + Temp_Sq | Individual))
model_tpc_quad <-
    brm(formula   = form_quad,
        data      = tbl_dragon_ds,
        save_pars = save_pars(group = FALSE),
        chains    = n_chains,
        cores     = n_chains,
        seed      = seed,
        iter      = n_iter,
        warmup    = n_warm,
        thin      = n_thin)
```

This model should take approximately the same amount of time to run as `model_agr` previously.

Checking the model We first need to check that everything went well by looking at the model summary:

```
summary(model_tpc)
```

```
 Family: gaussian
  Links: mu = identity; sigma = identity
Formula: Performance ~ Temp + Temp_Sq + (1 + Temp + Temp_Sq | Individual)
```

```
 Data: tbl_dragon_ds (Number of observations: 1000)
Draws: 4 chains, each with iter = 3000; warmup = 1000; thin = 1;
       total post-warmup draws = 8000

Multilevel Hyperparameters:
~Individual (Number of levels: 100)
                      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sd(Intercept)             0.20      0.02     0.16     0.24 1.00     3007     4285
sd(Temp)                  0.06      0.01     0.04     0.08 1.00     4355     5811
sd(Temp_Sq)               0.05      0.01     0.04     0.07 1.00     3420     4693
cor(Intercept,Temp)       0.52      0.15     0.22     0.79 1.00     3235     3793
cor(Intercept,Temp_Sq)   -0.88      0.05    -0.96    -0.76 1.00     4780     4768
cor(Temp,Temp_Sq)        -0.10      0.21    -0.51     0.29 1.00     3039     3868

Regression Coefficients:
          Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept     0.74      0.02     0.70     0.79 1.00     2869     4391
Temp          0.12      0.01     0.11     0.14 1.00     5290     5411
Temp_Sq      -0.17      0.01    -0.19    -0.16 1.00     4809     5624

Further Distributional Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sigma     0.26      0.01     0.25     0.27 1.00     8096     6268

Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
and Tail_ESS are effective sample size measures, and Rhat is the potential
scale reduction factor on split chains (at convergence, Rhat = 1).
```

We can also plot the traces and posterior distributions of the parameters of the model (see Figure 5):

```
plot(model_tpc)
```

▸ *3.2.3 A first variance decomposition*

▸ *3.2.4 Fitting a character-state model to the data*

Running the model

Checking the model

▸ *3.2.5 A second variance decomposition, with M^2_{Plas}*

• 3.3 Analysing a non-linear reaction norm with a non-linear model

■ 4 Studying reaction norms in a continuous environment

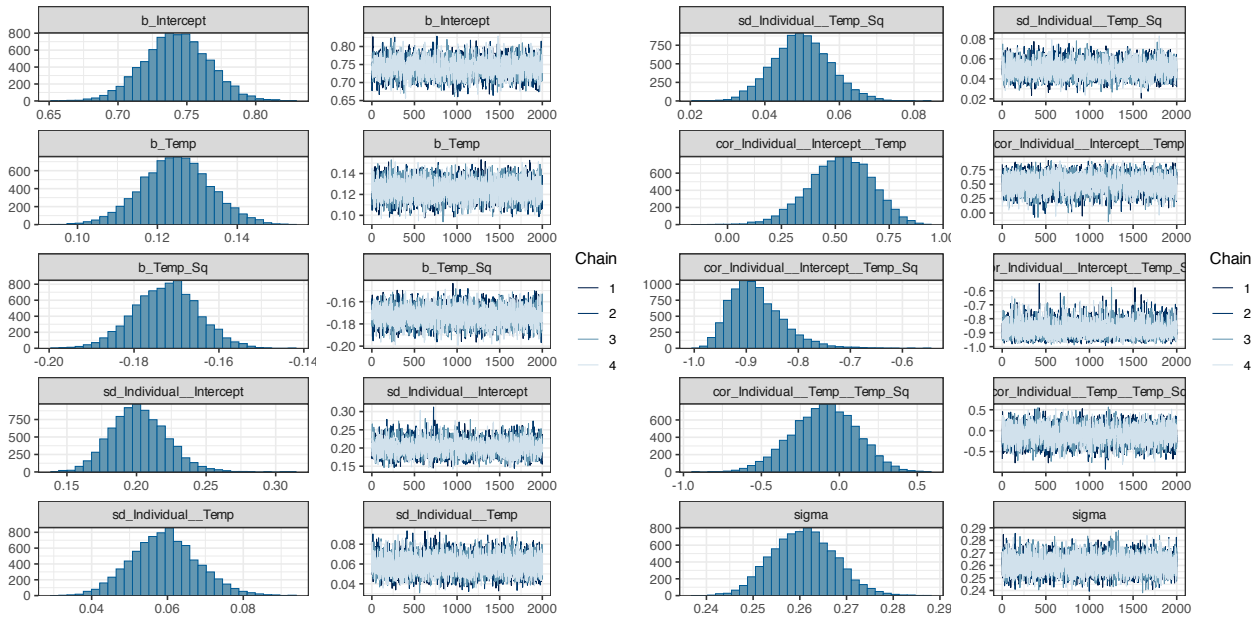

Figure 5: Plot of the `mod_tpc_quad` model. Parameters starting with "b" are the fixed effects parameters of the model, and parameters starting with "sd" are the standard deviation of the random effects. The parameter "sigma" is the residual standard deviation.