Studying the variations in reaction norms using the Reacnorm package

Pierre de Villemereuil

October 9, 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.

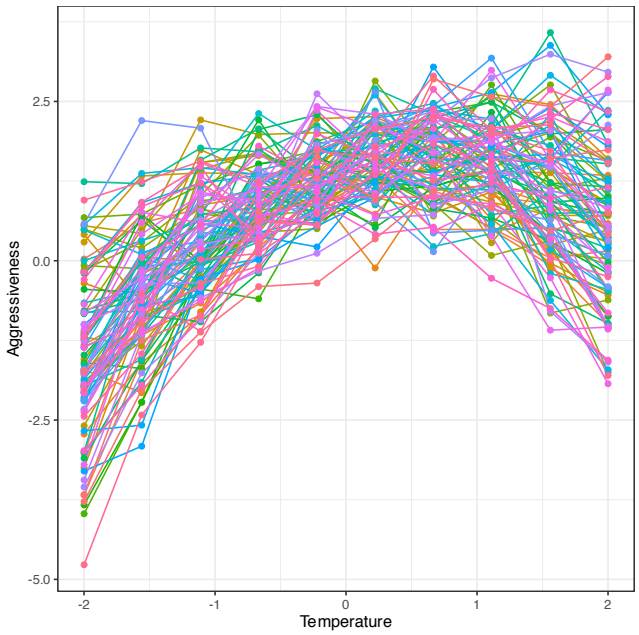

Figure 1: Dragons aggressiveness according to the experimental test temperature

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. Finally, the `save_pars = save_pars(group = FALSE)` tells `brms` that we do not want the random effects predictors to be saved in the model output, as they take a lot of space and are of no use for us in this tutorial.

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
```

[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.

```
Multilevel Hyperparameters:
~Individual (Number of levels: 100)
                      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 et al. 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 et al. 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 Figure 2):

```
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:

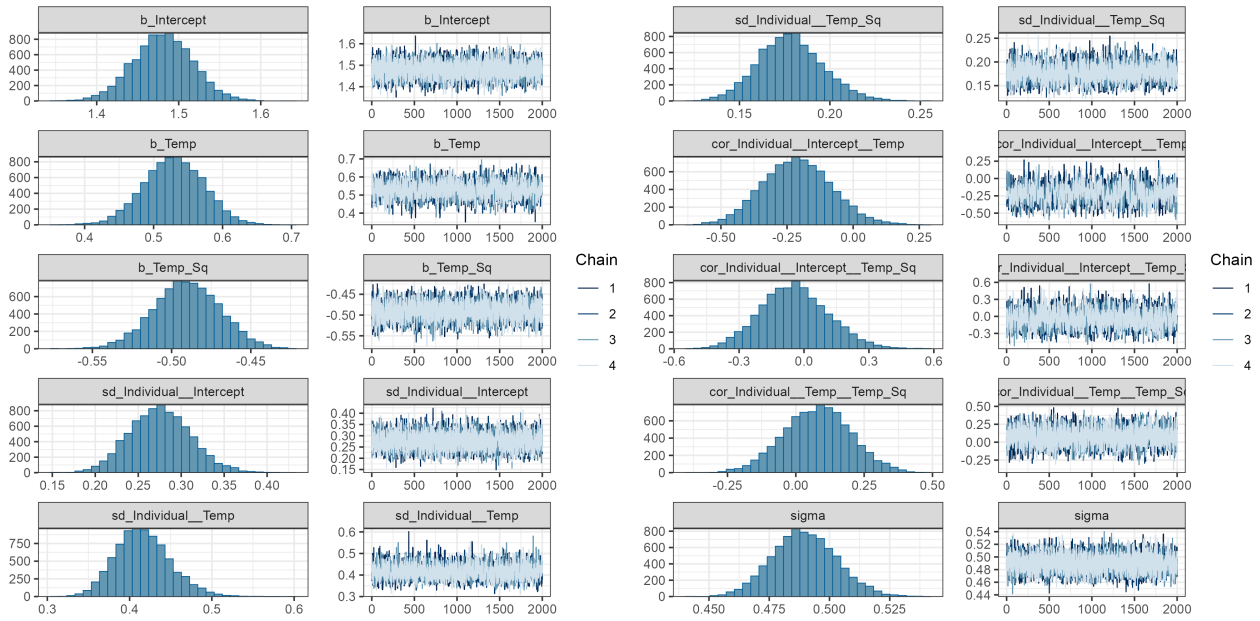

Figure 2: Plot of the `mod_agr` model. Parameters starting with "b" are the fixed effects parameters of the model, and parameters starting with "sd" are the standard deviation of the random effects. The parameter "sigma" is the residual standard deviation.

```
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)
```

▸ *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:

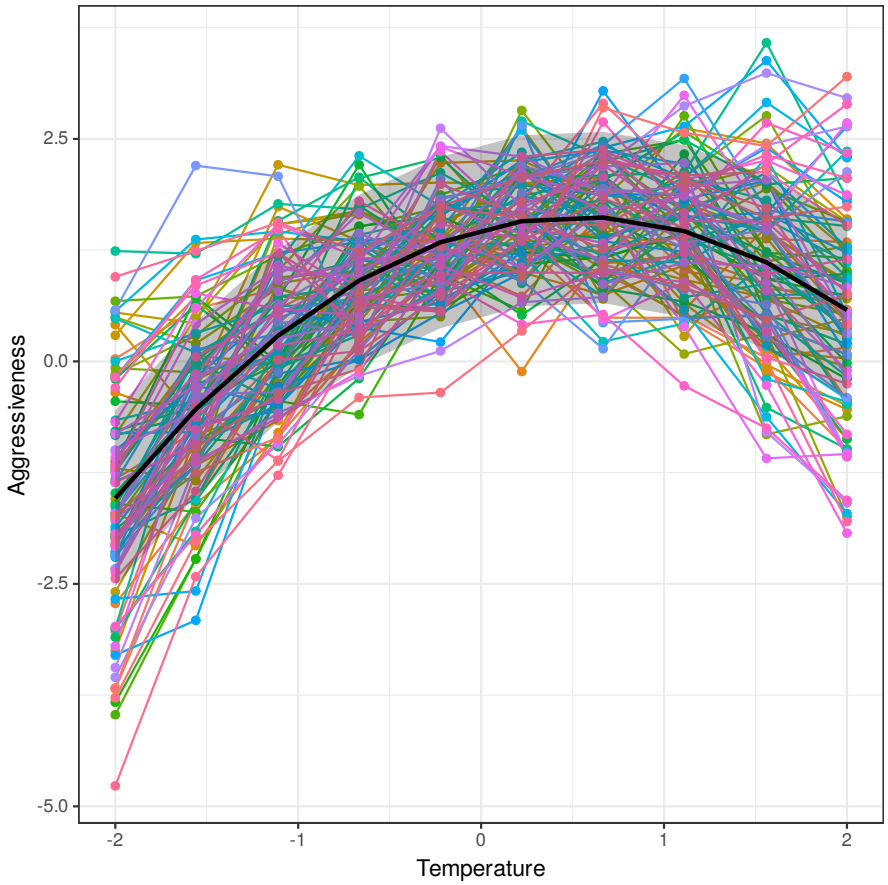

Figure 3: Aggressiveness individual data, with the average reaction norm predicted by the `mod_agr` model.

```
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 Figure 2), 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 can also extract the residual variance, that will be useful to get at the total phenotypic variance contained in the reaction norm:

```
vr_agr <- VarCorr(model_agr, robust = TRUE)[["residual__"]][["sd"]][ , "Estimate"]^2
vr_agr
```

```
[1] 0.2394762
```

Finally, we will 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 $V_{\mathrm{Plas}} = 0.95$, with $\pi_{\mathrm{Sl}} = 0.48$ and $\pi_{\mathrm{Cv}} = 0.52$. The variance V_{Plas} is the variance arising from variation along the black line in Figure 8. Slightly more of this variance is coming from the curvature of this line ($\pi_{\mathrm{Cv}} = 0.52$) than from its average slope ($\pi_{\mathrm{Sl}} = 0.48$), 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 ($V_{\mathrm{A\times E}}$) 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 $V_{\mathrm{Add}} = 0.50$), so roughly twice as low as V_{Plas}. It is composed for a third by the marginal additive genetic variance of the trait ($V_{\mathrm{A}} = 0.15$) and for two-thirds by the additive genetic variance of plasticity ($V_{\mathrm{A\times E}} = 0.35$). 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?

Computing the total phenotypic variance and the variance-standardised estimates Now that we have everything, we can finally compute the total phenotypic variance in the reaction norm:

```
v_tot_agr <- plas_agr[["V_Plas"]] + gen_agr[["V_Add"]] + vr_agr
v_tot_agr
```

```
[1] 1.686565
```

By dividing V_{Plas}, V_{Add}, V_{A} and $V_{\text{A×E}}$, we can obtain the variance-standardised estimates P^2_{RN}, h^2_{RN}, h^2 and h^2_{I}:

```
v_tot_agr <- plas_agr[["V_Plas"]] + gen_agr[["V_Add"]] + vr_agr
var_agr <-
    c(P2     = plas_agr[["V_Plas"]] / v_tot_agr,
      h2_RN  = gen_agr[["V_Add"]] / v_tot_agr,
      h2     = gen_agr[["V_A"]] / v_tot_agr,
      h2_I   = gen_agr[["V_AxE"]] / v_tot_agr,
      T2     = (plas_agr[["V_Plas"]] + gen_agr[["V_Add"]]) / v_tot_agr)
var_agr
```

```
        P2      h2_RN         h2       h2_I         T2
0.56310083 0.29490873 0.09010449 0.20480423 0.85800955
```

As we mentioned above, the contribution of the variance arising from plasticity due to the average reaction norm is larger than the contribution of the total additive genetic variance (i.e. $P^2_{\text{RN}} = 0.56 > 0.29 = h^2_{\text{RN}}$). This also illustrate one of the fundamental results of the companion paper, i.e. $h^2_{\text{RN}} = h^2 + h^2_{\text{I}}$. The reaction norm explains a large part of the total phenotypic variance ($T^2_{\text{RN}} = 0.86$).

▸ 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
```

```
vr_post_agr <-
    VarCorr(model_agr, summary = FALSE)[["residual__"]][["sd"]][ , 1]^2
head(vr_post_agr)
```

```
        1         2         3         4         5         6
0.2574979 0.2319611 0.2425255 0.2417751 0.2719007 0.2292515
```

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

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

```
# A draws_df: 2000 iterations, 4 chains, and 5 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  V_R
1           0.0737, -0.0185, 0.0081, -0.0185, 0.1925, 0.0053, 0.0081, 0.0053, 0.0288 0.26
2         0.0649, -0.0061, 0.0103, -0.0061, 0.1964, -0.0015, 0.0103, -0.0015, 0.0233 0.23
3            0.0574, 0.0040, 0.0055, 0.0040, 0.2056, 0.0043, 0.0055, 0.0043, 0.0218 0.24
4           0.0811, 0.0080, -0.0034, 0.0080, 0.1684, 0.0061, -0.0034, 0.0061, 0.0306 0.24
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 0.27
6 0.0698, -0.00548, -0.00811, -0.00548, 0.186, -0.00077, -0.00811, -0.00077, 0.0296 0.23
7           0.0625, -0.0058, 0.0085, -0.0058, 0.1668, 0.0107, 0.0085, 0.0107, 0.0419 0.24
8           0.0636, -0.0196, 0.0049, -0.0196, 0.2080, 0.0071, 0.0049, 0.0071, 0.0331 0.25
9   0.06724, -0.00861, -0.00063, -0.00861, 0.185, 0.011, -0.00063, 0.01109, 0.03672 0.22
10          0.0907, -0.0192, 0.0024, -0.0192, 0.2508, 0.0134, 0.0024, 0.0134, 0.0387 0.25
# ... 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 :

```
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`):

```
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()`:

```
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 5) 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")
```

[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.

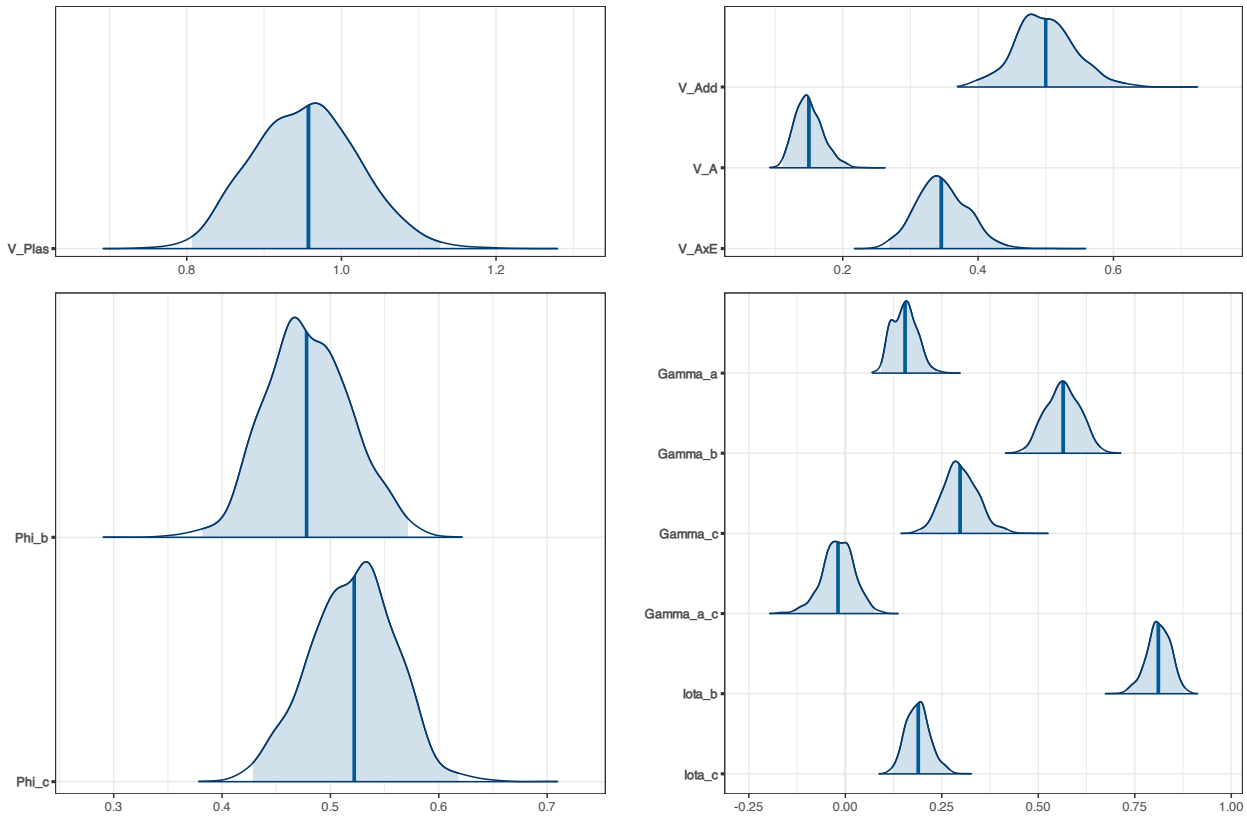

Figure 4: Posterior distribution of the variance decomposition of the reaction norm of aggressiveness, based on a quadratic model.

Note that we separated[5] the plot into the actual variance on the one hand, and the π-decomposition[6] 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)
```

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()) },
```

[5] Yes, that is the role of `/` between the two calls to `mcmc_areas()`, a syntax provided by the awesome `patchwork` package to combine plots!

[6] 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 $\pi_{\text{Cv}} = \varphi_c$.

```
    .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 5 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.

Getting the variance-standardised estimates If we want to compute the variance-standardised estimates of our variance-decomposition (i.e. P^2_{RN}, h^2_{RN}, h^2 and h^2_{I}), we will need to compute the total phenotypic variance in the reaction norm. An elegant way to do so is to construct a `posterior` draws object containing all the variance parameters:

```
post_var_agr <-
    bind_draws(post_agr, post_plas_agr, post_gen_agr) |>
    subset_draws(variable = c("V_Plas", "V_Add", "V_A", "V_AxE", "V_R")) |>
    mutate_variables(V_Tot = V_Plas + V_Add + V_R)
post_var_agr
```

```
# A draws_df: 250 iterations, 4 chains, and 6 variables
   V_Plas V_Add  V_A V_AxE  V_R V_Tot
1    0.88  0.55 0.18  0.37 0.26   1.7
2    0.93  0.54 0.16  0.38 0.22   1.7
3    0.99  0.44 0.13  0.31 0.25   1.7
```

```
4    0.97  0.46 0.15  0.32 0.25   1.7
5    1.12  0.54 0.16  0.38 0.25   1.9
6    0.87  0.53 0.13  0.41 0.24   1.6
7    1.06  0.48 0.17  0.31 0.25   1.8
8    0.90  0.58 0.20  0.37 0.22   1.7
9    0.89  0.60 0.21  0.39 0.27   1.8
10   0.91  0.57 0.23  0.34 0.25   1.7
# ... with 990 more draws
# ... hidden reserved variables {'.chain', '.iteration', '.draw'}
```

Then, we can produce a table of all the parameters divided by the total phenotypic variance of the reaction norm (we will use `transmute()` from `dplyr` in this case, to automatically get rid of the old columns, but this means we have to make our dataset a `posterior` object again):

```
post_std_agr <-
    post_var_agr |>
    transmute(P2    = V_Plas / V_Tot,
              H2_RN = V_Add / V_Tot,
              H2    = V_A / V_Tot,
              H2_I  = V_AxE / V_Tot,
              T2    = (V_Plas + V_Add) / V_Tot) |>
    cbind(post_agr_info) |>
    as_draws_df()

summarise_draws(post_std_agr)
mcmc_trace(post_std_agr)
mcmc_areas(post_std_agr,
           prob = 0.95,
           area_method = "scaled height")
```

```
# A tibble: 5 × 10
  variable   mean median     sd    mad     q5   q95  rhat ess_bulk ess_tail
  <chr>     <dbl>  <dbl>  <dbl>  <dbl>  <dbl> <dbl> <dbl>    <dbl>    <dbl>
1 P2        0.563  0.564  0.0283 0.0289 0.515  0.607 0.997     921.     915.
2 H2_RN     0.295  0.294  0.0272 0.0274 0.252  0.341 0.998     898.     895.
3 H2        0.0900 0.0883 0.0146 0.0148 0.0686 0.116 1.00      706.     882.
4 H2_I      0.205  0.204  0.0235 0.0224 0.170  0.246 0.998     898.    1021.
5 T2        0.858  0.859  0.0109 0.0113 0.840  0.876 0.999    1023.     953.
```

• 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
```

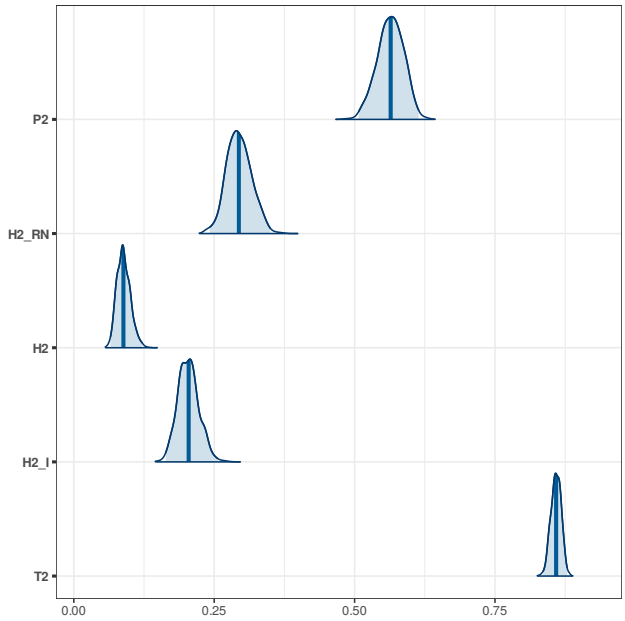

Figure 5: Posterior distribution of the variance-standardised estimates of our variance decomposition of the reaction norm of aggressiveness, based on a quadratic model.

```
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.

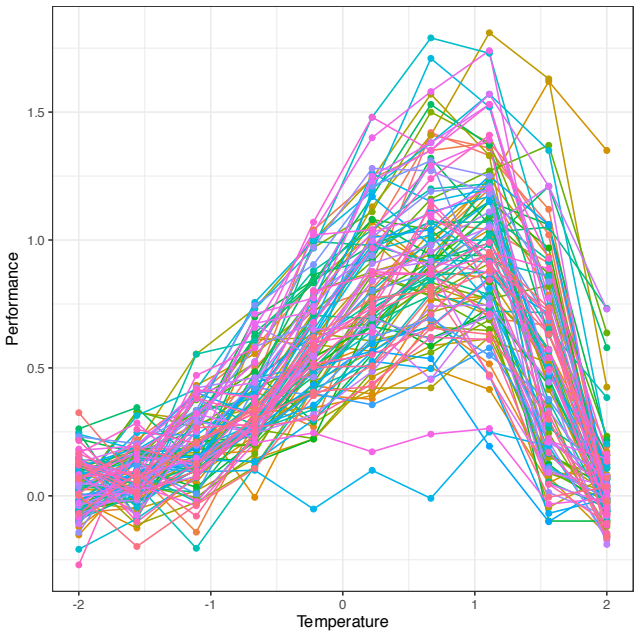

Figure 6: Dragons thermal performance, measured as locomotive performance, according to the experimental test temperature

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 6 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 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_quad)
```

```
 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 7):

```
plot(model_tpc_quad)
```

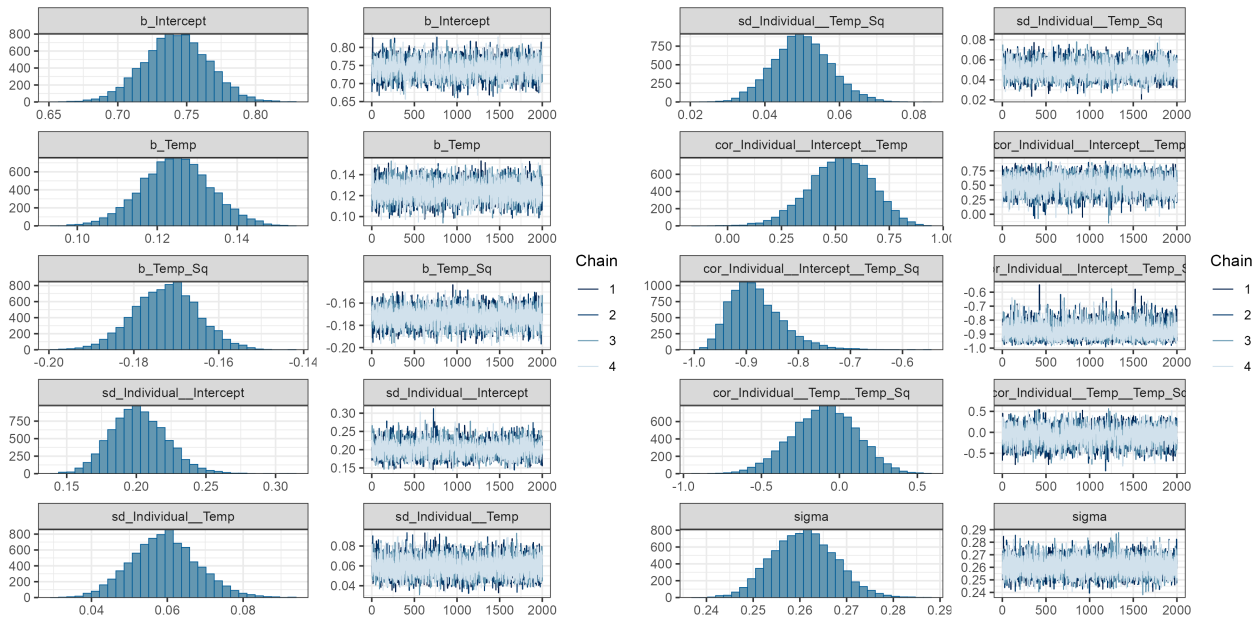

Figure 7: Plot of the `mod_tpc_quad` model. Parameters starting with "b" are the fixed effects parameters of the model, and parameters starting with "sd" are the standard deviation of the random effects. The parameter "sigma" is the residual standard deviation.

Looking at the model fit We can superimpose the predictions from the quadratic model over the actual reaction norms to visualise how good the fit is to the data (see the results in):

```
tbl_tpc_mod_quad <-
    tbl_dragon_ds |>
    mutate(Predict = predict(model_tpc_quad, 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_tpc <-
    p_tpc +
    geom_ribbon(data = tbl_tpc_mod_quad,
                mapping = aes(x = Temp, ymin = Predict_Low, ymax = Predict_Up),
                alpha = 0.3) +
    geom_line(data = tbl_tpc_mod_quad,
              mapping = aes(x = Temp, y = Predict),
              linewidth = 1)
```

Clearly, the fit is not great (notice also the strongest uncertainty than for aggressiveness), but it does get most of the variation in the reaction norm. We will see how we can quantify this in a more precise way using M^2_{Plas} in a bit below.

▸ 3.2.3 A first variance decomposition

Getting the posterior distributions of the parameters We can obtain the full posterior distribution of the parameters the same way as we did for the aggressiveness data[7]:

[7]Note that we will skip using point estimates here, as using the full posterior distribution is generally better, notably because we can assess the uncertainty surrounding our variance decomposition estimates

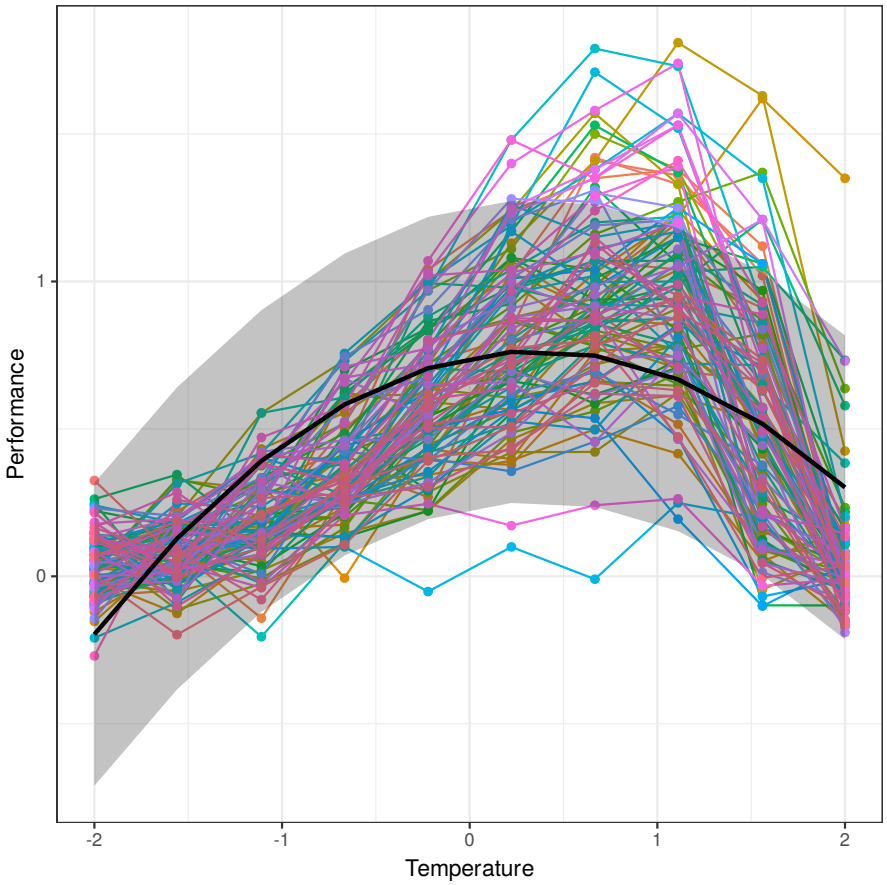

Figure 8: Fit of the quadratic model of the thermal performance from `mod_tpc_quad`, superimposed over the individual data.

```
# Getting the design matrix
design_mat <- model.matrix(Performance ~ Temp + Temp_Sq, data = tbl_dragon_ds)

# Getting the error variance-covariance matrix S_theta
S_theta_tpc <- vcov(model_tpc_quad)
rownames(S_theta_tpc) <- colnames(S_theta_tpc) <- c("a", "b", "c")

# Getting the fixed effects from the model (with the whole posterior distribution)
theta_post_tpc <- fixef(model_tpc_quad, summary = FALSE)
colnames(theta_post_tpc) <- c("a", "b", "c")
# Getting the G-matrix from the random effects variances-covariances
G_post_tpc <-
    VarCorr(model_tpc_quad, summary = FALSE)[["Individual"]][["cov"]] |>
    apply(1, \(mat_) { mat_ }, simplify = FALSE) |>
    map(\(mat_) { rownames(mat_) <- colnames(mat_) <- c("a", "b", "c"); return(mat_) })

# Creating a posterior sample using the posterior package
post_tpc <- as_draws_df(theta_post_tpc)
post_tpc[["G"]] <- G_post_tpc
post_tpc[["theta"]] <-
    post_tpc |>
    select(a:c) |>
    apply(1, \(vec_) { vec_ }, simplify = FALSE)
# Subsetting the iterations to 1000
post_tpc <- thin_draws(post_tpc, thin = nrow(theta_post_tpc) / 1000)
# Keep the iteration/chain info to create new posterior objects
post_tpc_info <- select(post_tpc, starts_with("."))
```

We did everything here at once, but the steps are more detailed for the aggressiveness trait in subsubsection 3.1.4.

Decomposing the average reaction norm variance We used a quadratic function, but we know that it is unlikely that the reaction norm curve truly is quadratic, so, we cannot use the π-decomposition in this case. We will thus use the φ-decomposition for good this time:

```
post_plas_tpc_quad <-
    post_tpc |>
    select(a, b, c) |>
    apply(1, \(th_) rn_phi_decomp(theta = th_, X = design_mat, S = S_theta_tpc)) |>
    bind_rows()  |>
    select(where(\(col_) { abs(mean(col_)) > 10^-5 })) |>
    cbind(post_tpc_info) |>
    as_draws_df()

summarise_draws(post_plas_tpc_quad)
```

```
# 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   0.0867 0.0865 0.00655 0.00631 0.0761 0.0980  1.00     946.     981.
2 Phi_b    0.290  0.290  0.0340  0.0330  0.234  0.352   1.00     957.     772.
3 Phi_c    0.710  0.710  0.0340  0.0330  0.648  0.766   1.00     957.     772.
```

We cannot directly interpret the φ_b and φ_c estimates in terms of the contribution of slope (π_{Sl}) and curvature (π_{Cv}) in the "geometric" sense of the term, because the environment is not normally distributed. But there's another problem: given that the quadratic curve does not entirely follow the reaction norms, we do not know whether we can trust the estimation of V_{Plas}, so we might want to fit a more applicable model to the data before we analyse anything.

▸ 3.2.4 Fitting a character-state model to the data

Running and checking the model The character-state model takes advantage of our discretised environments to analyse the environment as a categorical factor, rather than a continuous one. This way, there is no need to parametrised a curve in advance for the model, as each environmental value will have its own parameter. To do so, we will change the formula to define the model, using the environment name column (`Name_Env`), and pass it to `brms`:

```
form_cs <- brmsformula(Performance ~ 0 + Name_Env + (0 + Name_Env | Individual))
model_cs_tpc <-
    brm(formula   = form_cs,
        data      = tbl_dragon_ds,
        save_pars = save_pars(group = FALSE),
        chains    = n_chains,
        cores     = n_chains,
        seed      = seed,
        iter      = 6000,
        warmup    = 1000,
        thin      = 1)
summary(model_cs_tpc)
```

```
 Family: gaussian
  Links: mu = identity; sigma = identity
Formula: Performance ~ 0 + Name_Env + (0 + Name_Env | Individual)
   Data: tbl_dragon_ds (Number of observations: 1000)
  Draws: 4 chains, each with iter = 6000; warmup = 1000; thin = 1;
         total post-warmup draws = 20000

Multilevel Hyperparameters:
~Individual (Number of levels: 100)
                                Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sd(Name_EnvEnv_01)                  0.06      0.02     0.02     0.09 1.00     1765     3251
sd(Name_EnvEnv_02)                  0.06      0.01     0.03     0.09 1.00     2718     5557
sd(Name_EnvEnv_03)                  0.11      0.01     0.09     0.14 1.00     5238    11208
sd(Name_EnvEnv_04)                  0.14      0.01     0.11     0.16 1.00     7023    13178
sd(Name_EnvEnv_05)                  0.19      0.01     0.16     0.22 1.00     7363    12646
sd(Name_EnvEnv_06)                  0.24      0.02     0.21     0.28 1.00     6553    11333
sd(Name_EnvEnv_07)                  0.28      0.02     0.24     0.32 1.00     6239     9848
sd(Name_EnvEnv_08)                  0.30      0.02     0.26     0.34 1.00     6859    10767
sd(Name_EnvEnv_09)                  0.39      0.03     0.33     0.44 1.00     8810    11828
sd(Name_EnvEnv_10)                  0.20      0.02     0.17     0.24 1.00     9102    12198
cor(Name_EnvEnv_01,Name_EnvEnv_02)  0.12      0.24    -0.34     0.58 1.00     3369     6169
cor(Name_EnvEnv_01,Name_EnvEnv_03)  0.22      0.19    -0.16     0.58 1.00     2173     3928
cor(Name_EnvEnv_02,Name_EnvEnv_03)  0.30      0.18    -0.07     0.64 1.00     2576     4378
cor(Name_EnvEnv_01,Name_EnvEnv_04)  0.40      0.17     0.05     0.71 1.00     2553     4913
```

```
cor(Name_EnvEnv_02,Name_EnvEnv_04)  0.41      0.17     0.07     0.72 1.00     2517     4931
cor(Name_EnvEnv_03,Name_EnvEnv_04)  0.61      0.11     0.37     0.80 1.00     5894     8772
cor(Name_EnvEnv_01,Name_EnvEnv_05)  0.34      0.16     0.01     0.66 1.00     1855     2940
cor(Name_EnvEnv_02,Name_EnvEnv_05)  0.46      0.15     0.15     0.73 1.00     2523     4113
cor(Name_EnvEnv_03,Name_EnvEnv_05)  0.65      0.10     0.44     0.82 1.00     4754     7894
cor(Name_EnvEnv_04,Name_EnvEnv_05)  0.72      0.08     0.55     0.86 1.00     4598     9687
cor(Name_EnvEnv_01,Name_EnvEnv_06)  0.32      0.16     0.02     0.63 1.00     1693     2350
cor(Name_EnvEnv_02,Name_EnvEnv_06)  0.40      0.15     0.09     0.68 1.00     2141     3316
cor(Name_EnvEnv_03,Name_EnvEnv_06)  0.66      0.09     0.48     0.82 1.00     4355     7048
cor(Name_EnvEnv_04,Name_EnvEnv_06)  0.75      0.07     0.60     0.87 1.00     5458    10968
cor(Name_EnvEnv_05,Name_EnvEnv_06)  0.89      0.04     0.80     0.95 1.00     5881    10335
cor(Name_EnvEnv_01,Name_EnvEnv_07)  0.29      0.16    -0.03     0.60 1.00     1668     2143
cor(Name_EnvEnv_02,Name_EnvEnv_07)  0.30      0.15    -0.01     0.59 1.00     2130     3590
cor(Name_EnvEnv_03,Name_EnvEnv_07)  0.48      0.10     0.27     0.67 1.00     4374     7448
cor(Name_EnvEnv_04,Name_EnvEnv_07)  0.64      0.08     0.48     0.78 1.00     6163    11145
cor(Name_EnvEnv_05,Name_EnvEnv_07)  0.79      0.05     0.68     0.88 1.00     5849    10299
cor(Name_EnvEnv_06,Name_EnvEnv_07)  0.86      0.04     0.78     0.93 1.00     5889    11602
cor(Name_EnvEnv_01,Name_EnvEnv_08)  0.15      0.16    -0.16     0.48 1.00     1715     2172
cor(Name_EnvEnv_02,Name_EnvEnv_08)  0.06      0.16    -0.25     0.37 1.00     1956     2961
cor(Name_EnvEnv_03,Name_EnvEnv_08)  0.45      0.10     0.24     0.64 1.00     4189     8169
cor(Name_EnvEnv_04,Name_EnvEnv_08)  0.46      0.09     0.26     0.63 1.00     5904    10000
cor(Name_EnvEnv_05,Name_EnvEnv_08)  0.61      0.07     0.45     0.74 1.00     6915    11837
cor(Name_EnvEnv_06,Name_EnvEnv_08)  0.76      0.05     0.65     0.85 1.00     9471    14779
cor(Name_EnvEnv_07,Name_EnvEnv_08)  0.86      0.04     0.79     0.92 1.00     8228    14263
cor(Name_EnvEnv_01,Name_EnvEnv_09)  0.02      0.17    -0.32     0.35 1.00     1978     3020
cor(Name_EnvEnv_02,Name_EnvEnv_09) -0.22      0.16    -0.52     0.10 1.00     1890     3939
cor(Name_EnvEnv_03,Name_EnvEnv_09) -0.14      0.12    -0.36     0.09 1.00     4826     8989
cor(Name_EnvEnv_04,Name_EnvEnv_09) -0.17      0.11    -0.37     0.05 1.00     5328    11048
cor(Name_EnvEnv_05,Name_EnvEnv_09)  0.02      0.10    -0.17     0.22 1.00     7439    10641
cor(Name_EnvEnv_06,Name_EnvEnv_09)  0.20      0.09     0.02     0.37 1.00     9381    13733
cor(Name_EnvEnv_07,Name_EnvEnv_09)  0.47      0.08     0.31     0.61 1.00    10259    13876
cor(Name_EnvEnv_08,Name_EnvEnv_09)  0.65      0.06     0.52     0.75 1.00    11453    13792
cor(Name_EnvEnv_01,Name_EnvEnv_10) -0.09      0.18    -0.44     0.27 1.00     2048     3081
cor(Name_EnvEnv_02,Name_EnvEnv_10) -0.04      0.17    -0.38     0.29 1.00     1877     2577
cor(Name_EnvEnv_03,Name_EnvEnv_10) -0.18      0.12    -0.42     0.06 1.00     4643     9162
cor(Name_EnvEnv_04,Name_EnvEnv_10) -0.17      0.11    -0.38     0.06 1.00     6007    10506
cor(Name_EnvEnv_05,Name_EnvEnv_10) -0.15      0.10    -0.35     0.06 1.00     8391    13452
cor(Name_EnvEnv_06,Name_EnvEnv_10)  0.01      0.10    -0.19     0.21 1.00     9588    14460
cor(Name_EnvEnv_07,Name_EnvEnv_10)  0.10      0.10    -0.10     0.29 1.00    11711    14658
cor(Name_EnvEnv_08,Name_EnvEnv_10)  0.17      0.10    -0.03     0.36 1.00    13629    16173
cor(Name_EnvEnv_09,Name_EnvEnv_10)  0.54      0.08     0.37     0.68 1.00    13426    16683

Regression Coefficients:
              Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Name_EnvEnv_01     0.04      0.01     0.02     0.06 1.00    18101    15540
Name_EnvEnv_02     0.08      0.01     0.06     0.10 1.00    16308    14263
Name_EnvEnv_03     0.20      0.01     0.17     0.22 1.00     7720    12567
Name_EnvEnv_04     0.37      0.02     0.34     0.40 1.00     6724    11503
Name_EnvEnv_05     0.60      0.02     0.56     0.64 1.00     5497    10036
Name_EnvEnv_06     0.81      0.03     0.76     0.87 1.00     4865     9047
```

```
Name_EnvEnv_07     0.95      0.03     0.89     1.01 1.00     4923     8811
Name_EnvEnv_08     0.98      0.03     0.92     1.04 1.00     5091     9087
Name_EnvEnv_09     0.53      0.04     0.45     0.61 1.00     7929    11100
Name_EnvEnv_10     0.05      0.02     0.01     0.10 1.00    10566    12887

Further Distributional Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sigma     0.09      0.00     0.08     0.10 1.01     1184      582

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).
```

Note that we had to increase the number of iterations to run the model, because the residual standard deviation had too small efficient sample size and too high $\hat{R}$. The high number of parameters are due to the fact that, as part of the character-state model, we now infer a 10×10 G matrix, with additive genetic variances and covariances across all pairs of environments. We can also graphically check that everything went smoothly, but we will only select a few parameters to not overwhelm the graphic (see Figure 9):

```
# We select everything starting with a "b_" (fixed effects) and the residual sd
plot(model_cs_tpc, variable = c("^b_", "sigma"), regex = TRUE)
```

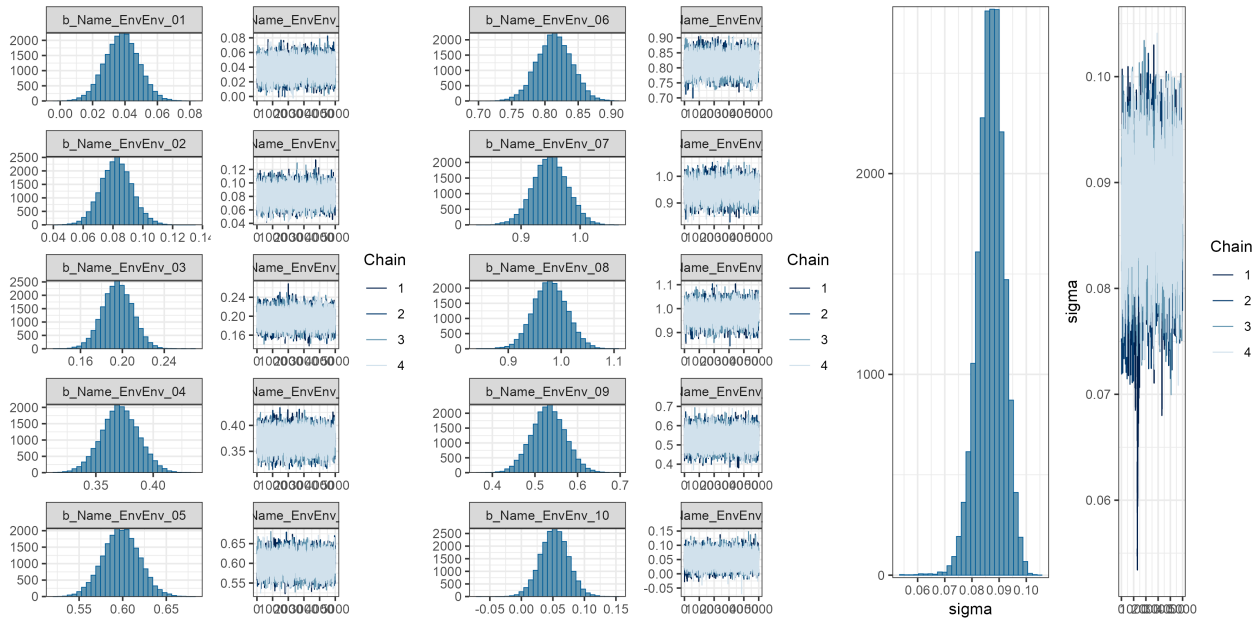

Figure 9: Plot of the `mod_cs_tpc` model. Parameters starting with "b" are the fixed effects parameters of the model. The parameter "sigma" is the residual standard deviation.

Because the character-state does not make explicit assumption about the shape of the curve of the reaction norm, we can see the fit of each point to the global curve is better than the quadratic curve (see Figure 10):

```
tbl_tpc_mod_cs <-
    tbl_dragon_ds |>
    mutate(Predict = predict(model_cs_tpc, 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_tpc_cs <-
    p_tpc +
    geom_pointrange(data = tbl_tpc_mod_cs,
                    mapping = aes(x = Temp, y = Predict, ymin = Predict_Low, ymax = Predict_Up),
                    size = 1, linewidth = 1, shape = 4)
```

Extracting the parameters from the model As always, the first thing to do is to extract the parameters of interest from the model. Since the character-state model is quite straightforward, we can directly extract V_{Plas} (which is simply the variance of the population-level effects) and the G-matrix. We will directly extract their posterior distribution this time:

```
# Getting the uncertainty on the parameters
var_uncert_cs_tpc <-
    vcov(model_cs_tpc) |>
    diag() |>
    mean()
```

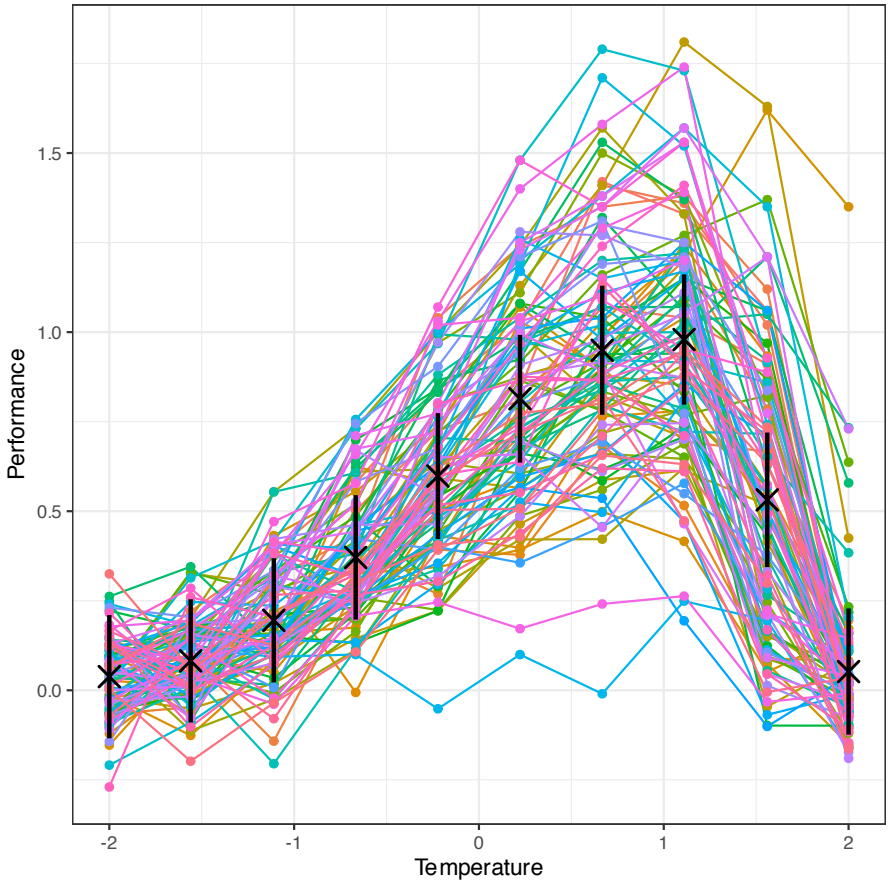

Figure 10: Fit of the character-state model of the thermal performance from `mod_cs_tpc`, superimposed over the individual data.

```
# Computing V_plas
post_theta_cs <-
    fixef(model_cs_tpc, summary = FALSE) |>
    as_draws_df()
var_plas_cs <-
    post_theta_cs |>
    select(starts_with("Name")) |>
    as.matrix() |>
    rowVars()

# Correcting for the uncertainty
post_cs <-
    data.frame(V_Plas = var_plas_cs - var_uncert_cs_tpc) |>
    cbind(select(post_theta_cs, starts_with("."))) |>
    as_draws_df()

# Getting the G-matrix
post_cs[["G"]] <-
    VarCorr(model_cs_tpc, summary = FALSE)[["Individual"]][["cov"]] |>
    apply(1, \(mat_) { mat_ }, simplify = FALSE)
# Getting the residual variance
post_cs[["V_R"]] <-
    VarCorr(model_cs_tpc, summary = FALSE)[["residual__"]][["sd"]][ , 1]^2
```

And of course, we will subset the iterations to a thousands, once again:

```
post_cs <- thin_draws(post_cs, thin = length(var_plas_cs) / 1000)
post_cs_info <- select(post_cs, starts_with("."))
```

Let's look at the output for V_{Plas}:

```
summarise_draws(subset_draws(post_cs, variable = "V_Plas"))
```

```
# A tibble: 1 × 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   0.136  0.136 0.00841 0.00838 0.122 0.150 0.999     845.    1011.
```

As expected, the variance due to the average reaction norm obtained from the character-state ($V_{\text{Plas}} = 0.136$) is bigger than the one obtained from the quadratic model ($V_{\text{Plas}} = 0.087$), so that we roughly have $M^2_{\text{Plas}} = 0.087/0.136 = 0.64$. We will see later how to compute M^2_{Plas} more properly, using the full posterior distribution.

Computing the additive genetic variances from the character-state model We can compute the additive genetic variances V_{Add}, V_{A} and $V_{\text{A}\times\text{E}}$ directly from the G-matrix when using a character-state model. V_{Add} is the average of the diagonal elements, while V_{A} is the average of all elements of the G-matrix. We can then simply obtain $V_{\text{A}\times\text{E}}$ using the difference between the two variances: $V_{\text{A}\times\text{E}} = V_{\text{Add}} - V_{\text{A}}$. This is implemented in the `rn_cs_gen()` function of the `Reacnorm` package:

```
post_gen_cs <-
    map(post_cs[["G"]], rn_cs_gen, .progress = TRUE) |>
    bind_rows()  |>
    select(where(\(col_) { abs(mean(col_)) > 10^-5 })) |>
```

```
    cbind(post_tpc_info) |>
    as_draws_df()

summarise_draws(post_gen_cs)
```

```
# A tibble: 4 × 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    0.0488 0.0485 0.00461 0.00444 0.0419 0.0567  1.00     959.     949.
2 V_A      0.0182 0.0180 0.00235 0.00225 0.0146 0.0222  1.00    1011.     968.
3 V_AxE    0.0306 0.0305 0.00291 0.00265 0.0262 0.0356  1.00     922.     742.
4 N_eff    1.69   1.69   0.108   0.112   1.53   1.88    1.00    1011.     987.
```

The function also outputs n_{eff}, the efficient number of dimensions. However, please keep in mind that this value seems to suffer from underestimation, as shown in the companion paper. Nevertheless, the number is relatively low compared to the total number of environment (10), suggesting a rather high level of constraints in the genetic variation of the reaction norm across environments. This is also supported by the additive genetic variance decomposition of the reaction norm, with almost two times higher additive genetic variance in plasticity ($V_{\text{A×E}} = 0.031$) than the marginal additive genetic variance of the trait ($V_{\text{A}} = 0.018$).

Computing the variance-standardised parameters We can compute the variance-standardised parameters from the character-state pretty much the same we did it for the aggressiveness trait (see Figure 9):

```
post_var_tpc_cs <-
    bind_draws(post_cs, post_gen_cs) |>
    subset_draws(variable = c("V_Plas", "V_Add", "V_A", "V_AxE", "V_R")) |>
    mutate_variables(V_Tot = V_Plas + V_Add + V_R)

post_std_tpc_cs <-
    post_var_tpc_cs |>
    transmute(P2    = V_Plas / V_Tot,
              H2_RN = V_Add / V_Tot,
              H2    = V_A / V_Tot,
              H2_I  = V_AxE / V_Tot,
              T2    = (V_Plas + V_Add) / V_Tot) |>
    cbind(post_cs_info) |>
    as_draws_df()

summarise_draws(post_std_tpc_cs)
mcmc_trace(post_std_tpc_cs)
mcmc_areas(post_std_tpc_cs,
           prob = 0.95,
           area_method = "scaled height")
```

```
# A tibble: 5 × 10
  variable   mean median      sd     mad     q5   q95  rhat ess_bulk ess_tail
  <chr>     <dbl>  <dbl>   <dbl>   <dbl>  <dbl> <dbl> <dbl>    <dbl>    <dbl>
1 P2        0.706  0.708  0.0207  0.0198  0.669  0.738  1.00     894.     713.
2 H2_RN     0.254  0.253  0.0211  0.0202  0.222  0.291  1.00     916.     836.
```

```
3 H2       0.0947 0.0941 0.0111  0.0106  0.0779 0.114  1.00    1009.     938.
4 H2_I     0.160  0.159  0.0138  0.0130  0.138  0.184  1.00     900.     806.
```

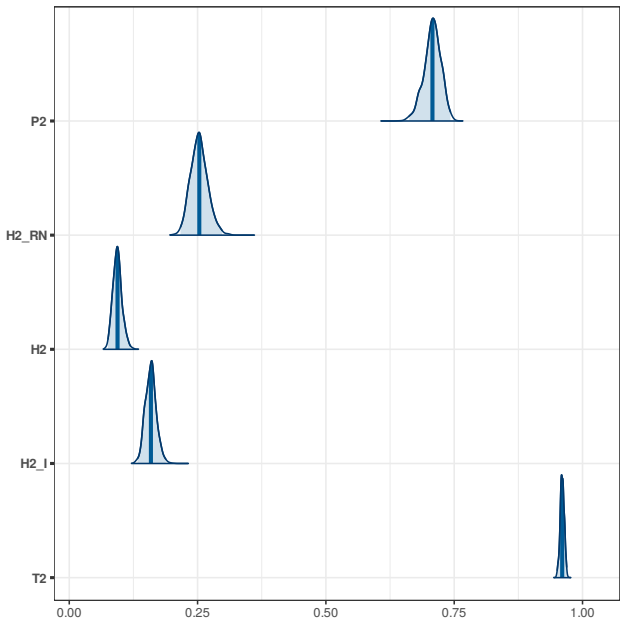

Figure 11: Posterior distribution of the variance-standardised estimates of our variance decomposition of the reaction norm of thermal performance, based on a character-state model.

▸ *3.2.5 A better variance decomposition, combining quadratic and character-state models*

Studying the average reaction norm Since we know the variances obtained from the character-state model are more trustworthy, we can use them for our variance decomposition. But at the same time, we would still like to be able to say how much of the variation we observe is explained by a first-order linear trend or a second-order one. To approximate such values, we can combine the estimates from the quadratic model with the variances (here V_{Plas}) obtained from the character-state model:

```
post_plas_tpc_withcs <-
    # Note that we get theta from the quadratic model,
    # but V_Plas from the character-state one
    map2(post_tpc[["Theta"]], post_cs[["V_Plas"]],
         \(th_, v_) rn_phi_decomp(theta   = th_,
                                  X       = design_mat,
                                  S       = S_theta_tpc,
                                  v_plas = v_),
         .progress = TRUE) |>
    bind_rows()  |>
    select(where(\(col_) { abs(mean(col_)) > 10^-5 })) |>
    cbind(post_tpc_info) |>
    as_draws_df()

summarise_draws(post_plas_tpc_withcs)
mcmc_trace(post_plas_tpc_withcs)
mcmc_areas(post_plas_tpc_withcs,
           regex_pars = "^V",
```

```
          prob = 0.95,
          area_method = "scaled height") /
    mcmc_areas(post_plas_tpc_withcs,
               regex_pars = "^[^V]",
               prob = 0.95,
               area_method = "scaled height") +
    plot_layout(heights = c(1, 3))
```

```
# A tibble: 4 × 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   0.136  0.136 0.00841 0.00838 0.122 0.150 1.00      886.    1051.
2 Phi_b    0.186  0.185 0.0287  0.0286  0.141 0.237 1.00      989.     916.
3 Phi_c    0.456  0.454 0.0487  0.0473  0.380 0.538 0.998    1004.    1016.
4 M2       0.642  0.638 0.0622  0.0624  0.541 0.748 1.00     1031.    1081.
```

This output (see also Figure 12) is different from the one we obtained directly with the quadratic model. First, the value for V_{Plas} (which was not computed here, but directly taken from `post_cs`) is larger here. Second, and a consequence, the values for φ_b and φ_c are smaller, and do not sum to 1 any more. Of course, however, their relative values (i.e. φ_b/φ_c) is conserved. Third, the function `rn_phi_decomp()` this time returned a new value: the ratio M^2_{Plas} of the estimation of V_{Plas} as estimated from the quadratic model to the estimation of V_{Plas} from the character-state. This value measures how well the quadratic model was a good approximation of the reaction norm. Here, $M^2_{\text{Plas}} = 0.64$[8], which is not extremely great (i.e. the fit is clearly not perfect and we should not have used the values from the quadratic model directly), but not too bad either (i.e. the combination with character-state as we're doing is still informative). Note that, because the character-state does not make explicit assumptions about the shape of the reaction norm and is thus more "encompassing", we expect $M^2_{\text{Plas}} \leq 1$ in general.

Studying the additive genetic variation We can have the same hybrid approach to the additive genetic variance decomposition, by providing the parameters values from the quadratic model, but the estimated variances from the character-state. To do so, we will use the `add_vars` argument from the `rn_gen_decomp()` function:

```
# Adding a column containing the three additive genetic variances
# to the character-state posterior draws
post_gen_cs[["Add_Vars"]] <-
    post_gen_cs |>
    select(starts_with("V")) |>
    apply(1, \(row_) { c(row_) }, simplify = FALSE)

# Calling rn_gen_decomp(), but provided values from the character-state
# to add_vars
post_gen_tpc_withcs <-
    pmap(list(th_ = post_tpc[["Theta"]],
              G_  = post_tpc[["G"]],
              v_  = post_gen_cs[["Add_Vars"]]),
         \(th_, G_, v_) rn_gen_decomp(theta    = th_,
                                      G_theta  = G_,
                                      X        = design_mat,
```

[8] Note that we were bad at all with our little computation above

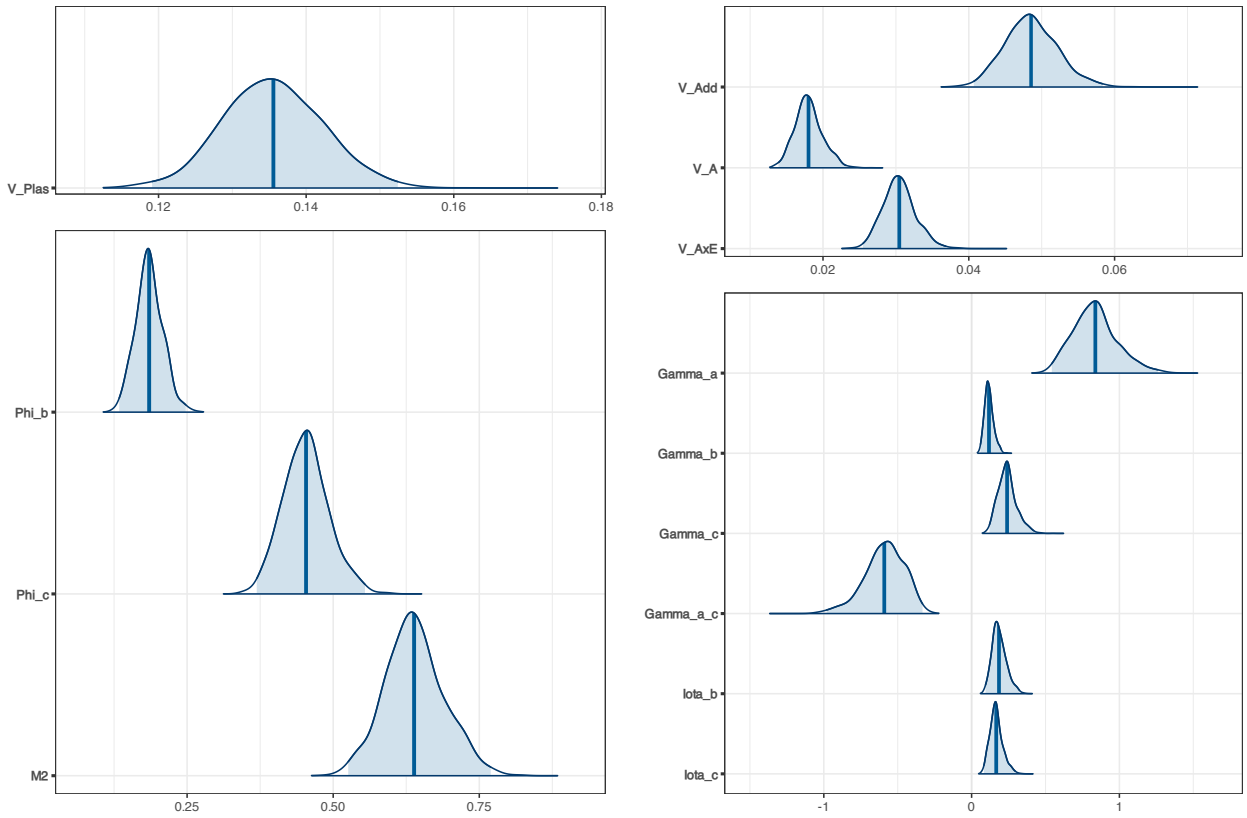

Figure 12: Posterior distribution of the variance decomposition of the reaction norm of aggressiveness, based on a quadratic model.

```
                                    add_vars = v_),
        .progress = TRUE) |>
    bind_rows()  |>
    select(where(\(col_) { abs(mean(col_)) > 10^-5 })) |>
    cbind(post_tpc_info) |>
    as_draws_df()

summarise_draws(post_gen_tpc_withcs)
mcmc_trace(post_gen_tpc_withcs)
mcmc_areas(post_gen_tpc_withcs,
           regex_pars = "^V",
           prob = 0.95,
           area_method = "scaled height") /
    mcmc_areas(post_gen_tpc_withcs,
               regex_pars = "^[^V]",
               prob = 0.95,
               area_method = "scaled height") +
    plot_layout(heights = c(3, 6))
```

```
# 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      0.0488  0.0485 0.00461 0.00444  0.0419  0.0567  1.00     959.     949.
2 V_A        0.0182  0.0180 0.00235 0.00225  0.0146  0.0222  1.00    1011.     968.
3 V_AxE      0.0306  0.0305 0.00291 0.00265  0.0262  0.0356  1.00     922.     742.
```

```
4 Gamma_a     0.852   0.837  0.185   0.177    0.582   1.19    1.00     944.     913.
5 Gamma_b     0.121   0.118  0.0372  0.0358   0.0669  0.190   1.00     922.     888.
6 Gamma_c     0.249   0.240  0.0789  0.0736   0.136   0.392   1.00    1058.     993.
7 Gamma_a_c -0.607  -0.591  0.168   0.164   -0.917  -0.367   1.00    1004.     969.
8 Iota_b      0.193   0.185  0.0597  0.0562   0.106   0.305   1.00     946.     861.
9 Iota_c      0.173   0.167  0.0554  0.0516   0.0955  0.276   1.00    1039.     993.
```

See Figure 12 for the posterior distributions. Of course, the values for V_{Add}, V_{A} and $V_{\text{A}\times\text{E}}$ are the same as the one we computed from the character-state and, they were directly used and not re-computed. Note also that, because we used the variances from the character-state to scale them (as we did for φ above), the γ's and ι's do not sum to 1 either in this case. All of this is a bit "hacky" and not as good as finding a proper curve, fitting well our reaction norm, as we will see now.

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

▸ *3.3.1 Running a non-linear model*

The Gaussian-Gompertz function Looking at Figure 6, the shape seems to follow the typical asymmetrical quasi-bell shape of thermal performance traits. A commonly used function to study such kind of traits is the Gaussian-Gompertz function. It is a relatively complex function that depends on 4 parameters (highlighted below):

$$\hat{z} = C_{\max} \exp\left(-\exp\left(\rho\left(\varepsilon - \varepsilon_0\right) - 6\right) - \sigma\left(\varepsilon - \varepsilon_0\right)^2\right) \tag{1}$$

However, for the sake of simplicity, we will only make two of them ($C_{\max}$ and ε_0) vary genetically[9].

Preparing the model Running a non-linear model in `brms` is relatively straightforward, but it does require new elements of syntax:

```
form_nl <- brmsformula(Performance ~ cmax * exp(
                                        - exp(rho * (Temp - xopt) - 6) -       # Gompertz part
                                          sigmagaus * (Temp - xopt)^2          # Gaussian part
                                    ),
                       cmax + xopt ~ 1 + (1 | ID1 | Individual),
                       rho + sigmagaus ~ 1,
                       nl = TRUE)
```

Starting from the end, notice we set the argument `nl` to `TRUE`, telling `brmsformula()` that we want to set up a non-linear model. Then (still from the end), we set up two groups of parameters: `rho` and `sigmagaus` will be inferred, but fixed across individuals; while `cmax` and `xopt` will be allow to vary across individuals. The `ID1` part is just a placeholder (it could be any string of character) to tell `brms` that we want to infer the covariances between `cmax` and `xopt`. Finally (at the top), we define the equation of the model, linking the response variable `Performance` with the environmental variable `Temp`, following Equation 1. While we were using the default priors until now, the situation is different for a non-linear model, because it is hard for `brms` to come up with relevant priors for the non-linear parameters. So, we will help it by providing priors for the parameters that cannot take negative values. Although we could come up with smarter priors, we will simply here use uniform priors for those parameters, specifying a higher bound far away enough from the values that we expect to be realistic:

[9] If you are curious at to what it looks like in practice, you can spoil the end for yourself and look directly at Figure 14.

```
prior_nl <-
    prior(uniform(0, 10), nlpar = "cmax", lb = 0, ub = 100) +
    prior(uniform(0, 100), nlpar = "rho", lb = 0, ub = 100) +
    prior(uniform(0, 10), nlpar = "sigmagaus", lb = 0, ub = 10)
```

Another thing that is now required and that is difficult for `brms` to figure out are the starting values for the non-linear parameters, which we will thus provide based on ballpark idea of what their value should be:

```
inits <- rep(list(list(b_cmax      = array(data = 1),
                       b_xopt      = array(data = 0.9),
                       b_rho       = array(data = 8),
                       b_sigmagaus = array(data = 0.4))), 4)
```

Finally, we will use an increased total number of iterations:

```
# Total number of iterations
n_iter_nl <- 7000
# Number of iterations that will be discarded for the warm-up
n_warm_nl <- 1000
# Thinning interval
n_thin_nl <- 1
```

Now, we are ready to run the model!

Running the model Now that we have prepared everything, running the model is very much like the linear instance (although note that we now provide `prior` and `init`):

```
model_nl_tpc <-
    brm(formula   = form_nl,
        data      = tbl_dragon_ds,
        save_pars = save_pars(group = FALSE),
        chains    = n_chains,
        cores     = n_chains,
        seed      = seed,
        init      = inits,
        prior     = prior_nl,
        iter      = n_iter_nl,
        warmup    = n_warm_nl,
        thin      = n_thin_nl)
```

This might take a bit longer than the other models, but not by much.

Checking the model Let's look at the model estimates:

```
summary(model_nl_tpc)
```

```
 Family: gaussian
  Links: mu = identity; sigma = identity
Formula: Performance ~ cmax * exp(-exp(rho * (Temp - xopt) - 6) - sigmagaus * (Temp - xopt)^2)
         cmax ~ 1 + (1 | ID1 | Individual)
         xopt ~ 1 + (1 | ID1 | Individual)
         rho ~ 1
         sigmagaus ~ 1
```

```
   Data: tbl_dragon_ds (Number of observations: 1000)
  Draws: 4 chains, each with iter = 7000; warmup = 1000; thin = 1;
         total post-warmup draws = 24000

Multilevel Hyperparameters:
~Individual (Number of levels: 100)
                                  Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sd(cmax_Intercept)                    0.32      0.02     0.28     0.38 1.00     1777     3667
sd(xopt_Intercept)                    0.21      0.02     0.18     0.24 1.00     2524     5804
cor(cmax_Intercept,xopt_Intercept)    0.24      0.10     0.02     0.43 1.00     1459     2915

Regression Coefficients:
                   Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
cmax_Intercept         1.01      0.03     0.95     1.08 1.01      813     1833
xopt_Intercept         0.95      0.03     0.90     1.01 1.00     1994     5184
rho_Intercept          8.37      0.27     7.88     8.93 1.00     9192    13020
sigmagaus_Intercept    0.38      0.01     0.36     0.40 1.00    10942    14635

Further Distributional Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sigma     0.10      0.00     0.10     0.11 1.00    18600    18545

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).
```

The $\hat{R}$ and effective sample size for all parameters are acceptable, and the point values and credible intervals for them seem coherent. We can verify also graphically that everything went smoothly by looking at their trace and posterior distributions (see Figure 13):

```
plot(model_nl_tpc)
```

Finally, we can also look at the fit of the model to the raw data, by placing the model predictions over the raw phenotypes (see Figure 14):

```
tbl_tpc_mod <-
    tbl_dragon_ds |>
    mutate(Predict = predict(model_nl_tpc, 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_tpc <-
    p_tpc +
    geom_ribbon(data = tbl_tpc_mod,
                mapping = aes(x = Temp, ymin = Predict_Low, ymax = Predict_Up),
                alpha = 0.3) +
```

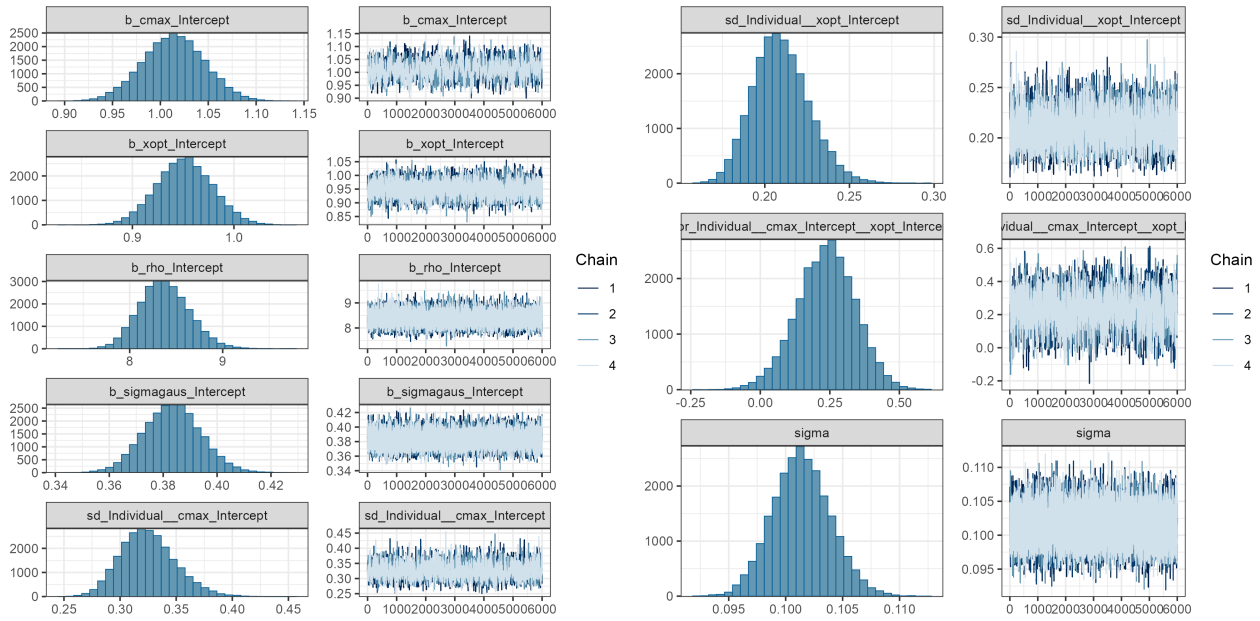

Figure 13: Plot of the `mod_nl_tpc` model. Parameters starting with "b" are the fixed effects of the non-linear parameters of the model, and parameters starting with "sd" are the standard deviation of the random effects of the non-linear parameters. The parameter "sigma" is the residual standard deviation.

```
    geom_line(data = tbl_tpc_mod,
              mapping = aes(x = Temp, y = Predict),
              linewidth = 1)
```

The fit this time is much better than with the quadratic curve[10]. So, with this much better curve, we should be able to readily apply our variance decomposition!

▸ *3.3.2 Decomposing the variance of a non-linear model*

Extracting the parameters The code used to extract the estimation of the parameters of interest for the variance decomposition is surprisingly similar to the linear case. We just to do a bit more work regarding the names:

```
theta_post_nl_tpc <- fixef(model_nl_tpc, summary = FALSE)
# We remove the "_Intercept" part of the name
colnames(theta_post_nl_tpc) <- str_remove(colnames(theta_post_nl_tpc), "_Intercept")
head(theta_post_nl_tpc)
```

```
    variable
draw      cmax      xopt       rho sigmagaus
   1 0.9968932 0.9641778 8.219535 0.3843630
   2 1.0087171 0.9462906 8.454820 0.3873613
   3 1.0038337 0.9577721 8.381722 0.3799357
   4 1.0123249 0.9456947 8.433239 0.3816431
   5 0.9988809 0.9244892 8.479489 0.3733265
   6 1.0398035 0.9162629 8.312176 0.3971904
```

```
G_post_nl_tpc <-
    VarCorr(model_nl_tpc, summary = FALSE)[["Individual"]][["cov"]] |>
    apply(1, \(mat_) { mat_ }, simplify = FALSE) |>
```

[10] Well, that is to be expected, because this happen to be exactly the true curve of reaction norms.

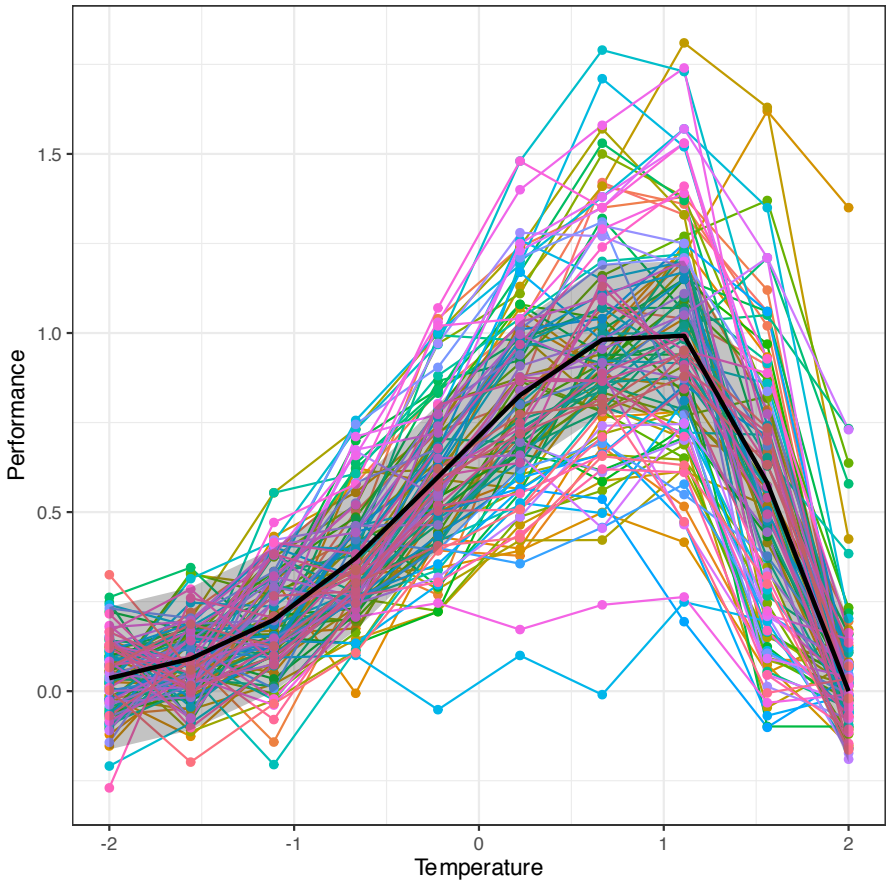

Figure 14: Thermal performance individual data, with the non-linear reaction norm predicted by the `mod_tpc_nl` model.

```
    # Same here for "_Intercept" part of the name
    map(\(mat_) { rownames(mat_) <- colnames(mat_) <- str_remove(rownames(mat_), "_Intercept"); return(mat_
G_post_nl_tpc[[1]]
```

```
            cmax        xopt
cmax 0.102789698 0.001781295
xopt 0.001781295 0.028796905
```

```
vr_post_nl_tpc <-
    VarCorr(model_nl_tpc, summary = FALSE)[["residual__"]][["sd"]][ , 1]^2
```

The object `theta_post_nl_tpc` contains all the "fixed effect" part of the parameters estimation, while `G_post_nl_tpc` contains the variances and covariance estimates from their "random part". Remember that we only allowed `cmax` and `xopt` to vary genetically across individuals in the model. Because of that, our G-matrix is smaller than our parameter vector $\bar{\theta}$, but worry not, as `Reacnorm` will be able to account for this! It is especially important here to reduce the number of kept iterations to a thousand, because the functions that we will use rely (more) on numerical integration and are thus a bit slower:

```
post_nl_tpc <- as_draws_df(theta_post_nl_tpc)
post_nl_tpc[["G"]] <- G_post_nl_tpc
post_tpc[["Theta"]] <-
    post_tpc |>
    select(a:c) |>
    apply(1, \(vec_) { vec_ }, simplify = FALSE)
```

```
post_nl_tpc[["V_R"]] <- vr_post_nl_tpc
post_nl_tpc <- thin_draws(post_nl_tpc, thin = nrow(theta_post_nl_tpc) / 1000)
# Keep the iteration/chain info to create new posterior objects
post_nl_tpc_info <- select(post_nl_tpc, starts_with("."))
```

Generating the expression for the reaction norm curve Because the model in non-linear this time, `Reacnorm` has no idea what the assumed shape of the reaction norm was simply based on vector of parameters and the environmental values (i.e. the `design_matrix` we used before). This time, we need to be able to provide the functions with the shape we used for the reaction norm. To do so, we will have to generate an "expression" in R, which will refer to the environment as `x` and use **exactly the same parameter names as we did for** `brms`. This can be done quite easily in R using the `expression()` function:

```
gg_shape <- expression(
    cmax * exp(
        - exp(rho * (x - xopt) - 6) -
            sigmagaus * (x - xopt)^2
    )
)
```

We will also require a vector of unique environmental value that we will prepare:

```
vec_env <- unique(tbl_dragon_ds[["Temp"]])
```

Computing the variance of average reaction norm (and its decomposition?) This time, since the model is non-linear, we cannot compute the φ-decomposition using the `rn_phi_decomp()`. Also, because the environmental variable is a fixed, discretised variable, it does not follow a normal distribution, so we cannot properly compute the π-decomposition either[11]. Still, we can obtain V_{Plas} directly using the `rn_vplas()` function. To do so, we have to provide the shape of the reaction norm (with `gg_shape`) and the vector of environmental values (with `vec_env`) directly. Finally, we need to state to `Reacnorm` that the third (`rho`) and fourth (`sigmagaus`) values of the vector of parameters are not present in the G-matrix, which we do using the `fixed` parameter:

```
post_v_plas_nl_tpc <-
    map2(post_nl_tpc[["Theta"]], post_nl_tpc[["G"]],
         \(th_, G_) { data.frame(V_Plas = rn_vplas(theta    = th_,
                                                   G_theta  = G_,
                                                   env      = vec_env,
                                                   shape    = gg_shape,
                                                   fixed    = c(3, 4))) },
         .progress = TRUE) |>
    bind_rows() |>
    cbind(post_nl_tpc_info) |>
    as_draws_df()
summarise_draws(post_v_plas_nl_tpc)
```

```
# A tibble: 1 × 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   0.121  0.121 0.00801 0.00791 0.108 0.135  1.01     763.     689.
```

[11] Although we'll see how we can find a way in an instant

Note that this value is relatively close to the one we obtained using the character-state model in subsubsection 3.2.4.

The π-decomposition assuming a normal distribution Now, if we wanted to still use the π-decomposition, we would need to assumed that the temperature is actually normally distributed. One way to do so is to weight each of the environments according the density of a normal distribution. We can use the `dnorm()` function to compute such densities and pass them to `wt_env` argument of `Reacnorm`. This way, we can use the `rn_pi_decomp()` function of `Reacnorm`, since we're assuming a normal distribution:

```
post_plas_nl_tpc_norm <-
    map2(post_nl_tpc[["Theta"]], post_nl_tpc[["G"]],
         \(th_, G_) { rn_pi_decomp(theta    = th_,
                                   G_theta  = G_,
                                   env      = vec_env,
                                   shape    = gg_shape,
                                   fixed    = c(3, 4),
                                   wt_env   = dnorm(vec_env)) },
         .progress = TRUE) |>
    bind_rows() |>
    cbind(post_nl_tpc_info) |>
    as_draws_df()
summarise_draws(post_plas_nl_tpc_norm)
```

```
# 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   0.0927 0.0927 0.00621 0.00619 0.0828 0.103 1.01      761.     755.
2 Pi_Sl    0.329  0.325  0.0265  0.0205  0.292  0.384 0.998     818.     953.
3 Pi_Cv    0.195  0.195  0.0101  0.00949 0.178  0.211 0.999     871.    1032.
```

Since we assumed a different distribution for the environment, the estimated value for V_{Plas} changed. This time, however, we obtained a proper π-decomposition into a slope and curvature components[12]. Notably, the part of the variance arising from the slope is considerable ($\pi_{\text{Sl}} = 0.33$), as is the part arising from curvature to a lesser extent ($\pi_{\text{Cv}} = 0.20$). It should be noted, for the interpretation of those values, that since we assumed a normal distribution, the values of the environment close to 0 (the mean value of the environment) are given more weight than values close to -2 or 2. As such, the slope and curvature near 0 are the ones that are driving the values for π_{Sl} and π_{Cv}.

Computing the additive genetic variance decomposition When studying the additive genetic variance, we do not require the normality assumption about the environment to perform the γ- or ι-decomposition, and thus, we can readily apply the `rn_gen_decomp()` function:

```
post_gen_nl_tpc <-
    map2(post_nl_tpc[["Theta"]], post_nl_tpc[["G"]],
         \(th_, G_) { rn_gen_decomp(theta    = th_,
                                    G_theta  = G_,
                                    env      = vec_env,
                                    shape    = gg_shape,
                                    fixed    = c(3, 4)) },
```

[12] Note that we could have used the same trick for the quadratic curve we used above, but given the not-so-good fit, the results would not have been very trustworthy.

```
        .progress = TRUE) |>
    bind_rows() |>
    select(where(\(col_) { abs(mean(col_)) > 10^-5 })) |>
    cbind(post_nl_tpc_info) |>
    as_draws_df()
summarise_draws(post_gen_nl_tpc)
```

```
# 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            0.0495   0.0491  0.00557 0.00552  0.0411   0.0594    1.01      994.     836.
2 V_A              0.0224   0.0220  0.00342 0.00355  0.0173   0.0284    1.01      936.     800.
3 V_AxE            0.0272   0.0271  0.00250 0.00251  0.0234   0.0314    1.01     1024.     984.
4 Gamma_cmax       0.700    0.703   0.0403  0.0412   0.632    0.762     1.00      864.     892.
5 Gamma_xopt       0.303    0.300   0.0405  0.0415   0.241    0.373     1.00      870.     893.
6 Gamma_cmax_xopt -0.00298 -0.00204 0.00327 0.00249 -0.00965  0.000270 0.999      811.     856.
7 Iota_cmax        0.460    0.461   0.0490  0.0509   0.382    0.540     1.00      873.     943.
8 Iota_xopt        0.548    0.548   0.0494  0.0509   0.469    0.629     1.00      881.     979.
9 Iota_cmax_xopt  -0.00806 -0.00638 0.00692 0.00588 -0.0214  -0.000332 1.00      834.     943.
```

Comparing such values with the character-state model in subsubsection 3.2.4, the values for the total additive genetic variance (V_{Add}) are very close, but the values for V_{A} and $V_{\text{A}\times\text{E}}$ are different, with more balance between these two components in this non-linear model than in the character-state model. Given the credible intervals, this is likely to be just stochastic fluctuation. As we can see, most of the additive genetic variance in the reaction norm seems to come from genetic variation in $C_{\max}$ ($\gamma_{C_{\max}} = 0.7$), while the (short) majority of the additive genetic variation in plasticity rather comes from ε_0 ($\iota_{\varepsilon_0} = 0.55$). Since the model is non-linear this time, there is a distinction to be made between V_{Add} and V_{Gen} which are not equal any more. So, we can compute V_{Gen} separately:

```
post_gen_nl_tpc[["V_Gen"]] <-
    map2_dbl(post_nl_tpc[["Theta"]], post_nl_tpc[["G"]],
             \(th_, G_) { rn_vgen(theta    = th_,
                                  G_theta  = G_,
                                  env      = vec_env,
                                  shape    = gg_shape,
                                  fixed    = c(3, 4)) },
             .progress = TRUE)
summarise_draws(post_gen_nl_tpc)
```

```
# A tibble: 10 × 10
   variable       mean   median      sd     mad       q5      q95  rhat ess_bulk
   <chr>         <dbl>    <dbl>   <dbl>   <dbl>    <dbl>    <dbl> <dbl>    <dbl>
 1 V_Add       0.0495   0.0491  0.00557 0.00552  0.0411   5.94e-2 1.01      994.
 2 V_A         0.0224   0.0220  0.00342 0.00355  0.0173   2.84e-2 1.01      936.
 3 V_AxE       0.0272   0.0271  0.00250 0.00251  0.0234   3.14e-2 1.01     1024.
 4 Gamma_cmax  0.700    0.703   0.0403  0.0412   0.632    7.62e-1 1.00      864.
 5 Gamma_xopt  0.303    0.300   0.0405  0.0415   0.241    3.73e-1 1.00      870.
 6 Gamma_cma… -0.00298 -0.00204 0.00327 0.00249 -0.00965  2.70e-4 0.999     811.
 7 Iota_cmax   0.460    0.461   0.0490  0.0509   0.382    5.40e-1 1.00      873.
 8 Iota_xopt   0.548    0.548   0.0494  0.0509   0.469    6.29e-1 1.00      881.
 9 Iota_cmax… -0.00806 -0.00638 0.00692 0.00588 -0.0214  -3.32e-4 1.00      834.
```

```
10 V_Gen       0.0579   0.0531  0.0417  0.00626  0.0444   6.58e-2 1.01     1017.
# ℹ 1 more variable: ess_tail <dbl>
```

We can see that V_{Gen} is slightly higher than V_{Add}, because the non-linearity in the model is introducing non-additive genetic variance in the trait, even though all the genetic variance in the parameters is additive.

Computing the additive genetic variance decomposition with a normal assumption If we wanted to match the π-decomposition assuming a normal distribution, we can also compute the γ- and ι-decomposition also assuming a normal distribution, using the same `wt_env` argument[13]:

```
post_gen_nl_tpc_norm <-
    map2(post_nl_tpc[["Theta"]], post_nl_tpc[["G"]],
         \(th_, G_) { rn_gen_decomp(theta    = th_,
                                    G_theta  = G_,
                                    env      = vec_env,
                                    shape    = gg_shape,
                                    fixed    = c(3, 4),
                                    wt_env   = dnorm(vec_env),
                                    width    = 8) },
         .progress = TRUE) |>
    bind_rows() |>
    select(where(\(col_) { abs(mean(col_)) > 10^-5 })) |>
    cbind(post_nl_tpc_info) |>
    as_draws_df()

summarise_draws(post_gen_nl_tpc_norm)
```

```
# A tibble: 9 × 10
  variable        mean   median      sd     mad       q5      q95  rhat ess_bulk
  <chr>          <dbl>    <dbl>   <dbl>   <dbl>    <dbl>    <dbl> <dbl>    <dbl>
1 V_Add        0.0547   0.0540  0.00684 0.00670  0.0442   0.0668  1.01      931.
2 V_A          0.0349   0.0345  0.00536 0.00536  0.0269   0.0447  1.01      882.
3 V_AxE        0.0197   0.0197  0.00190 0.00191  0.0169   0.0231  1.01     1058.
4 Gamma_cmax   0.850    0.851   0.0406  0.0415   0.783    0.915   1.00      818.
5 Gamma_xopt   0.217    0.215   0.0354  0.0349   0.164    0.281   1.00      877.
6 Gamma_cmax… -0.0666  -0.0643  0.0330  0.0317  -0.125   -0.0133  0.998     798.
7 Iota_cmax    0.484    0.486   0.0498  0.0508   0.402    0.563   1.00      879.
8 Iota_xopt    0.513    0.512   0.0495  0.0501   0.434    0.595   1.00      883.
9 Iota_cmax_…  0.00271  0.00303 0.00273 0.00213 -0.00209  0.00622 1.00      932.
# ℹ 1 more variable: ess_tail <dbl>
```

Computing the variance-standardised estimates We have to gather all our estimates to compute the total phenotypic variance in the reaction norm, and use it to standardise our estimates. This will be done almost exactly as for aggressiveness and the character-state of thermal performance (see Figure 15), with the main difference being that we need to use V_{Gen} rather than V_{Add}:

```
post_var_nl_tpc <-
    bind_draws(post_nl_tpc, post_v_plas_nl_tpc, post_gen_nl_tpc) |>
```

[13]Notet that we have to set the argument *width* to 8 here, due to a slight numerical instability when it is set to 10. It is not advisable to reduce this argument too much beyond that limit, as this will start to generate underestimation of the variance.

```
    subset_draws(variable = c("V_Plas", "V_Gen", "V_Add", "V_A", "V_AxE", "V_R")) |>
    mutate_variables(V_Tot = V_Plas + V_Gen + V_R)

post_std_nl_tpc <-
    post_var_nl_tpc |>
    transmute(P2            = V_Plas / V_Tot,
              Broad_H2_RN   = V_Gen / V_Tot,
              H2_RN         = V_Add / V_Tot,
              H2            = V_A / V_Tot,
              H2_I          = V_AxE / V_Tot,
              T2            = (V_Plas + V_Gen) / V_Tot) |>
    cbind(post_nl_tpc_info) |>
    as_draws_df()

summarise_draws(post_std_nl_tpc)
mcmc_trace(post_std_nl_tpc)
mcmc_areas(post_std_nl_tpc,
           prob = 0.95,
           area_method = "scaled height")
```

```
# A tibble: 6 × 10
  variable     mean median      sd     mad     q5   q95  rhat ess_bulk ess_tail
  <chr>       <dbl>  <dbl>   <dbl>   <dbl>  <dbl> <dbl> <dbl>    <dbl>    <dbl>
1 P2          0.648  0.654 0.0395  0.0278  0.586  0.694  1.01      948.      785.
2 Broad_H2_RN 0.297  0.289 0.0417  0.0283  0.250  0.363  1.01      976.      905.
3 H2_RN       0.265  0.264 0.0261  0.0250  0.223  0.307  1.01      796.      857.
4 H2          0.120  0.119 0.0167  0.0168  0.0935 0.147  1.01      790.      761.
5 H2_I        0.145  0.146 0.0115  0.00973 0.126  0.163  1.01      968.      982.
6 T2          0.945  0.945 0.00490 0.00457 0.937  0.953  1.00      972.      952.
```

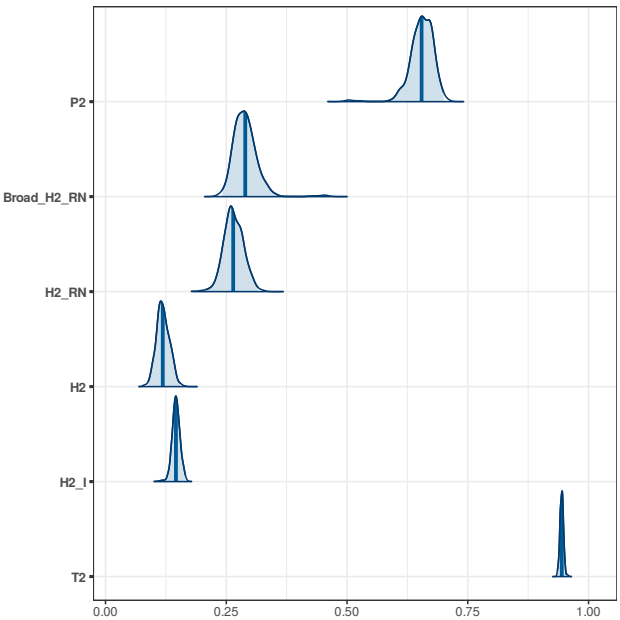

Figure 15: Posterior distribution of the variance-standardised estimates of our variance decomposition of the reaction norm of thermal performance, based on the non-linear model.

So, most of the total variance is coming from the average shape of the reaction norm ($P^2_{\text{RN}} = 0.65$), and less from the total genetic variation ($H^2_{\text{RN}} = 0.30$). Regarding, more specifically, the additive genetic variation ($h^2_{\text{RN}} = 0.27$), it composed of almost the same amount of marginal heritability of the trait ($h^2 = 0.12$) and heritability of plasticity ($h^2_{\text{I}} = 0.15$). As for the character-state, the reaction norm explains almost all of the total phenotypic variation ($T^2_{\text{RN}} = 0.95$).

4 Studying reaction norms in a continuous environment

Coming soon!

References

Vehtari, A et al. (2021). Rank-normalization, folding, and localization: an improved R for assessing convergence of MCMC. *Bayesian Analysis*, 16, 667–718. DOI: 10.1214/20-BA1221 (cit. on p. 5).