Estimating Nonlinear Selection on Behavioral Reaction Norms

Tutorials in Stan

Jordan Scott Martin

3/8/2021

Contents

Introduction

This series of tutorials demonstrates how to effectively code and interpret models of nonlinear selection on behavioral reaction norms (RNs), using the Stan statistical programming language (Carpenter et al. 2017) in R (R Core Team 2020). Stan is an open-source programming language for estimating probabilistic models of arbitrary complexity using fully Bayesian inference with state-of-the-art Markov Chain Monte Carlo (MCMC) sampling techniques (Hoffman and Gelman 2014). Stan interfaces with R through the RStan package (Carpenter et al. 2017), but you will first need to install Stan on your computer and ensure that it is appropriately configured with your C++ toolchain. This can be accomplished by following the instructions for your operating system on the RStan Getting Started page. Once you are able to effectively use RStan, you can begin creating the `.stan` files necessary for estimating models. These files can be composed using RStudio or any text editor. A file can be also be composed directly in R

```
write("// for Stan comments
    functions{...} // Stan models are composed of
    data {...} // multiple programming blocks
    transformed data {...} //only data, parameters, and model
    parameters {...} //blocks are necessary
    transformed parameters {...}
    model {...}
    generated quantities {...} ",
  "mod1.stan")
```

Once an appropriate `.stan` file is prepared, it can be compiled with R for the C++ toolchain using the `stan_model()` function and subsequently estimated with an appropriate list of empirical data using the `sampling()` function. The resulting posteriors of a model can then be accessed with the `extract()` function and manipulated for any further quantities or analyses of interest.

```
#load package
library(rstan)

#compiles the model in C++ for MCMC estimation
mod1 = stan_model("mod1.stan")

#samples posterior distribution of the model with default MCMC settings
results = sampling(object = mod1, data = data)

#extracts posterior estimates
samples =  extract(results)
```

This series is currently under development and will continue to be extended in the coming months to cover a variety of additional modeling scenarios. For now, a full Gaussian model is presented to provide a general introduction to the proposed approach.

Full Gaussian model

Formal model

It's always helpful to write out the formal model we'd like to estimate in Stan before attempting to code it. There are a few reasons for this. Firstly, Stan is a probabilistic programming language and, as such, facilitates coding of formal probabilistic models through direct specification of model parameters and likelihood functions. Therefore, some understanding of the formal structure of any model is necessary to code in Stan. Gaining a deeper understanding of formal statistical models can also be extremely valuable for building researchers' autonomy and ingenuity in data analysis, which opens up the door to developing novel models capturing the most salient features of one's specific empirical system and dataset, rather than pigeonholing things into prepackaged toolkits that may require some undesirable assumptions or simplifications. Researchers unfamiliar with formal statistical models are encouraged to see McElreath (2020) for detailed explanation and examples.

A Gaussian model of selection on a full behavioral reaction norm, i.e. with parameters personality, plasticity, and predictability, can be given by

$$z_{ij} \sim \text{Normal}\left(\mu_{ij}^{(z)}, \sigma_{ij}^{(z)}\right)$$
$$\mu_{ij}^{(z)} = \mu_0^{(z)} + \mu_j^{(z)} + \left(\beta_1^{(z)} + \beta_j^{(z)}\right) x_{ij}$$
$$\log\left(\sigma_{ij}^{(z)}\right) = \theta_0^{(z)} + \theta_j^{(z)}$$
$$\boldsymbol{z}_{\mathbf{p}} = \begin{bmatrix}\boldsymbol{\mu}^{(z)} & \boldsymbol{\beta}^{(z)} & \boldsymbol{\theta}^{(z)}\end{bmatrix}' \sim \text{MVNormal}\left(\mathbf{0}, \mathbf{SRS}\right)$$

$$w_j \sim \text{Normal}\left(\mu_j, \sigma_j\right)$$
$$\mu_j = \mu_0 + \beta_1\left(\mu_j^{(z)}\right) + \beta_2\left(\beta_j^{(z)}\right) + \beta_3\left(\theta_j^{(z)}\right)$$
$$+\beta_4\left(\mu_j^{(z)}\mu_j^{(z)}\right) + \beta_5\left(\beta_j^{(z)}\beta_j^{(z)}\right) + \beta_6\left(\theta_j^{(z)}\theta_j^{(z)}\right)$$
$$+\beta_7\left(\mu_j^{(z)}\beta_j^{(z)}\right) + \beta_8\left(\mu_j^{(z)}\theta_j^{(z)}\right) + \beta_9\left(\beta_j^{(z)}\theta_j^{(z)}\right)$$

$$\mu_0^{(z)}, \beta_1^{(z)}, \theta_0^{(z)}, \mu_0, \beta_1, ..., \beta_9 \sim \text{Normal}(0, 1)$$
$$\mathbf{S}, \sigma \sim \text{Exponential}(1)$$
$$\mathbf{R} \sim \text{LKJ}(2)$$

Notation follows Martin (2021), where this model is explained and justified in greater detail. The individual-specific RN parameter values of behavior $\boldsymbol{z}$ for all individuals are contained in the BLUP vector $\boldsymbol{z}_{\mathbf{p}}$ and the selection effects are described by the regression coefficients $\beta_1, ..., \beta_9$ on fitness measure $\boldsymbol{w}$. For this tutorial, we use general-purpose, weakly regularizing priors on model parameters to promote more robust inference and enhance model identification (Lemoine 2019).

Simulate dataset

With the formal model in place, we can now simulate appropriate data to use for its estimation. We assume a sample of 500 individuals with 6 repeated behavioral measures across the lifespan and a single fitness measure. Parameter values are arbitrarily fixed so that the population-level intercepts and slopes are 0, with 0.3 for all regression coefficients in the fitness model and correlations among random effects, as well as residual variances of 0.5 for the behavior and fitness response models.

```
#simulation parameters
I = 500 #number of individuals
repm = 6 #repeated behavioral measures

#fixed effects
beta = 0.3 #regression coefficients
popint = 0 #population behavior intercept
popslope = 0 #population behavior slope

#random effects
sd = sqrt(0.3) #RN parameter standard deviations
cor = 0.3 #correlations between RN parameters
popdisp = sqrt(0.5) #residual SD of behavior
res = sqrt(0.5) #residual SD of fitness
```

As discussed in Martin (2021), we simulate the variance-covariance matrix $\mathbf{P}$ of RN parameters through matrix multiplication $\mathbf{SRS}$ of a matrix $\mathbf{S}$ with standard deviations on the diagonal and a correlation matrix $\mathbf{R}$.

```
#generate RN covariance matrix P
  R = matrix(cor, nrow=3, ncol=3 )
  R[lower.tri(R)] = t(R)[lower.tri(R)] #force symmetric
  diag(R) = 1 #make correlation matrix
  S = matrix( c(sd,0,0,0,sd,0,0,0,sd), nrow=3, ncol=3 ) #SD matrix
  P = S %*% R %*% S #covariance matrix

#simulate RN parameters for individuals
  library(mvtnorm)
  z_p = rmvnorm(I, mean = rep(0,3), sigma = P)

#separate each parameter
  personality = z_p[,1]
  plasticity = z_p[,2]
  predictability = z_p[,3]
```

We then simulate a random environmental gradient across individuals, which we assume for simplicity is identically and independently distributed acros all observations

```
  #environmental covariate (z-score)
  x = rnorm(I*repm, 0, 1)
```

along with an index used to link each observation to the corresponding individual being observed.

```
  #index of repeated individual measures
  ind = rep(1:I, each = repm)
```

The mean and standard deviations of behavior can then be used to simulate individuals' raw data.

```
  #behavioral response model
  z_mu = popint + personality[ind] + (popslope + plasticity[ind])*x #mean of normal dist
  z_sigma = log(popdisp) + predictability[ind] #SD of normal dist
  z = rnorm(I*repm, mean = z_mu, sd = exp(z_sigma) ) #observations
```

The fitness model is simulated so that each individual has a single measure.

```
  #regression coefficients
  betas = rep(beta, 9) #naive assumption of equivalent coefficients

  #fitness response model
  w_mu = 1 + betas[1]*personality + betas[2]*plasticity + betas[3]*predictability +
             betas[4]*(personality^2) + betas[5]*(plasticity^2) + betas[6]*(predictability^2) +
             betas[7]*(personality*plasticity) + betas[8]*(personality*predictability) +
             betas[9]*(plasticity*predictability)
  w = rnorm(I, mean = w_mu, sd = res) #observations
```

Stan expects a list rather than dataframe of observed values for model estimation. This provides desirable flexibility because it allows for the specification of complex multi-response models with vectors of differing size, as the dimensionality of each variable in this list is declared separately in Stan.

```
data = list(x = x, z = z, w = w, ind = ind, I = I, N = I*repm)
lapply(data,head) #see initial entries of each list item
```

```
## $x
## [1] -0.8123490 -0.8549919  0.3167677  1.9382171 -0.9975228 -1.8273083
##
## $z
## [1]  0.5456486  1.3171713 -0.5391375  2.3724060  1.6555719 -1.2748413
```

```
##
## $w
## [1] 4.1978781 2.3191116 2.4431962 1.2072576 0.8970198 1.4684744
##
## $ind
## [1] 1 1 1 1 1 1
##
## $I
## [1] 500
##
## $N
## [1] 3000
```

Code model

Stan uses its own language for writing probabilistic models, including a variety of built-in functions designed to aid in efficient computation. The biggest conceptual hurdle for new users of Stan is likely to be the absence of an intuitive R-like syntax for specifying model formulas, such as formulas like `y ~ x + (1|z)` that can be used to quickly specify complex generalized linear mixed-effects models. These formulas facilitate highly efficient statistical modeling, but do so at the cost of limiting users' ability to specify atypical model structures. Instead, Stan provides the benefit of nearly unlimited flexibility in model specification, with the added cost of a steeper learning curve. In particular, as noted above, models must be formally specified with mathematically appropriate likelihood functions, rather than this process being handled on the back-end through textual inputs from the user such as `family= poisson(link = "log")`. This may at first seem like a cumbersome task, but it affords a degree of flexibility and autonomy necessary for easily estimating the proposed models in Stan, which to the best of my knowledge cannot be accomplished with other mainstream statistical software. Nonetheless, it is important to recognize that some practice and trial-and-error will also be required to gain competency and comfortability with Stan. I therefore encourage researchers to review the Stan Reference Manual, as well the extensive collection of Stan Case Studies, which will provide a more robust foundation for estimating any model of interest in Stan.

As mentioned above, a basic Stan model consists of multiple programming blocks that together specify the data, parameters, likelihood, and quantities of interest for a model. Rather than tackling the model in a single step, we consider the blocks in turn before putting them together in a single file.

Data

The first component of a Stan model is the data block, where we'll tell the model what to expect from our data list, as well as how to treat that data inside the model.

```
data {
  int<lower=1> I; //total individuals
  int<lower=1> N; //total number of observations
  int<lower=1> ind[N]; //index of individual observations
  vector[N] x; //environmental covariate
  vector[N] z; //behavioral measurements
  vector[I] w; //fitness measurements
}
```

We first tell the model to expect integers with values greater than 1 for the total number of individuals observed `I` and the total number of observations for the repeatedly measured behavioral measure `N`. We know we only have a single fitness measure per individual, so `I` also tells us the total number of fitness observations. The next step is to specify an index for connecting repeated observations of the behavior `z` to the identity of the individual being observed. This index should be represented with integers specified according to the order of the data vectors `z` and `w`. The argument `ind[N]` tells Stan that these integer values should in total be of length N. If one has indexed observations in their data using character strings, they will need to first be converted to integers. For the simulated dataset, this index looks like

```
head(cbind(z,ind),15)
```

```
##                z ind
##  [1,]  0.5456486   1
##  [2,]  1.3171713   1
##  [3,] -0.5391375   1
##  [4,]  2.3724060   1
##  [5,]  1.6555719   1
##  [6,] -1.2748413   1
##  [7,] -0.6485254   2
##  [8,] -1.0972027   2
##  [9,]  0.1376370   2
## [10,] -0.6416482   2
## [11,] -0.1813423   2
## [12,]  0.3179547   2
## [13,] -1.2096485   3
## [14,] -0.8298268   3
## [15,] -0.2577220   3
```

The remaining arguments tell Stan to expect vectors of appropriate length for the environmental covariate x used to estimate plasticity, the behavioral measure z, and the fitness measure w.

Parameters

The parameters block will take all of the basic parameters that are specified in the model. We begin by considering the fixed effects in the formal model, although the order of specification in the parameters block is entirely arbitrary.

```
parameters {
  //fixed population effects
  real mu_0z; //z population intercept
  real beta_1z; //z population slope
  real theta_0z; //z population dispersion
  real mu_0; //w population intercept
  vector[9] betas; //fitness regression coefficients
//...
```

mu_0z is the population intercept x of the linear predictor of behavior z, beta_1z is the population slope x, and theta_0z is the population intercept of the dispersion parameter x. For the fitness model, we specify mu_0 for the global intercept x, as well as a vector betas containing 9 regression coefficients for each of the selection effects $\beta_1, ..., \beta_9$ in the fitness model. Note that this could be equivalently specified by giving each element of this vector separately, e.g.

```
  real beta_1;
  real beta_2;
  real beta_3;
  real beta_4;
//...
```

For the random effects, a slightly more complicated setup is used.

```
//...
  //random effects
  real<lower=0> sigma_0; //w dispersion
  vector<lower=0>[3] sd_zp; //RN parameter sds
  matrix[I,3] std_dev; //individual-level RN deviations
  cholesky_factor_corr[3] R_chol; //RN parameter correlations
}
```

First, we specify `sigma_0` for the residual standard deviation (SD) of the linear fitness model σ_0, along with a vector `sd_zp` of length 3 for each of the SDs of the RN parameters (personality, plasticity, and predictability). The matrix $\mathbf{S}$ in the formal model has `sd_zp` on its diagonal. Importantly, because SD parameters by definition cannot take on values below zero, we need to specify `<lower=0>` so that the parameters do not take on values lower than 0 during model estimation. A matrix of dimensions (I x 3) is also specified for the standardized deviations of each individual's RN parameter values from the population values. As explained below, these standard normal deviates are scaled by the SDs and correlations among RN parameters to derive BLUPs of appropriate magnitude.

Finally, a matrix parameter `R_chol` is specified for the RN parameter correlation matrix $\mathbf{R}$. However, rather than using the function `corr_matrix` for a full correlation matrix, we instead use a special function `Cholesky_factor_corr` to estimate a so-called *Cholesky decomposition* of $\mathbf{R}$. To understand why we do this, note that for any positive definite matrix $\mathbf{R}$, a Cholesky decomposition can be defined such that

$$\mathbf{R} = \mathbf{R}_{\mathrm{L}}\mathbf{R}_{\mathrm{L}}^{\mathrm{T}}$$

where $\mathbf{R}_{\mathrm{L}}$ is a lower-triangular matrix and $^{\mathrm{T}}$ indicates matrix transposition. This property means that we can always estimate the model using a smaller lower-triangular matrix $\mathbf{R}_{\mathrm{L}}$ and subsequently recover the full positive-definitive matrix $\mathbf{R}$ by post-multiplying $\mathbf{R}_{\mathrm{L}}$ with its transpose. This trick is useful for making any Stan model sample more efficiently, as computations can be done more quickly with the reduced matrix of lower dimensionality that lacks the redundant features of the full symmetric correlation matrix.

Transformed parameters

With these basic parameters in place, we can also further specify parameters in the transformed parameters block that are simply combinations of the basic parameters. In this model, we specifically need derive RN parameters (BLUPs) that are appropriately scaled by the RN covariance matrix $\mathbf{P}$ in the formal model. This is accomplished as follows

```
transformed parameters {
  matrix[I,3] zp; //individual phenotypic RN parameter values
  zp =  std_dev * diag_pre_multiply(sd_zp, R_chol)' ;
}
```

This specification gives the appropriate BLUPs for each individual, as described in the formal model by

$$\boldsymbol{z}_{\mathbf{p}} = \begin{bmatrix}\boldsymbol{\mu}^{(z)} & \boldsymbol{\beta}^{(z)} & \boldsymbol{\theta}^{(z)}\end{bmatrix}' \sim \mathrm{MVNormal}\left(\mathbf{0}, \mathbf{SRS}\right)$$

To see how this works, note that any normally distributed random variable $\boldsymbol{z}$

$$\boldsymbol{z} \sim \mathrm{Normal}(0, \sigma_z)$$

can also be expressed as a standard normal variable z_{std} scaled by the original SD

$$\boldsymbol{z} \equiv \boldsymbol{z}_{\mathbf{std}}\sigma_z$$

$$\boldsymbol{z}_{\mathbf{std}} \sim \mathrm{Normal}(0, 1)$$

Similarly for an I x p matrix $\boldsymbol{Z}$ of p phenotypes where

$$\boldsymbol{Z} \sim \mathrm{MVNormal}(\mathbf{0}, \mathbf{P})$$

we can derive the appropriately scaled values with a matrix of standard normals $\boldsymbol{Z}_{\mathbf{std}}$ and a Cholesky decomposition of $\mathbf{P}$, so that

$$\boldsymbol{Z} \equiv \boldsymbol{Z}_{\mathbf{std}}\mathbf{P}_{\mathrm{L}}^{\mathrm{T}}$$

$$\mathbf{P}_{\mathrm{L}}^{\mathrm{T}} = \mathrm{Chol}(\mathbf{P})^{\mathrm{T}} = \mathrm{Chol}(\mathbf{SRS})^{\mathrm{T}} = (\mathbf{SR}_{\mathrm{L}})^{\mathrm{T}}$$

In this case, $\boldsymbol{Z}_{\mathbf{std}}$ corresponds to `std_dev` and the function `diag_pre_multiply()` first creates a matrix with `sd_zp` on the diagonal, i.e. $\mathbf{S}$, and then multiplies it with the lower Cholesky matrix `R_chol` representing $\mathbf{R}_{\mathrm{L}}$. The `'` symbol applies the transpose operator $^{\mathrm{T}}$. Although this so-called *non-centered parameterization* may seem like a lot of unnecessary work, separating out the scale and associations of the random effects in this way will often lead to better model convergence and thus more efficient model estimation. Therefore, these mathematically equivalent reparameterizations of the formal model are generally worth implementing although not strictly necessary.

Model

The model block contains the likelihood functions of the model, the priors for the basic parameters, as well as any data structures that one may want to create for pragmatic convenience in specifying the model but not save in the output (e.g. to reduce memory usage). We can again work through each section these sections in turn.

```
model{
  //separate RN parameters
  vector[I] zp_mu = col(zp,1); //personality
  vector[I] zp_beta = col(zp,2); //plasticity
  vector[I] zp_theta = col(zp,3); //predictability

  //initialize vectors for response models
  vector[N] z_mu; //linear predictor of behavior expectation
  vector[N] z_sigma; //linear predictor of behavior dispersion
  vector[I] w_eta; //linear predictor of fitness expectation
//...
```

In this first step, we specify a few new vectors to separate out each RN parameter from the matrix `zp` created in the transformed parameters block. This is not strictly necessary, but avoids clutter in the model likelihood caused by repeatedly subsetting the matrix for the respective columns `col(zp,1)`, `col(zp,2)`, and `col(zp,3)`. Similarly, to tidy up the model likelihood, we create new vectors to hold the linear predictors of each behavioral and fitness observation. Note that there is no need to create a linear predictor for the dispersion of fitness, as nothing is predicting the residual SD of the fitness model, which is already taken care of by the `sigma_0` parameter.

The next step is then to fill in these vectors. For the response model of behavior `z`

```
//...
  //behavioral RN response model
  z_mu = mu_0z + zp_mu[ind] + (beta_1z + zp_beta[ind]).*x ;
  z_sigma = exp(theta_0z + zp_theta[ind]) ;
  z ~ normal(z_mu, z_sigma);
```

The final line tells Stan that the observed values `z` were generated by a Normal distribution with a likelihood function described by the expected means `z_mu` and standard deviations `z_sigma` of each observation. Note that `z_sigma` is calculated with the exponential function `exp()` because the formal model is specified with a log link function, so that the inverse exponential link function is applied to the linear predictor in order return estimates on the appropriate scale, i.e. if $\log(\sigma) = \theta$ then $\exp(\theta) = \sigma$. The operator `.*` indicates element-wise multiplication of vectors, which in this case multiplies the slopes `beta_1z + zp_beta[ind]` by the observed environmental gradient x. These three lines of code are therefore equivalent to

$$z_{ij} \sim \text{Normal}\left(\mu_{ij}^{(z)}, \sigma_{ij}^{(z)}\right)$$
$$\mu_{ij}^{(z)} = \mu_0^{(z)} + \mu_j^{(z)} + \left(\beta_1^{(z)} + \beta_j^{(z)}\right) x_{ij}$$
$$\log\left(\sigma_{ij}^{(z)}\right) = \theta_0^{(z)} + \theta_j^{(z)}$$

The index `ind` is here used to appropriately repeat the random effect values of each RN parameter across repeated observations of the behavior. For example, if the first four observations are for individual 1, so that `ind`={1,1,1,1,2,...}, then `zp_mu[ind]` will repeat the first value of `zp_mu` for the first four observations. This is why it is essential to correctly match the order of the index and the response vectors.

The fitness model can also be specified accordingly

```
//...
  //fitness response model
  w_eta = mu_0 + betas[1]*zp_mu + betas[2]*zp_beta + betas[3]*zp_theta +
                 betas[4]*(zp_mu .*zp_mu) + betas[5]*(zp_beta .*zp_beta) +
                 betas[6]*(zp_theta .*zp_theta) +
                 betas[7]*(zp_mu .*zp_beta) + betas[8]*(zp_mu .*zp_theta) +
                 betas[9]*(zp_beta .*zp_theta) ;
  w ~ normal(w_eta, sigma_0);
```

There is no need for the `ind` index here because each individual's fitness is only observed once. The final necessary step is to introduce priors for all basic parameters listed in the parameters block.

```
//...
  //model priors

  //fixed effects
  mu_0z ~ normal(0,1);
  beta_1z ~ normal(0,1);
  theta_0z ~ normal(0,1);
  mu_0 ~ normal(0,1);
  betas ~ normal(0,1);

  //random effects
  sd_zp ~ exponential(1);
  R_chol ~ lkj_corr_cholesky(2);
  to_vector(std_dev) ~ std_normal();
  sigma_0 ~ exponential(1);
}
```

For the matrix of standard normal RN parameter deviations `std_dev`, we should always specify that the vector of all elements in this matrix are described by a `std_normal()` distribution, which is necessary for the non-centered parameterization introduced in the transformed parameters block. The other parameters can be given whatever priors are intended for the analysis, which in this case are the weakly regularizing priors used in the formal model, i.e.

$$\mu_0^{(z)}, \beta_1^{(z)}, \theta_0^{(z)}, \mu_0, \beta_1, ..., \beta_9 \sim \text{Normal}(0,1)$$
$$\mathbf{S}, \sigma \sim \text{Exponential}(1)$$
$$\mathbf{R} \sim \text{LKJ}(2)$$

Generated quantities

The final programming block in our Stan model concerns the calculation of any quantities of interest which weren't directly estimated in earlier blocks.

```
generated quantities{
matrix[3,3] R = R_chol * R_chol'; //RN correlation matrix
matrix[3,3] S = diag_matrix(sd_zp); //RN correlation matrix
matrix[3,3] P = S*R*S; //RN covariance
vector<lower=0>[3] V_P = sd_zp .* sd_zp; //RN variances
}
```

We derive the full correlation matrix $\mathbf{R}$ by multiplying the Cholesky matrix $\mathbf{R}_{\text{L}}$ used for model estimation with its transpose, accomplished with the transpose operator `'`. The covariance matrix $\mathbf{P}$ is derived by multiplying the full correlation and standard deviation matrices $\mathbf{SRS}$, and the variances of the RN parameters are derived by squaring the SDs in `sd_zp`.

Final model code

With each programming block coded, we can put them all together and write to a single `.stan` file in R

```
write("
data {
  int<lower=1> I; //total individuals
  int<lower=1> N; //total number of observations
  int<lower=1> ind[N]; //index of individual observations
  vector[N] x; //environmental covariate
  vector[N] z; //behavioral measurements
  vector[I] w; //fitness measurements
}
parameters {
  //fixed population effects
  real mu_0z; //z population intercept
  real beta_1z; //z population slope
  real theta_0z; //z population dispersion
  real mu_0; //w population intercept
  vector[9] betas; //fitness regression coefficients

  //random effects
  real<lower=0> sigma_0; //w dispersion (sigma for Gaussian)
  vector<lower=0>[3] sd_zp; //RN parameter sds
  matrix[I,3] std_dev; //individual-level RN deviations
  cholesky_factor_corr[3] R_chol; //RN parameter correlations
}
transformed parameters {
  matrix[I,3] zp; //individual phenotypic RN parameter values
  zp =  std_dev * diag_pre_multiply(sd_zp, R_chol)' ;
}
model{
  //separate RN parameters
  vector[I] zp_mu = col(zp,1); //personality
  vector[I] zp_beta = col(zp,2); //plasticity
  vector[I] zp_theta = col(zp,3); //predictability

  //initialize vectors for response models
  vector[N] z_mu; //linear predictor of behavior expectation
  vector[N] z_sigma; //linear predictor of behavior dispersion
  vector[I] w_eta; //linear predictor of fitness expectation

  //behavioral RN response model
  z_mu = mu_0z + zp_mu[ind] + (beta_1z + zp_beta[ind]).*x ;
  z_sigma = exp(theta_0z + zp_theta[ind]) ;
  z ~ normal(z_mu, z_sigma);

  //fitness response model
  w_eta = mu_0 + betas[1]*zp_mu + betas[2]*zp_beta + betas[3]*zp_theta +
               betas[4]*(zp_mu .*zp_mu) + betas[5]*(zp_beta .*zp_beta) +
               betas[6]*(zp_theta .*zp_theta) +
               betas[7]*(zp_mu .*zp_beta) + betas[8]*(zp_mu .*zp_theta) +
               betas[9]*(zp_beta .*zp_theta) ;
  w ~ normal(w_eta, sigma_0);

  //model priors

  //fixed effects
  mu_0z ~ normal(0,1);
  beta_1z ~ normal(0,1);
  theta_0z ~ normal(0,1);
  mu_0 ~ normal(0,1);
  betas ~ normal(0,10);

  //random effects
  sd_zp ~ exponential(1);
  R_chol ~ lkj_corr_cholesky(2);
  to_vector(std_dev) ~ std_normal();
  sigma_0 ~ exponential(1);
}

generated quantities{
matrix[3,3] R = R_chol * R_chol'; //RN correlation matrix
matrix[3,3] S = diag_matrix(sd_zp); //RN correlation matrix
matrix[3,3] P = S*R*S; //RN covariance
vector<lower=0>[3] V_P = sd_zp .* sd_zp; //RN variances
}",
"mod1.stan")
```

Estimate model

To estimate this model, we first pass it to Stan for C++ compilation.

```
#load package
library(rstan)

#compiles the model in C++ for MCMC estimation
mod1 = stan_model("mod1.stan")

#basic settings for rstan
options(mc.cores = parallel::detectCores())
rstan_options(auto_write = TRUE)
```

The compiled model in `mod1` is now ready to be sampled immediately using Markov Chain Monte Carlo (MCMC), which is accomplished by passing it to the `sampling()` function. For default default MCMC settings in Stan, we could run

```
#sampling posterior dist of the model with default MCMC settings
results = sampling(object = mod1, data = data)
```

However, given that our model is somewhat complex, it is helpful to use custom settings for the sampler that will reduce the risk of poor sampling performance. In particular, we can manually specify that the MCMC sampler should use 1000 iterations per chain to converge on the target joint posterior distribution `warmup=1500`, with the subsequent 1500 iterations/chain used as posterior samples `iter = 3000` (i.e. `iter - warmup` = number of MCMC samples per chain). `init = 0` initializes the samplers near null values, which is not necessary but can aid sampling of complex models. Four MCMC chains are used to assess model convergence across independent random samplers `chains=4`, with one core assigned to each chain for parallel processing `cores=4`. The appropriate number of cores to use will be contingent on one's hardware. The `adapt_delta=0.95` argument reduces the risk of divergent transitions during sampling.

```
#progress of MCMC chains can be tracked in the viewer pane of RStudio
results = sampling(object = mod1, data = data, warmup=1500, iter = 3000, init = 0,
                    chains=4, cores=4, control=list(adapt_delta=0.95) )
#save model
saveRDS(results, "results_mod1.RDS")
```

Some readers may note that there is no argument specified for thinning the chain, which implicitly specifies the default argument `thin=1`. Although there are specific contexts where thinning is useful for MCMC sampling, it is generally unnecessary and computationally inefficient (Link and Eaton 2012).

If you estimate a model in Stan and receive a warning or error, it may indicate issues with the MCMC sampler, which should always be taken seriously. Further description of these and other warnings can be found in the Stan Warning Guide. Some warnings can be safely ignored in particular contexts, but efforts should always be taken to first remove the issue before interpreting or reporting results from the sampler. If you receive a warning regarding divergence transitions, a straightforward first step is to increase the `adapt_delta` value closer to 1, e.g. from 0.95 to 0.99. The higher this value, the slower the model will sample but the less likely that divergent iterations will occur. Similarly, if warnings of bulk or tail ESS are received, a first step is to simply let the chains sample for longer by increasing the `iter`, e.g. from 2000 to 2500 or 3000.

Assuming that the sampler worked as intended, we can then extract the posterior MCMC samples from the model.

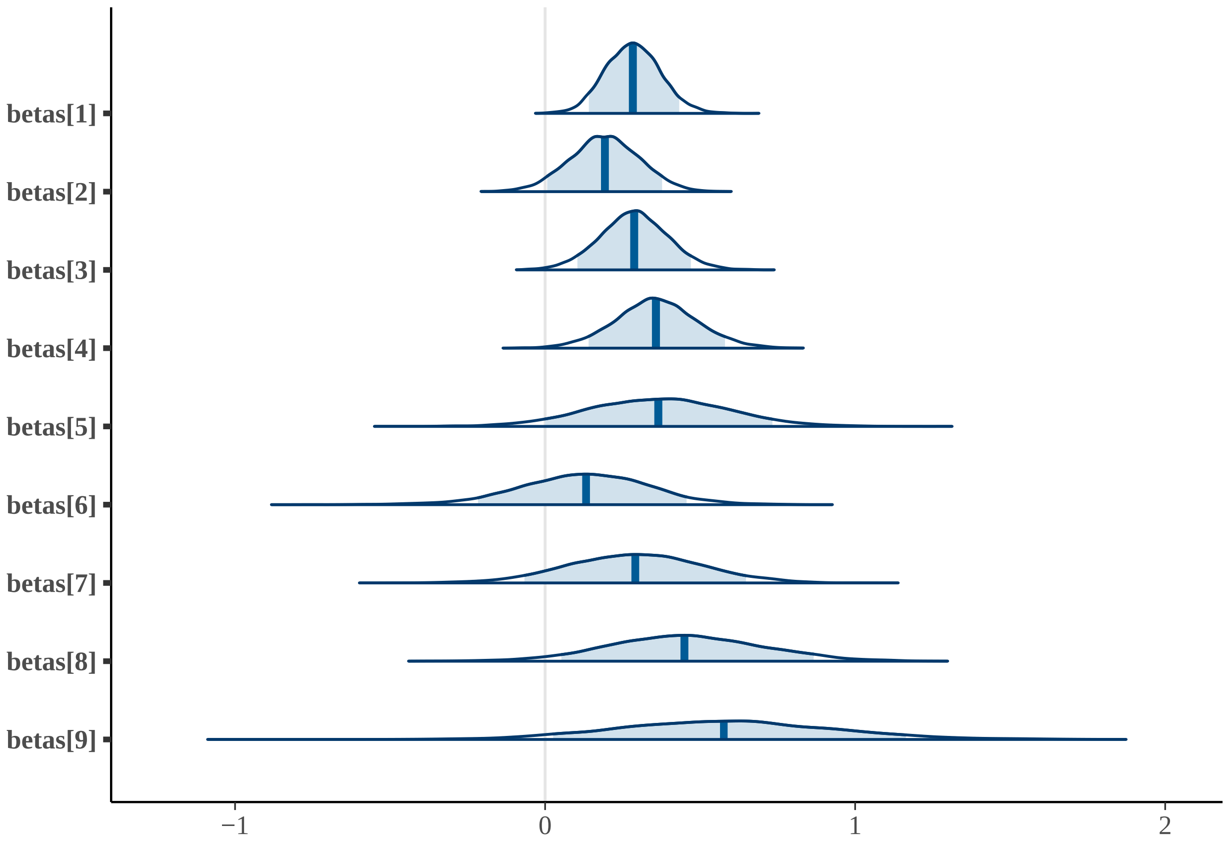
```
#extracts posterior estimates
samples =  extract(results)

#MCMC samples for linear selection coefficients
head(samples$betas[,1:3])
```

```
##
## iterations        [,1]       [,2]      [,3]
##          [1,] 0.4286686 0.13072399 0.2313102
##          [2,] 0.4047872 0.12240215 0.1741672
##          [3,] 0.1281531 0.26382081 0.1530637
##          [4,] 0.5762619 0.18941427 0.2699328
##          [5,] 0.3410815 0.05963205 0.2641004
##          [6,] 0.1497065 0.22204168 0.3474415
```

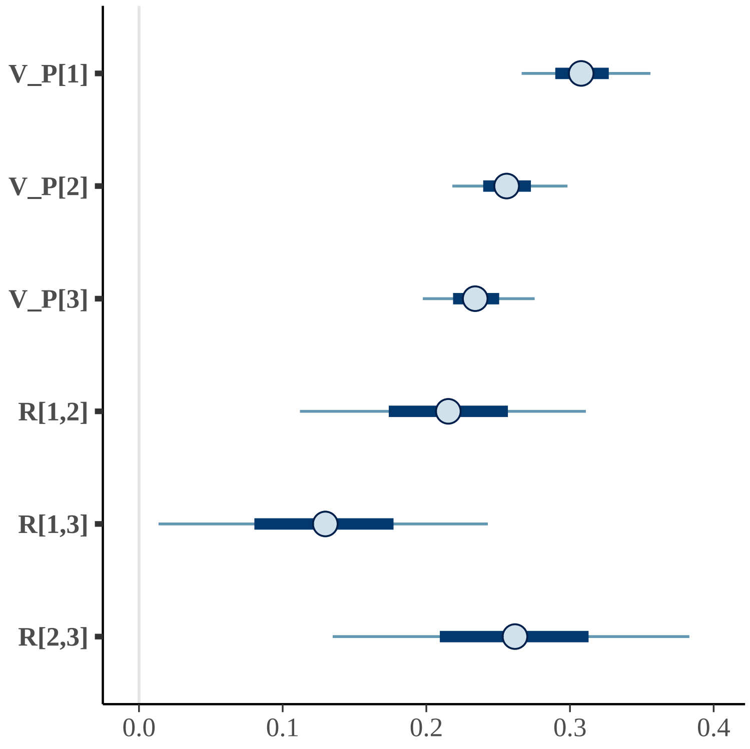
Investigate results

Before hypothesis testing, it is useful to visualize the shapes and locations of the posterior distributions of model parameters. There are a variety of ways this can be accomplished. For example, the `bayesplot` package can be used to generate a variety of useful plots.

```r
library(bayesplot)

#selection coefficients, expected value and 90% CIs
mcmc_areas(results, pars = c( paste0("betas[",seq(1:9),"]") ), prob = 0.9 )
```



```r
#variance & corrs of RN parameters, mean and 50% CIs (dark line) and 90% CIs
mcmc_intervals(results, pars = c( paste0("V_P[",seq(1:3),"]"),"R[1,2]","R[1,3]","R[2,3]" ) )
```



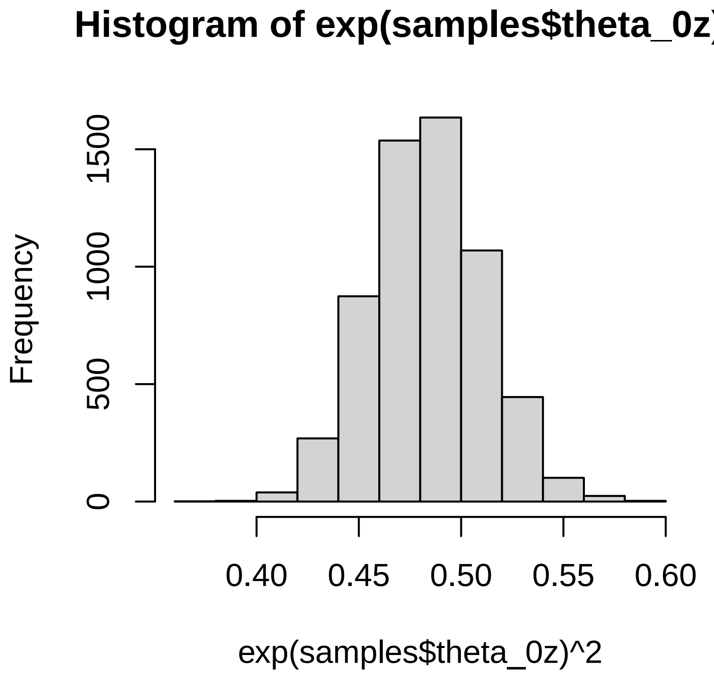
Point estimates summarizing these posteriors can be quickly generated by summarizing the model.

```r
#only first 17 parameters, round to ease interpretation
round(summary(results)$summary[1:17,],2)
```

```
##              mean se_mean   sd  2.5%   25%   50%   75% 97.5%   n_eff Rhat
## mu_0z        0.00    0.00 0.03 -0.06 -0.02  0.00  0.02  0.06 2495.95    1
## beta_1z     -0.05    0.00 0.03 -0.11 -0.07 -0.05 -0.03  0.01 2962.68    1
## theta_0z    -0.36    0.00 0.03 -0.42 -0.38 -0.36 -0.34 -0.31 3038.88    1
## mu_0         0.92    0.00 0.08  0.76  0.87  0.92  0.97  1.08 2577.48    1
## betas[1]     0.28    0.00 0.09  0.12  0.22  0.28  0.34  0.46 3876.57    1
## betas[2]     0.19    0.00 0.11 -0.03  0.12  0.19  0.27  0.41 3035.44    1
## betas[3]     0.29    0.00 0.11  0.07  0.21  0.29  0.36  0.51 4394.43    1
## betas[4]     0.36    0.00 0.13  0.10  0.27  0.36  0.44  0.62 4453.06    1
## betas[5]     0.37    0.01 0.22 -0.07  0.21  0.37  0.52  0.81 1960.91    1
## betas[6]     0.13    0.00 0.21 -0.30 -0.01  0.13  0.27  0.53 2362.55    1
## betas[7]     0.29    0.00 0.22 -0.14  0.14  0.29  0.44  0.71 2653.50    1
## betas[8]     0.45    0.00 0.25 -0.02  0.28  0.45  0.62  0.93 2638.06    1
## betas[9]     0.58    0.01 0.35 -0.07  0.35  0.58  0.81  1.29 1515.57    1
## sigma_0      0.73    0.00 0.03  0.68  0.71  0.73  0.75  0.79 5177.85    1
## sd_zp[1]     0.56    0.00 0.02  0.51  0.54  0.55  0.57  0.60 2375.14    1
## sd_zp[2]     0.51    0.00 0.02  0.46  0.49  0.51  0.52  0.55 2368.15    1
## sd_zp[3]     0.48    0.00 0.02  0.44  0.47  0.48  0.50  0.53 2296.65    1
```

The extracted posterior samples can also be manually plotted and summarized using base R functions. For example, we can look at the population average residual variance of behavior $\theta_0^{(z)}$ on the original data scale by manually applying the inverse link function (`exp()`) to the log-scale SD `theta_0z` and subsequently squaring to return the variance.

```r
#discrete approximation of posterior dist
hist(exp(samples$theta_0z)^2)
```



The `shinystan` package also provides a very helpful graphical user interface for looking at all aspects of model fit and estimation. Running this code will open a new window in your internet browser for looking at the model in greater detail.

```r
library(shinystan)
launch_shinystan(results)
```

Hypothesis testing

MCMC not only facilitates sampling complex Bayesian models but also conducting straightforward and direct forms of hypothesis testing. For example, if want to know how much support there is for positive linear and nonlinear selection effects, we simply need to calculate the proportion of the MCMC samples for these parameters with positive magnitude, which approximates the area under the posterior distribution providing support for positive effects.

```r
#for each column, calculate probability of positive effect
apply(samples$betas, 2, FUN = function(x) sum(x>0)/length(x) )
```

```
## [1] 0.9996667 0.9553333 0.9948333 0.9963333 0.9493333 0.7398333 0.9096667
## [8] 0.9690000 0.9595000
```

Overall, the model provides consistent support for positive linear and nonlinear selection across the RN parameters, with most effects showing posterior probabilities ≥ 0.90. However, the evidence for the quadratic effect of predictability `beta[6]`, or $\beta_6 \left(\theta_j^{(z)}\theta_j^{(z)}\right)$, is much weaker, with a posterior probability of only 0.74 in support of a positive effect, suggesting that there is a 0.26 probability or ~1/4 chance of a negative effect being observed. Another way to think about these probabilities is in relation to Bayesian credible intervals (CIs).

In particular, we expect that if there is at least 0.95 probability of a directional effect, the 90% Bayesian CI will exclude zero.

```
#for each column, calculate probability of positive effect
apply(samples$betas, 2, FUN = function(x) quantile(x, c(0.05, 0.95)) ) #90% CI
```

```
##
##           [,1]        [,2]      [,3]      [,4]          [,5]       [,6]
##   5%  0.1411164 0.006752543 0.1041608 0.1403970 -0.0006781906 -0.2164636
##   95% 0.4325692 0.377335106 0.4701338 0.5801455  0.7331204912  0.4563855
##
##             [,7]       [,8]       [,9]
##   5%  -0.06665065 0.05239159 0.02603818
##   95%  0.64781650 0.86620450 1.16348908
```

We can see, for example, that the lower bound of `beta[5]`, corresponding to $\beta_5\left(\beta_j^{(z)}\beta_j^{(z)}\right)$, is just at the negative boundary of zero, consistent with the posterior probability of 0.949. It is important to emphasize that although 0.95 is a useful heuristic for designating clear evidence of an effect, discretizing this information into "significant" or "non-significant" is generally a waste of information. Put another way, these Bayesian hypothesis tests provides a continuous measure of evidence that should also be interpreted continuously. Much as the difference between significance and non-significance is itself generally not statistically significant (see McShane et al. 2019 for discussion), so too is the difference between e.g. a posterior probability of 0.93 versus 0.97 indicate a transition across a biologically or mathematically meaningful threshold. Thus, much like a null hypothesis test, one should eschew the notion that a posterior probability <0.95 indicates "no evidence of an effect," and instead get comfortable describing varying degrees of support (weak, moderate, and strong) for or against hypothesized effects. Any probability greater than 0.50 provides some support for an effect, but most researchers would be uncomfortable to confidently assert empirical claims without much greater empirical support in their favor, e.g. only a 1/20 chance of an effect in the opposite direction (i.e. a posterior probability of 0.95). Therefore, the posterior probability of 0.74 for a positive `beta[6]` indicates that our data provides *some* evidence for a positive quadratic selection effect on predictability, but this evidence is nonetheless very weak/highly uncertain and warrants cautious description and interpretation. In the opinion of the author, promoting this Bayesian attitude toward evidence within behavioral ecology will be an important tool for promoting the goals of open science, as it may help to dampen file-drawer effects and reduce the risk of p-hacking. A continuous approach to statistical inference also encourages researchers to put greater emphasis on effect sizes, credible intervals, and additional metrics which can collectively increase or decrease the overall "significance" of an empirical finding (McShane et al. 2019).

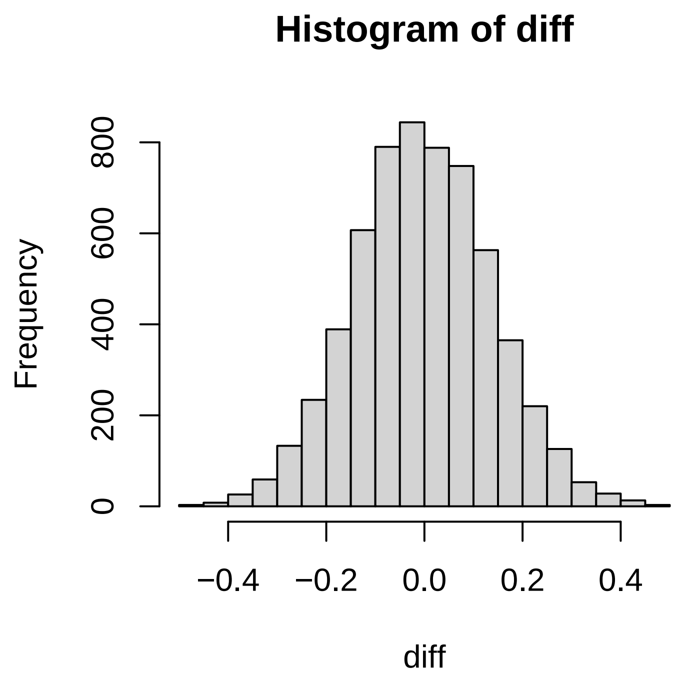
As discussed in Martin (2021), A variety of other hypotheses can be easily tested for any parameter in the model. For instance, we could ask whether there is support for directional selection on personality being greater than directional selection on plasticity among individuals.

```
sum(samples$betas[,1] > samples$betas[,3])/length(samples$betas[,1])
```

```
## [1] 0.4845
```

This value lower than 0.50 indicates that there is 0.52 probability in favor of the plasticity selection effect being greater than the personality effect. As we expect based on simulating equivalent regression coefficients, this indicates little to no evidence in support of a difference between these selection effects. We can also use further pieces of information about the difference of these effect sizes to inform our inferences

```
diff = samples$betas[,1] - samples$betas[,3] #posterior of the difference in coefs
hist(diff) #visualize the posterior
```



```
mean(diff) #expected difference between regression coefs
```

```
## [1] -0.003190825
```

```
mad(diff) #median absolute deviation (robust SD) of expected difference
```

```
## [1] 0.1390162
```

```
quantile(diff, c(0.05,0.95)) #90% CI of expected difference
```

```
##         5%        95%
## -0.2320389  0.2300859
```

We can see that the posterior is centered near zero, with an expected difference of -0.003 and a very wide 90% CI. One might report this in a manuscript by stating that, "Little to no evidence was found for stronger directional selection on personality as compared to plasticity ($\Delta\beta = -0.00,\ \mathrm{MAD} = 0.14,\ 90\%\mathrm{CI}[-0.23, 0.23],\ p_{+} = 0.48$), suggesting that these RN parameters have equal effects on individual fitness."

If one is so inclined, robust null hypothesis tests can also be conducted within a Bayesian framework by specifying a range of biologically trivial effect sizes. For example, on a standardized scale such as a regression coefficient, values $-0.10 < r < 0.10$ are considered extremely small, explaining less than 1% of variance in a measure. We might, therefore, think of these as "trivial hypothesis" tests rather than null hypotheses per se. We can comparing these trivial hypothesis tests for the RN parameter correlations with directional hypothesis tests to get distinct pieces of information.

```
R=samples$R[,,] #3d array, 1 dim = samples, 2 dim = rows, 3 dim = columns

#directional hypothesis tests
sum(R[,1,2]>0)/length(R[,1,2]) #cor(personality, plasticity)
```

```
## [1] 0.9993333
```

```
sum(R[,1,3]>0)/length(R[,1,3]) #cor(personality, predictability)
```

```
## [1] 0.9638333
```

```
sum(R[,2,3]>0)/length(R[,2,3]) #cor(plasticity, predictability)
```

```
## [1] 0.9993333
```

```
#trivial hypothesis tests
sum(-0.1< R[,1,2] & R[,1,2]  <0.1)/length(R[,1,2]) #cor(personality, plasticity)
```

```
## [1] 0.03383333
```

```
sum(-0.1< R[,1,3] & R[,1,3]  <0.1)/length(R[,1,3]) #cor(personality, predictability)
```

```
## [1] 0.3423333
```

```
sum(-0.1< R[,2,3] & R[,2,3]  <0.1)/length(R[,2,3]) #cor(plasticity, predictability)
```

```
## [1] 0.0195
```

Together, these directional and trivial hypothesis tests provide very useful additional sources of information for interpreting the direction and magnitude of the estimated effects. Firstly, we see clear evidence of positive effect sizes for all RN parameter correlations (probability 0.96-0.99), suggesting that personality, plasticity, and predictability are not developing independently among individuals (and may constrain microevolutionary trajectories as a result). We also see that the correlations between personality and plasticity and between plasticity and predictability are very unlikely to take on trivial effect sizes (probability 0.02-0.03), and thus may be worthy of further attention to explain how and why these correlations arise. However, there is an ~1/3 chance of the correlation between personality and predictability being trivially small, or put the other way, only a 0.66 probability of a non-trivial effect size. Therefore, while there is evidence of a positive correlation between these RN parameters, we cannot be confident that this is association is of a biologically meaningful magnitude.

Calculate selection differentials

We now want to calculate the total Δ_{T} and direct Δ_{D} selection differentials on RN parameters, which allows us to consider both the total expected change due to direct and indirect selection as well as the hypothetical expected change due solely to direct selection effects. The total differentials are crucial for capturing patterns of evolutionary constrain or facilitation due to phenotypic integration, while the direct differentials are crucial for testing adaptive hypotheses irrespective of integration among traits. As proposed and explained in Martin (2021), these differentials are calculated by

$$\Delta_{\mathrm{T}}\bar{\boldsymbol{z}}_{\mathbf{p}} = \mathbf{P}\boldsymbol{\beta},$$
$$\Delta_{\mathrm{T}}\mathbf{P} = \mathbf{P}\left(\boldsymbol{\gamma} - \boldsymbol{\beta}\boldsymbol{\beta}'\right)\mathbf{P}$$

$$\Delta_{\mathrm{D}}\bar{\boldsymbol{z}}_{\mathbf{p}} = \mathbf{V}\boldsymbol{\beta},$$
$$\Delta_{\mathrm{D}}\mathbf{V} = \mathbf{V}\left(\boldsymbol{\gamma} - \boldsymbol{\beta}\boldsymbol{\beta}'\right)\mathbf{V}$$

The first step in calculating selection differentials is to appropriately square the quadratic selection effects (Stinchcombe et al. 2008) and to create lists containing each posterior sample of the linear vector $\boldsymbol{\beta}$ and the nonlinear selection matrix $\boldsymbol{\gamma}$. This will ensure that posterior uncertainty in the selection coefficients is pooled across all stages of analysis (Stinchcombe, Simonsen, and Blows 2014).

```
betas = samples$betas #all coefficients
direct = betas[,1:3] #directional gradients
quad = betas[,4:6]^2 #square to get quadratic gradients
cor = betas[,7:9] #correlational gradients

#create beta vector
beta_vec = list() #initialize list of vectors
for(i in 1:nrow(direct)) {
  beta_vec[[i]] = matrix(direct[i,],nrow = 3, ncol = 1)
  }

#create gamma matrix
gamma_mat = list() #initialize list of matrices
for(i in 1:nrow(quad)) {
  #diagonal with quad gradients
  gamma = diag(quad[i,])
  #add in off-diagonal elements
  gamma[1,2] = cor[i,1] #personality, plasticity
  gamma[1,3] = cor[i,2] #personality, predictability
  gamma[2,3] = cor[i,3] #plasticity, personality
  #make symmetric
  gamma[lower.tri(gamma)] = t(gamma)[lower.tri(gamma)]
  #add to list
  gamma_mat[[i]] = gamma
  }
```

We then need to create lists of the matrices **P** and **V** for the total and direct selection differentials respectively.

```
P = samples$P

#create list of P matrix
P_mat = list()
for(i in 1:nrow(P)){
  P_mat[[i]] = P[i,,]
  }

#create list of V matrix
V_mat = list()
for(i in 1:nrow(P)){
  V_mat[[i]] = diag(diag(P[i,,]))
  }
```

The posteriors of the Δ_{T} and Δ_{D} differentials can now be calculated.

```
#change in mean
dT_mean = Map('%*%',P_mat,beta_vec)
dD_mean = Map('%*%',V_mat,beta_vec)

#change in (co)variance
dT_vcv = Map('%*%',
          Map('%*%', P_mat,
             Map('-',gamma_mat,
                 Map('%*%',beta_vec,lapply(beta_vec,t) ))),
          P_mat )
dD_vcv = Map('%*%',
          Map('%*%', V_mat,
             Map('-',gamma_mat,
                 Map('%*%',beta_vec,lapply(beta_vec,t) ))),
          V_mat )
```

Let's calculate the expectations and uncertainty of the mean differentials.

```
#expected total mean change (personality, plasticity, predictability)
apply(simplify2array(dT_mean), 1:2, mean)
```

```
##             [,1]
## [1,] 0.10865359
## [2,] 0.08432013
## [3,] 0.08924963
```

```
#90% CI for mean change
apply(simplify2array(dT_mean), 1:2, function(x) quantile(x,c(0.05,0.95)))
```

```
## , , 1
##
##           [,1]       [,2]       [,3]
## 5%  0.06512052 0.03778112 0.04838245
## 95% 0.15569664 0.13134327 0.13111724
```

```
#expected direct mean change
apply(simplify2array(dD_mean), 1:2, mean)
```

```
##             [,1]
## [1,] 0.08745891
## [2,] 0.04922922
## [3,] 0.06722475
```

```
#90% CI for mean change
apply(simplify2array(dD_mean), 1:2, function(x) quantile(x,c(0.05,0.95)))
```

```
## , , 1
##
##           [,1]        [,2]       [,3]
## 5%  0.04408368 0.001632592 0.02447695
## 95% 0.13413236 0.097099634 0.10991089
```

There is clear evidence that selection is acting to increase the mean levels of the RN parameters for personality, plasticity, and predictability in the population, both directly and indirectly through the effects of phenotypic integration. It is important to note that the predictability parameters $\boldsymbol{\theta}^{(z)}$ are defined formally such that larger values lead to greater residual variance in individual behavior. Therefore, increasing the mean level of this parameter will lead to *less* predictable behavioral responses (i.e. the residual variance gets larger). For the variances and covariances of these parameters

```
#expected total mean change (personality, plasticity, predictability)
apply(simplify2array(dT_vcv), 1:2, median)
```

```
##            [,1]       [,2]       [,3]
## [1,] 0.02387443 0.03732750 0.04186769
## [2,] 0.03732750 0.03501465 0.04374964
## [3,] 0.04186769 0.04374964 0.02006556
```

```
#90% CI for mean change
apply(simplify2array(dT_vcv), 1:2, function(x) quantile(x,c(0.05,0.95)))
```

```
## , , 1
##
##            [,1]        [,2]       [,3]
## 5%  0.003733472 0.009451298 0.01536705
## 95% 0.049193036 0.067161180 0.07036538
##
## , , 2
##
##            [,1]       [,2]       [,3]
## 5%  0.009451298 0.01495914 0.01420889
## 95% 0.067161180 0.06209723 0.07925445
##
## , , 3
##
##           [,1]       [,2]        [,3]
## 5%  0.01536705 0.01420889 0.002576406
## 95% 0.07036538 0.07925445 0.045640251
```

```
#expected direct mean change
apply(simplify2array(dD_vcv), 1:2, median)
```

```
##             [,1]        [,2]        [,3]
## [1,] 0.004389223 0.018730889  0.02645083
## [2,] 0.018730889 0.005743641  0.03149115
## [3,] 0.026450831 0.031491153 -0.00224727
```

```
#90% CI for mean change
apply(simplify2array(dD_vcv), 1:2, function(x) quantile(x,c(0.05,0.95)))
```

```
## , , 1
##
##             [,1]         [,2]         [,3]
## 5%  -0.008899966 -0.009712015 -0.001034179
## 95%  0.023906066  0.048163066  0.055253205
##
## , , 2
##
##             [,1]         [,2]          [,3]
## 5%  -0.009712015 -0.004209494 -0.0006163572
## 95%  0.048163066  0.030693024  0.0658182535
##
## , , 3
```

```
##
##               [,1]          [,2]         [,3]
## 5%  -0.001034179 -0.0006163572 -0.010198105
## 95%  0.055253205  0.0658182535  0.008096716
```

Note that the 90% CIs are calculated across the columns of the $\Delta_{\mathrm{T}}\mathbf{P}$ and $\Delta_{\mathrm{D}}\mathbf{P}$ matrices. We find evidence that both direct and indirect selection are increasing the correlations of the RN parameters. While it is also expected that the total effect of selection will increase trait variance, the direct effect of selection on trait variances is less certain and much smaller. It is notable as well that the effects of direct selection are expected to be slightly negative thought extremely small for the variance of predictability. Phenotypic integration is thus leading to greater maintenance of individual variation and trait correlations than would otherwise be expected by the direct effects of selection alone.

Plot results

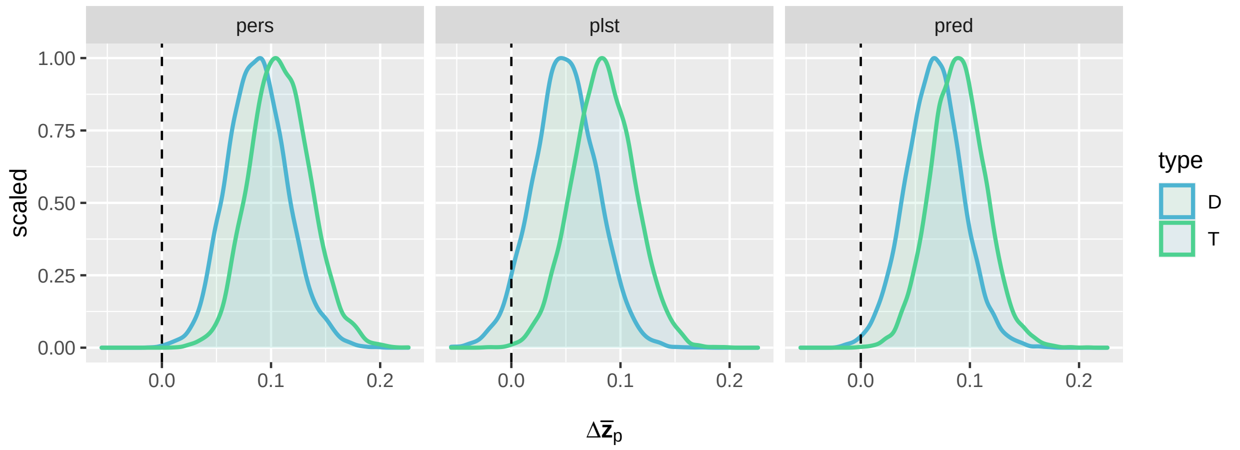
While it is helpful to summarize the differentials with point estimates, the models provide posterior distributions that can be also plotted and visually interpreted to gain a fuller sense of the uncertainty in these estimates. We'd also like to know how the expected within-generation changes of each RN parameter will change the overall shape of the behavioral RN within the population. First we'll consider plotting the differentials for RN parameters. As argued in Martin (2021), it is helpful to separately visualize the expected change in means, trait variance, and trait correlations, as each provides unique information relevant for testing adaptive hypotheses. First we need to create dataframes in long format for plotting.

```r
#change in means (list to matrix)
dT_mean = matrix(unlist(dT_mean), nrow=length(dT_mean), ncol=3, byrow=TRUE,
                 dimnames = list(1:length(dT_mean), c("pers","plst","pred")))
dD_mean = matrix(unlist(dD_mean), nrow=length(dD_mean), ncol=3, byrow=TRUE,
                 dimnames = list(1:length(dD_mean), c("pers","plst","pred")))

#separate change in variance from covariance
dT_V = matrix(unlist(lapply(dT_vcv,diag)), nrow=length(dT_vcv), ncol=3, byrow=TRUE,
              dimnames = list(1:length(dT_vcv), c("pers","plst","pred")))
#dT_S = sqrt(abs(dT_V))*sign(dT_V) #change in SD (abs to avoid negative sqrt)
dD_V = matrix(unlist(lapply(dD_vcv,diag)), nrow=length(dD_vcv), ncol=3, byrow=TRUE,
              dimnames = list(1:length(dD_vcv), c("pers","plst","pred")))
#dD_S = sqrt(abs(dD_V))*sign(dD_V)

#change in correlations
dT_R = matrix(unlist(lapply(dT_vcv,FUN = function(x) x[lower.tri(x)])),
                    nrow=length(dT_vcv), ncol=3, byrow=TRUE,
              dimnames = list(1:length(dT_vcv), c("pers_plst","pers_pred","plst_pred")))
dD_R = matrix(unlist(lapply(dD_vcv,FUN = function(x) x[lower.tri(x)])),
                    nrow=length(dD_vcv), ncol=3, byrow=TRUE,
              dimnames = list(1:length(dD_vcv), c("pers_plst","pers_pred","plst_pred")))

#wide to long for plotting
library(reshape)
dT_meanl = melt(dT_mean)
dD_meanl = melt(dD_mean)
d_meanl = rbind(dT_meanl,dD_meanl) #combine
d_meanl$type = rep(c("T","D"), each = nrow(dT_meanl))

dT_Vl = melt(dT_V)
dD_Vl = melt(dD_V)
d_Vl = rbind(dT_Vl,dD_Vl)
```

```
d_Vl$type = rep(c("T","D"), each = nrow(dT_Vl))

dT_Rl = melt(dT_R)
dD_Rl = melt(dD_R)
d_Rl = rbind(dT_Rl,dD_Rl)
d_Rl$type = rep(c("T","D"), each = nrow(dT_Rl))
```

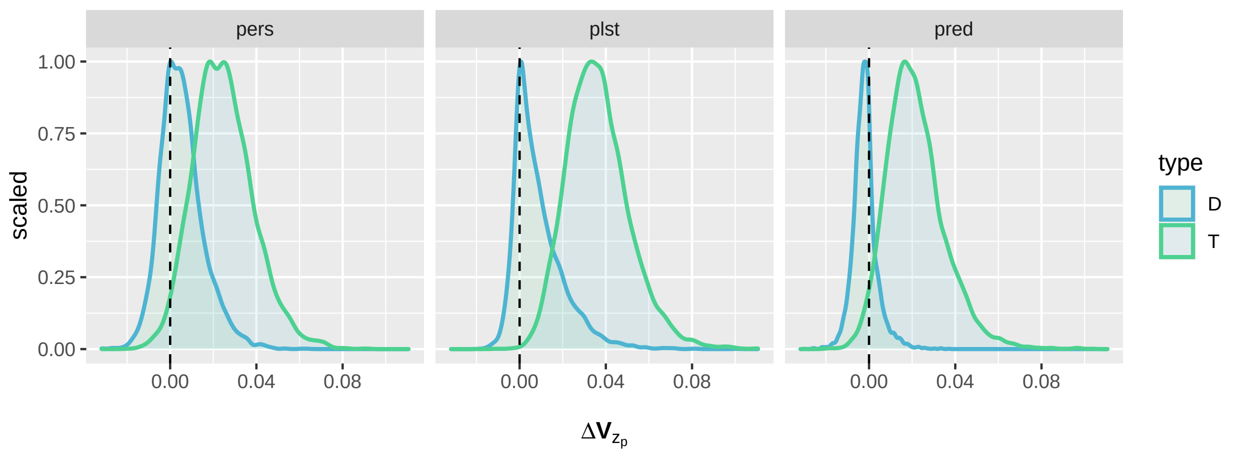
561 The data are now ready for plotting.

```
library(ggplot2)

#mean differentials
ggplot(d_meanl, aes( x = value, color = type, fill = type)) +
  geom_density(aes(y=..scaled..), size = 0.9, alpha = 0.10) +
  facet_grid(. ~ X2)+
  scale_color_manual(values=c("#4db4d1","#4dd191"))+
  geom_vline(xintercept = 0, linetype = "dashed", size = 0.5)+
  xlab( bquote(atop("", paste(Delta,bold(bar(z))[p] ))))+
  scale_fill_manual(values=c("#4dd191","#4db4d1"))
```



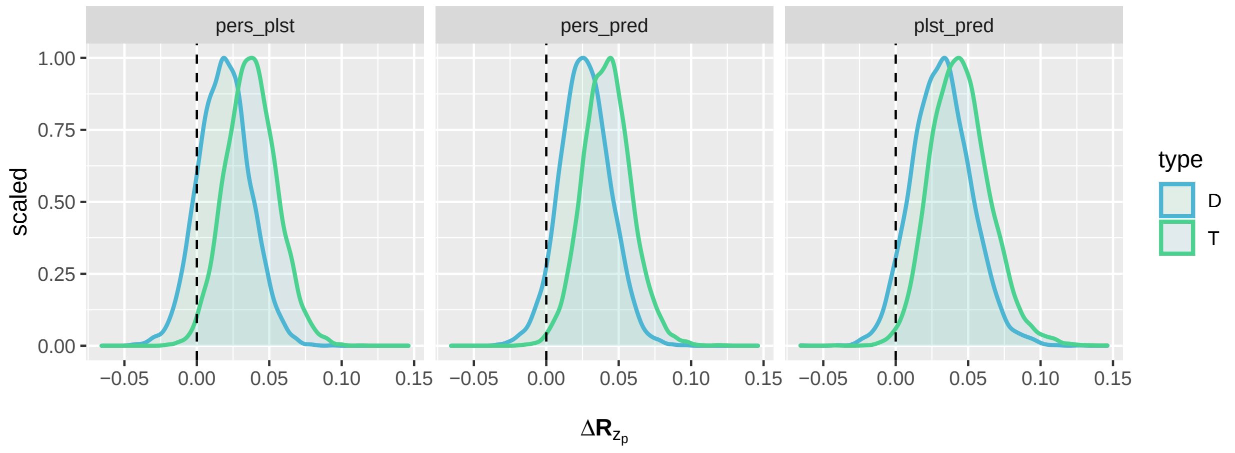
562

```
library(ggplot2)

#variance differentials
ggplot(d_Vl, aes( x = value, color = type, fill = type)) +
  geom_density(aes(y=..scaled..), size = 0.9, alpha = 0.10) +
  facet_grid(. ~ X2)+
  scale_color_manual(values=c("#4db4d1","#4dd191"))+
  geom_vline(xintercept = 0, linetype = "dashed", size = 0.5)+
  xlab( bquote(atop("",paste(Delta,bold(V)[z[p]] ))))+
  scale_fill_manual(values=c("#4dd191","#4db4d1"))
```



563

```
library(ggplot2)

#correlation differentials
ggplot(d_R1, aes( x = value, color = type, fill = type)) +
  geom_density(aes(y=..scaled..), size = 0.9, alpha = 0.10) +
  facet_grid(. ~ X2)+
  scale_color_manual(values=c("#4db4d1","#4dd191"))+
  geom_vline(xintercept = 0, linetype = "dashed", size = 0.5)+
  xlab( bquote(atop("",paste(Delta,bold(R)[z[p]] ))))+
  scale_fill_manual(values=c("#4dd191","#4db4d1"))
```



564

To visualize the full population RNs, we need to first add the original population values $\mu_0^{(z)}$, $\beta_1^{(z)}$, and $\theta_0^{(z)}$ to the Δ_T and Δ_D to get absolute values following the selection event. We can then use point estimates of these posteriors to generate a single plot of the RN function.

```
#personality (pop intercept mu)
mu = median(samples$mu_0z) #linear w/o link function
T_mu = median(dT_mean[,1]) #following selection
D_mu = median(dD_mean[,1])

#plasticity (pop slope beta)
beta = median(samples$beta_1z)
T_beta = median(samples$beta_1z + dT_mean[,2])
D_beta = median(samples$beta_1z + dD_mean[,2])
```

```r
#predictability (sigma)
theta = median(exp(samples$theta_0z))
#exp inverse link on absolute value on the log scale
T_theta = median(exp(samples$theta_0z + dT_mean[,3]))
D_theta = median(exp(samples$theta_0z + dD_mean[,3]))
```

568 We create an arbitrary environmental covariate for visualizing the population RN before and after selection

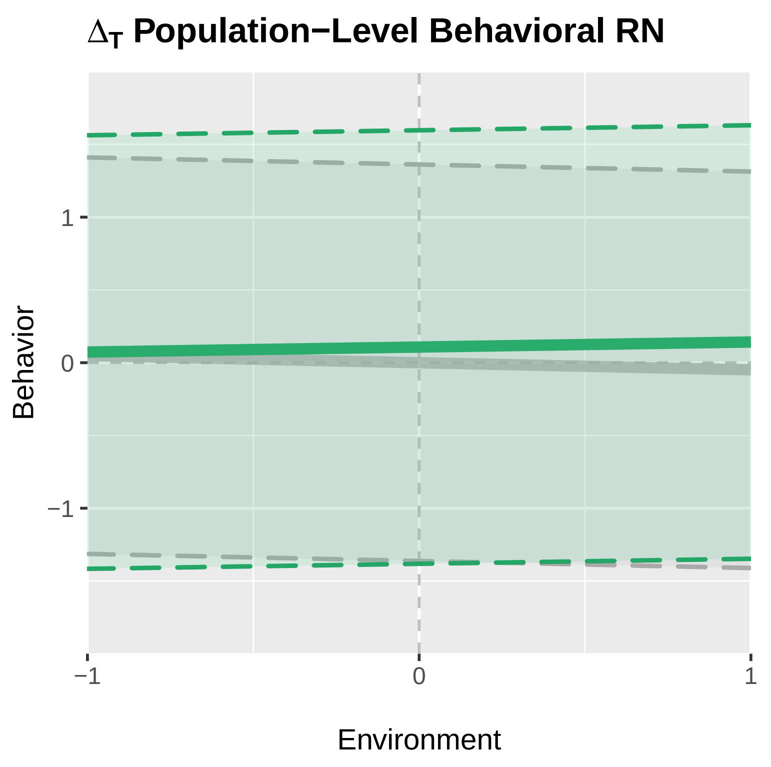
```r
x = seq(-1,1,by = 0.05)
```

569 and calculate the 90% or 95% CI for the RN.

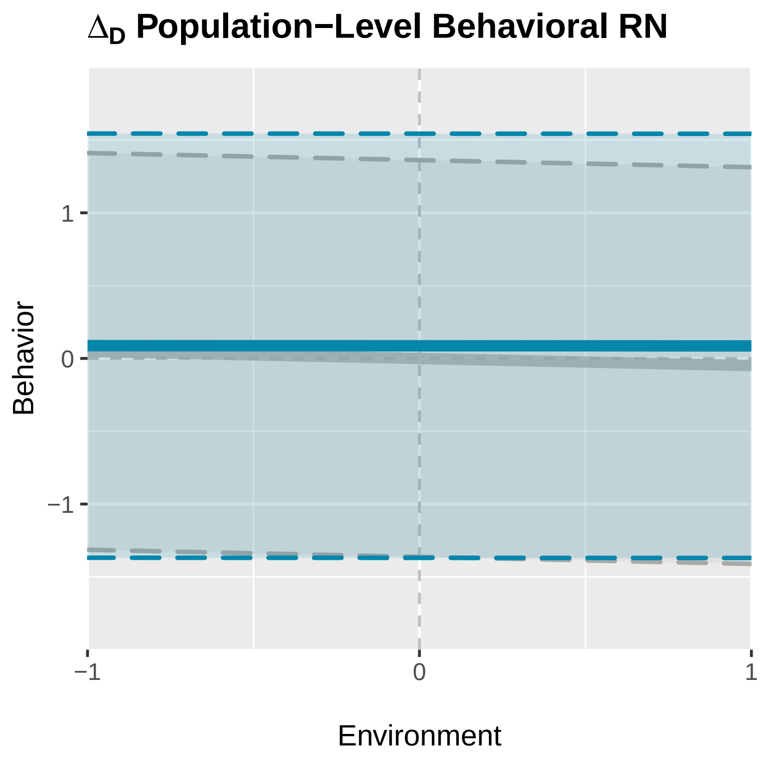
```r
#before selection
y_low = mu + beta*x -1.96*theta #1.96 = 95% CI
y_high = mu + beta*x +1.96*theta

#after selection
Ty_low = T_mu + T_beta*x -1.96*T_theta
Ty_high = T_mu + T_beta*x +1.96*T_theta

Dy_low = D_mu + D_beta*x -1.96*D_theta
Dy_high = D_mu + D_beta*x +1.96*D_theta
```

570 We're now ready for plotting.

```r
ggplot() +
  coord_cartesian(xlim=c(-1, 1), ylim=c(-2, 2)) +
  scale_x_continuous(expand = c(0, 0), breaks = c(-1,0,1),
                     labels = c(-1,0,1) )+
  scale_y_continuous(expand = c(0, 0), breaks = c(-1,0,1),
                     labels = c(-1,0,1) ) +
  geom_hline(yintercept=0,linetype="dashed", alpha = 0.25)+
  geom_vline(xintercept=0,linetype="dashed", alpha = 0.25) +

  geom_abline(intercept = mu, slope = beta, size = 2, alpha =0.75, color = "darkgrey") +
  geom_ribbon(aes(x = x, y = mu + beta*x, ymin = y_low, ymax = y_high),
              size = 0.8, linetype = 5, alpha = 0.15, color = "darkgrey", fill = "darkgrey")+

  geom_abline(intercept = T_mu, slope = T_beta, size = 2, color = "#23a666") +
  geom_ribbon(aes(x = x, y = T_mu + T_beta*x, ymin = Ty_low, ymax = Ty_high),
              size = 0.8, linetype = 5, alpha = 0.15, color = "#23a666", fill = "#4dd191")+
  xlab("\nEnvironment")+
  ylab("Behavior")+
  ggtitle(expression(paste(bold(Delta[T]),bold(" Population-Level Behavioral RN"))))+
        guides(fill=FALSE, color=FALSE)
```



```
ggplot() +
  coord_cartesian(xlim=c(-1, 1), ylim=c(-2, 2)) +
  scale_x_continuous(expand = c(0, 0), breaks = c(-1,0,1),
                     labels = c(-1,0,1) )+
  scale_y_continuous(expand = c(0, 0), breaks = c(-1,0,1),
                     labels = c(-1,0,1) ) +
  geom_hline(yintercept=0,linetype="dashed", alpha = 0.25)+
  geom_vline(xintercept=0,linetype="dashed", alpha = 0.25) +

  geom_abline(intercept = mu, slope = beta, size = 2, alpha =0.75, color = "darkgrey") +
  geom_ribbon(aes(x = x, y = mu + beta*x, ymin = y_low, ymax = y_high),
            size = 0.8, linetype = 5, alpha = 0.15, color = "darkgrey", fill = "darkgrey")+

  geom_abline(intercept = D_mu, slope = D_beta, size = 2, color = "#0586ab") +
  geom_ribbon(aes(x = x, y = D_mu + D_beta*x, ymin = Dy_low, ymax = Dy_high),
            size = 0.8, linetype = 5, alpha = 0.15, color = "#0586ab", fill = "#0586ab")+
  xlab("\nEnvironment")+
  ylab("Behavior")+
  ggtitle(expression(paste(bold(Delta[D]),bold(" Population-Level Behavioral RN"))))+
      guides(fill=FALSE, color=FALSE)
```



As in the main text, grey is used to indicate the current population RN. Overall, we can see that direct selection is expected to have a fairly negligible impact on the population RN, with total selection leading to a small but more apparent increase in personality (RN intercept) and plasticity (RN slope). For both direct and total selection, there is also an increase in the magnitude of the predictability parameter (RN dispersion/shaded band), suggesting that individuals' behavior will become slightly more random in the population.

Forthcoming tutorials

Further examples will be added in the future for simplifying the full model (e.g. only considering selection on personality), estimating non-Gaussian response models and selection gradients, introducing repeated fitness measures, and including structural equation models.

References

Carpenter, B., A. Gelman, M. D. Hoffman, D. Lee, B. Goodrich, M. Betancourt, and... A. Riddell. 2017. "Stan: A Probabilistic Programming Language." *Journal of Statistical Software* 74. https://www.jstatsoft.org/article/view/v076i01.

Hoffman, M. D., and A. Gelman. 2014. "The No-u-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo." *Journal of Machine Learning Research* 15: 1593–623.

Lemoine, N. P. 2019. "Moving Beyond Noninformative Priors: Why and How to Choose Weakly Informative Priors in Bayesian Analyses." *Oikos* 128. https://onlinelibrary.wiley.com/doi/full/10.1111/oik.05985.

Link, W. A., and M. J. Eaton. 2012. "On Thinning of Chains in MCMC." *Methods in Ecology and Evolution* 3: 112–15.

Martin, J. S. 2021. "Estimating Nonlinear Selection on Behavioral Reaction Norms." *BioRxiv Preprint* XX: XX–.

McElreath, R. 2020. *Statistical Rethinking: A Bayesian Course with Examples in r and Stan.* 2nd ed. CRC Press. https://xcelab.net/rm/statistical-rethinking/.

McShane, B. B., D. Gal, A. Gelman, C. Robert, and J. L. Tackett. 2019. "Abandon Statistical Significance." *The American Naturalist* 73: 235–45.

R Core Team. 2020. *R: A Language and Environment for Statistical Computing.* Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org.

Stinchcombe, J. R., A. F. Agrawal, P. A. Hohenlohe, S. J. Arnold, and M. W. Blows. 2008. "Estimating Nonlinear Selection Gradients Using Quadratic Regression Coefficients: Double or Nothing?" *Evolution* 68. https://onlinelibrary.wiley.com/doi/full/10.1111/evo.12321.

Stinchcombe, J. R., A. K. Simonsen, and M. W. Blows. 2014. "Estimating Uncertainty in Multivariate Responses to Selection." *Evolution* 68. https://onlinelibrary.wiley.com/doi/full/10.1111/evo.12321.