Supplementary text S8: Practical guide

This document is a supplement to the paper "Most published selection gradients are underestimated: why this is and how to fix it". It describes how we fitted i) classic Lande-Arnold regressions, ii) multivariate mixed-effects models, and iii) errors-in-variables models to investigate their ability to produce accurate and precise estimates. It also decribes iv) how we calculated corrections and applied them to attenuated estimates derived from classic Lande-Arnold regressions. We achieved this by means of data simulation.

Linear selection scenario

We start with creating a simulated dataset of individuals assayed multiple times for a fixed trait to which we then add error to generate a target trait repeatability (R=0.3 or R=0.7). Each individual is assigned a fitness value based on a pre-set relationship between traits and fitness.

```
n.ind <- 800  ## Number of individuals

V_m.error <- 7  ## Within-individual variance in the trait
V.trait <- 3  ## Among-individual variance in the trait
mean.trait <- 0  ## Population-mean trait value
w.bar <- 2  ## Population-mean fitness

B1_z <- 0.3  ## True standardized selection gradient

B1_t <- B1_z/sqrt(V.trait) * w.bar  ## True unstandardized selection gradient

## Generate data set
d <- data.frame(ID = 1:n.ind, t = rnorm(n.ind, mean.trait, sqrt(V.trait)))
d$z <- scale(d$t)  # Scale the trait

# Simulate fitness
d$w <- w.bar + d$t * B1_t + rnorm(n.ind, 0, sqrt(0.1)) - mean(d$t * B1_t + rnorm(n.ind,
    0, sqrt(0.1)))
d$rw <- d$w/mean(d$w)  ## Transform absolute fitness to relative fitness

# Simulate 3 measurements of the trait per individual
d$obs.t1 <- rnorm(n.ind, d$t, sqrt(V_m.error))
d$obs.t2 <- rnorm(n.ind, d$t, sqrt(V_m.error))
d$obs.t3 <- rnorm(n.ind, d$t, sqrt(V_m.error))

# Estimate means and standard deviations
m <- mean(c(d$obs.t1, d$obs.t2, d$obs.t3))
sd <- sd(c(d$obs.t1, d$obs.t2, d$obs.t3))

# Standardize trait values
d$obs.z1 <- (d$obs.t1 - m)/sd
d$obs.z2 <- (d$obs.t2 - m)/sd
d$obs.z3 <- (d$obs.t3 - m)/sd

# Calculate a mean value for each individual over its 3 measurements
d$mean.t <- apply(d[, c("obs.t1", "obs.t2", "obs.t3")], 1, mean)
d$mean.z <- scale(d$mean.t)
```

Simulated data set

Data set 1 (d) has the "real" values for the phenotype (t) and the standardized trait (z). The data also contains the raw measurements taken by the hypothetical researcher (obs.t1 to obs.t3) followed by their

standardized values (obs.z1 to obs.z3). The last columns are the estimated means for the three measurements (mean.t and mean.z), w is absolute fitness and rw relative fitness.

```
head(d)
```

```
##   ID          t          z        w        rw     obs.t1    obs.t2
## 1  1 -1.2769070 -0.6723787 2.430981 1.2094363  2.2180894 -7.776166
## 2  2 -0.9686534 -0.4934013 1.504754 0.7486296 -0.8203357 -1.886769
## 3  3  0.3458197  0.2698049 1.886902 0.9387517 -0.7987428 -4.231165
## 4  4  0.9677422  0.6309040 2.874306 1.4299950  0.7517050  1.289563
## 5  5  0.3492557  0.2717999 1.982787 0.9864559 -5.2044925  2.802198
## 6  6 -0.4423378 -0.1878132 2.257220 1.1229886  3.3419426  3.282130
##       obs.t3     obs.z1     obs.z2     obs.z3     mean.t     mean.z
## 1 -0.9940796  0.7621115 -2.3960170 -0.2529159 -2.1840520 -0.8594708
## 2 -1.4615695 -0.1980138 -0.5350008 -0.4006400 -1.3895582 -0.5163939
## 3 -4.2604331 -0.1911905 -1.2758168 -1.2850652 -3.0967804 -1.2536036
## 4  2.4167146  0.2987423  0.4687024  0.8248760  1.4859942  0.7253221
## 5  0.6771973 -1.5833827  0.9466866  0.2751983 -0.5750324 -0.1646668
## 6  2.7182279  1.1172428  1.0983423  0.9201525  3.1141001  1.4283679
```

Expected selection gradient

Because we set the true selection gradient, we can study how well a particular method works in estimating it without bias. Importantly, each simulation run represents just a sample of an infinite population characterized by the true value; sampling variance will therefore create variation in true parameter values across runs. As a predictor of the "true" standardized selection gradients, we extract the "true" gradient from the sample using a simple regression.

```
mod <- lm(rw ~ z, data = d)
coef(mod)[2]
```

```
##         z
## 0.3048345
```

In the next section, we use simulations to show that selection gradient estimates are attenuated when analyses do not account for biasing effects of within-individual variation in trait values.

Univariate model analyses with one observation per individual

We first fit a model with relative fitness (rw) as the response variable and the standardized trait values (obs.z1) as the predictor.

```
mod1 <- lm(rw ~ obs.z1, data = d)
coef(mod1)[2]
```

```
##    obs.z1
## 0.1640798
```

We can estimate trait repeatability because we had created repeated measures. This allows us to correct the attenuated estimate using Equation 5 (Main Text). We first need to convert the data to a long format.

$$\beta_1 = \frac{\beta_1^*}{\sqrt{R_t}}$$

```
dt <- gather(d, obs, t, obs.t1:obs.t3, factor_key = TRUE)
dt$ID <- as.factor(dt$ID)
```

```
modR2 <- lmer(t ~ 1 + (1 | ID), data = dt)
VI2 <- VarCorr(modR2)$ID[1, 1]
VR2 <- attr(VarCorr(modR2), "sc")^2
Rt <- VI2/(VI2 + VR2)
coef(mod1)[2]/sqrt(Rt)
```

```
##    obs.z1
## 0.2980599
```

Univariate model analyses using a mean of three values per individual

Next, we estimate each individual's mean trait value over its 3 measurments, and continue by fitting a model with relative fitness (rw) as the response variable and individual-mean trait values (mean.z) as the predictor. Note that we standardized trait values after calculating individual-mean trait values as discussed in Supplementary Text S5.

```
mod.mean <- lm(rw ~ mean.z, data = d)
coef(mod.mean)[2]
```

```
##   mean.z
## 0.226563
```

Estimates of among- and within-individual variance derived from the mixed model (previously used to calculate repeatability) are used to correct the attenuated estimate using the formula presented in Table 1 (Main Text; see also Supplementary Text S5).

$$\beta_1 = \frac{\beta_1^*}{\sqrt{\frac{V_{i_t}}{V_{i_t} + \frac{V_{e_t}}{N}}}}$$

```
coef(mod.mean)[2]/sqrt(VI2/(VI2 + VR2/3))
```

```
##    mean.z
## 0.3011348
```

Bivariate mixed-effect model

Because we had generated repeated measures data, we were also able to fit a bivariate mixed-effects model with individual intercepts, where the trait and fitness were fitted as the two response variables. Because we have repeated measures for the trait but only one measure of fitness, we need to "inform" the model that not all parameters are estimable. There are multiple solutions for this problem. Thomson et al. 2020 (Evolution 71: 716-732) suggest to use prior option covu=TRUE in MCMCglmm. This requires to use the long format of the data set in order to estimate the covariance between the mean value of the trait and the unique value of fitness.

```
dw <- d[, c("ID", "w")]
dw$measure <- "w"
dw$type <- "s"
dw$time <- NA
colnames(dw)[2] <- "t"

dt <- dt[, c("ID", "t")]
dt$measure <- "t1"
dt$type <- "r"
dt$time <- rep(1:3, each = 800)
```

```
d3 <- rbind(dw, dt)
d3$measure <- as.factor(d3$measure)
d3$type <- as.factor(d3$type)

prior0 <- list(R = list(R1 = list(V = diag(2), nu = 2, covu = TRUE), R2 = list(V = diag(1),
    nu = 0.002)))
modbi <- MCMCglmm(t ~ measure - 1, random = ~us(at.level(type, "r")):ID, rcov = ~us(at.level(type,
    "s")):ID + us(at.level(type, "r")):ID:time, data = d3, prior = prior0, nitt = 105000,
    burnin = 5000, thin = 100, verbose = FALSE)
```

```
## Warning in buildZ(rmodel.terms[r], data = data): missing values in random
## predictors
```

We then calculated standardized selection gradient using the estimated among-individual covariance between the trait and fitness, the mean fitness of the population (represented by the model's intercept), and the estimated among-individual trait variance (Equation 11; Main Text).

$$\beta_1 = \frac{C_{i_{t,w}}}{V_{i_t}} \frac{\sqrt{V_{i_t}}}{\beta_{0_w}}$$

```
Bz <- posterior.mode(modbi$VCV[, 2]/modbi$VCV[, 1] * sqrt(modbi$VCV[, 1])/modbi$Sol[,
    2])
Bz
```

```
##      var1
## 0.2939657
```

Errors-in-variables model

We use Rstan to parameterize errors-in-variables models. This requires writing a model that defines the data, parameters, and likelihood function. The code provided below also estimates downstream parameters (e.g., standardized selection gradient). The model is written in a file that connects to stan; note that this stan-code differs slightly from standard R-code. For instance comments are denoted by // instead of #. Here, we first present the code defining the variables.

```
"data {
   // Define variables in data
   int<lower=0> N;
  // Number of individuals (an integer)
  int<lower=0> nind;
  // Individual IDs
  int<lower=0> ID[N];
  // Continuous outcomes, the trait and fitness
   real t[N];
   real w[nind];
 }"
```

We then continued by defining the parameters to be estimated: the population-mean intercept for the phenotype β_{0_z}, for fitness β_{0_w} and the unstandardized selection gradient b_1. We also define the individual-specific values for the phenotype I_{i_t}, the standard deviation for the individual intercepts $\sqrt{V_{i_t}}$, and residual standard deviations in phenotypes $\sqrt{V_{e_t}}$ and fitness $\sqrt{V_{e_w}}$.

```
"parameters {
   // Define parameters to estimate
   // Population-level intercepts (a real number) and slopes
  real beta_0z;
```

```
  real beta_0w;
  real b_1;

   // Random effects
  real I[nind];

   // Standard deviations
  real<lower=0> sigma_i;
  real<lower=0> sigma_e;
  real<lower=0> sigma_w;
 }"
```

Next, we write the model, where we define various predicted values: μ_z is the predicted values for the phenotypic observations and μ_w the predicted values for fitness. The observed phenotypes and fitness result from a normal distribution with mean 0, μ_z and μ_w, and standard deviations σ_i, σ_e and σ_w, respectively.

$$I \sim N(0, \sigma_i)$$
$$z \sim N(\mu_z, \sigma_e)$$
$$w \sim N(\mu_w, \sigma_w)$$

```
"model {
   real mu_t[N];
   real mu_w[nind];

   for (j in 1:nind) {
     mu_t[j]  = beta_0t + I[j];
     mu_w[j] = beta_0w + b_1*I[j];
   }

   // Prior part of Bayesian inference
   // Flat prior for mu (no need to specify if non-informative)
   sigma_i ~ uniform(0, 10);
   sigma_e ~ uniform(0, 10);

   // Random effects distribution
   I ~ normal(0, sigma_i);

   // Likelihood part of Bayesian inference
   for (i in 1:N) {
     z[i] ~ normal(mu_t[ID[i]], sigma_e);
   }

 for (i in 1:nind) {
     w[i] ~ normal(mu_w[i], sigma_w);
   }
 }"
```

Finally, we calculate the generated quantities: standaradized selection gradients. This requires transforming the among-individual standard deviation into among-individual variance to be used to calculate the standardized selection gradient as:

$$\beta_1 = b_1^* \frac{\sqrt{V_{i_t}}}{\beta_{0_w}}$$

```
"generated quantities {
  real beta_1;
  real sigma2_i;
  sigma2_i = sigma_i^2;
  beta_1 = b_1/beta_0w*sigma_i;
}"
```

We then combine all components using the function "write", and run the model.

```
write(

"data {
   // Define variables in data
   // Number of observations (an integer)
  int<lower=0> N;
  // Number of clusters (an integer)
  int<lower=0> nind;
  // Cluster IDs
  int<lower=0> ID[N];
  // Continuous outcomes
   real t[N];
   real w[nind];
 }

 parameters {
   // Define parameters to estimate
   // Population intercepts (a real number) and slopes
  real beta_0t;
  real beta_0w;
  real b_1;

   // Random effects
  real I[nind];

   // Standard deviation of random effects
  real<lower=0> sigma_i;
   // Residual standard deviations
   real<lower=0> sigma_e;
   real<lower=0> sigma_w;
 }

model {
   real mu_t[N];
   real mu_w[nind];

   for (j in 1:nind) {
     mu_t[j]  = beta_0t + I[j];
     mu_w[j] = beta_0w + b_1*I[j];
   }

   // Prior part of Bayesian inference
   // Flat prior for mu (no need to specify if non-informative)
   sigma_i ~ uniform(0, 10);
   sigma_e ~ uniform(0, 10);
```

```
   // Random effects distribution
   I ~ normal(0, sigma_i);

   // Likelihood part of Bayesian inference
   for (i in 1:N) {
     t[i] ~ normal(mu_t[ID[i]], sigma_e);
   }

 for (i in 1:nind) {
     w[i] ~ normal(mu_w[i], sigma_w);
   }
 }

 generated quantities {
  real beta_z;
  real sigma2_i;

  sigma2_i = sigma_i^2;
  beta_z = b_1/beta_0w*sigma_i;
}"

, "ME1.stan")

dt$ID<-as.numeric(dt$ID)

stan_data <- list(t = dt$t,  nind = max(dt$ID),
                  ID=dt$ID,
                  w=d$w, N=nrow(dt))

inits <- function() list(sigma_i = runif(1, 0, 10),
                         sigma_e = runif(1, 0, 10),
                         sigma_w = runif(1, 0, 10))

#Parameters monitored
params <- c("sigma_i", "b_1", "sigma_e", "beta_z")

##MCMC sampling design
ni <- 3000
nt <- 2
nb <- 1000
nc <- 2

## Call Stan from R
md <- stan("ME1.stan",
           data = stan_data, init = inits, pars = params,
chains = nc, iter = ni, warmup = nb, thin = nt, cores=2, refresh=0)
```

```
summary(md)$summary[4, 1]
```

```
## [1] 0.3026918
```

Quadratic selection scenario

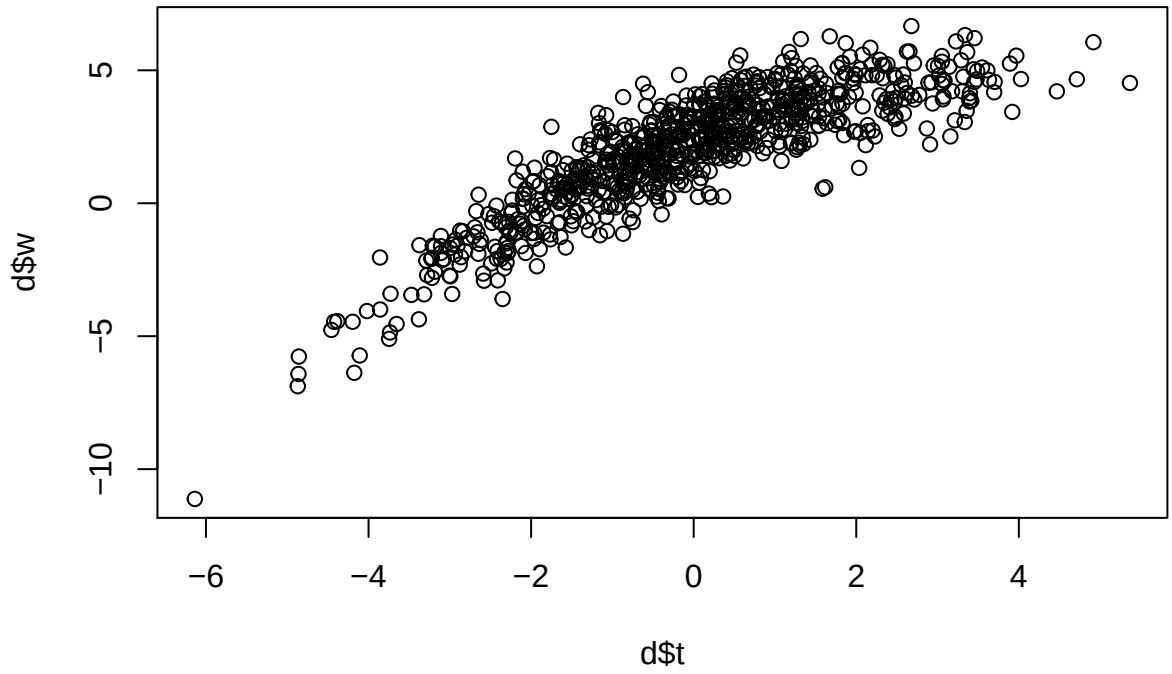
Here we simulate a quadratic selection analysis using similar approaches as detailed above for the linear

scenario. The dataset therefore includes the square of the trait (t2), and the standardized values of the square (z2).

```
# New simulation
n.ind <- 800

V_m.error <- 7
V.trait <- 3
mean.trait <- 0

B1_z <- 1.9
B2_z2 <- -0.3

B1_t <- B1_z/sqrt(V.trait)
B2_t2 <- B2_z2/sqrt(2 * (V.trait^2 + 2 * V.trait * mean.trait^2)) * w.bar


d <- data.frame(ID = 1:n.ind, t = rnorm(n.ind, mean.trait, sqrt(V.trait)))
d$t2 <- d$t^2
d$z <- scale(d$t)
d$z2 <- scale(d$t2)
d$w <- 2 + d$t * B1_t + d$t2 * B2_t2 + rnorm(n.ind, 0, sqrt(1)) - (mean(d$t *
    B1_t + d$t2 * B2_t2 + rnorm(n.ind, 0, sqrt(1))))
d$rw <- d$w/mean(d$w)

d$obs.t1 <- rnorm(n.ind, d$t, sqrt(V_m.error))
d$obs.t2 <- rnorm(n.ind, d$t, sqrt(V_m.error))
d$obs.t3 <- rnorm(n.ind, d$t, sqrt(V_m.error))

d$obs.t12 <- d$obs.t1^2
d$obs.t22 <- d$obs.t2^2
d$obs.t32 <- d$obs.t3^2

m <- mean(c(d$obs.t1, d$obs.t2, d$obs.t3))
sd <- sd(c(d$obs.t1, d$obs.t2, d$obs.t3))

m2 <- mean(c(d$obs.t12, d$obs.t22, d$obs.t33))
sd2 <- sd(c(d$obs.t12, d$obs.t22, d$obs.t33))

d$obs.z1 <- (d$obs.t1 - m)/sd
d$obs.z2 <- (d$obs.t2 - m)/sd
d$obs.z3 <- (d$obs.t3 - m)/sd

d$obs.z12 <- (d$obs.t12 - m2)/sd2
d$obs.z22 <- (d$obs.t22 - m2)/sd2
d$obs.z32 <- (d$obs.t32 - m2)/sd2

d$mean.t <- apply(d[, c("obs.t1", "obs.t2", "obs.t3")], 1, mean)
d$mean.t2 <- d$mean.t^2

d$mean.z <- scale(d$mean.t)
d$mean.z2 <- scale(d$mean.t2)
```

Here we visualize the fitness function.

```r
plot(d$w ~ d$t)
```



We can check the realized quadratic standardized selection gradient. As in the previous linear selection example, z and z2 are the "real" standarized values of the trait and its square.

```r
mod <- lm(rw ~ z + z2, data = d)
coef(mod)[3]
```

```
##         z2
## -0.3185942
```

Quadratic selection gradient with one value per individual

We estimate the quadratic selection gradient, which should be attenuated if the trait is not fully repeatable.

```r
mod1 <- lm(rw ~ obs.z1 + obs.z12, data = d)
coef(mod1)[3]
```

```
##     obs.z12
## -0.06143189
```

Here, we correct the attenuated estimate using the estimated trait repeatability value (see Table 1; Main Text).

$$\gamma_{11} = \frac{\gamma^*_{11}}{R_t}$$

```r
dt <- gather(d, obs, t, obs.t1:obs.t3, factor_key = TRUE)
dt <- dt[, c("ID", "t")]

modR2 <- lmer(t ~ 1 + (1 | ID), data = dt)
VI2 <- VarCorr(modR2)$ID[1, 1]
VR2 <- attr(VarCorr(modR2), "sc")^2
```

```
Rt2 <- VI2/(VI2 + VR2)
coef(mod1)[3]/Rt2
```

```
##   obs.z12
## -0.201364
```

Quadratic selection gradient using a mean of three values per individual

Here, we estimate the gradient based on the individual-mean trait values calculated from the 3 observations per individual.

```
mod.mean <- lm(rw ~ mean.z + mean.z2, data = d)
coef(mod.mean)[3]
```

```
##    mean.z2
## -0.1647601
```

Those attenuated estimates are corrected here using the formula in Table 1 (Main Text).

$$\gamma_{11} = \frac{\gamma^*_{11}}{R_t}$$

```
Rtm2 <- VI2/(VI2 + (VR2/3))
coef(mod.mean)[3]/Rtm2
```

```
##   mean.z2
## -0.288694
```

$$\gamma_{11} = \frac{\gamma^*_{11}}{\frac{V_{i_t}}{V_{i_t}+\frac{V_{e_t}}{n}}}$$

Multivariate model

Here, we extend our worked example to estimate nonlinear selection gradients.This involves convert the estimated variances and covariances into partial regression coefficients, which are then standardized using among-individual variance in the trait and the population-mean fitness.

$$\gamma_{11} = 2\frac{C'_{i_{t2,w}}}{V_{i_{t2}}}\frac{\sqrt{V_{i_{t2}}}}{\beta_{0_w}}$$

$C'_{i_{t2,w}}$ is the partial covariance between the squared trait and fitness, representing the covariance independent from the linear relation between the trait and fitness. This parameter can be calculated as

$$A^{-1}B$$

Where A^-1 is the inverse of the covariance between predictors

$$A^{-1} = \begin{bmatrix} V_{i_t} & C_{i_{t,t2}} \\ C_{i_{t2,t}} & V_{i_{t2}} \end{bmatrix}^{-1}$$

and B is the covariance between the trait, its square, and fitness.

$$B = \begin{bmatrix} C_{i_{t,w}} \\ C_{i_{t2,w}} \end{bmatrix}$$

The second element of this matrix multiplication is then used to estimate the partial regression coefficient for the squared trait b_2.

```
dw <- d[, c("ID", "w")]
dw$measure <- "w"
dw$type <- "s"
dw$time <- NA
colnames(dw)[2] <- "t"

dt <- gather(d, obs, t, obs.t1:obs.t3, factor_key = TRUE)
dt <- dt[, c("ID", "t")]
dt$measure <- "t1"
dt$type <- "r"
dt$time <- rep(1:3, each = n.ind)

dt2 <- gather(d, obs, t, obs.t12:obs.t32, factor_key = TRUE)
dt2 <- dt2[, c("ID", "t")]
dt2$measure <- "t2"
dt2$type <- "r"
dt2$time <- rep(1:3, each = n.ind)

d3 <- rbind(dw, dt, dt2)
d3$measure <- as.factor(d3$measure)
d3$type <- as.factor(d3$type)


prior0 <- list(R = list(R1 = list(V = diag(3), nu = 3, covu = TRUE), R2 = list(V = diag(2),
    nu = 0.002)))
modbi2 <- MCMCglmm(t ~ measure - 1, random = ~us(at.level(type, "r"):measure):ID,
    rcov = ~us(at.level(type, "s")):ID + idh(at.level(type, "r"):measure):ID:time,
    data = d3, prior = prior0, nitt = 55000, burnin = 5000, thin = 50, verbose = FALSE)
```

```
## Warning in buildZ(rmodel.terms[r], data = data): missing values in random
## predictors
```

```
m <- matrix(posterior.mode(modbi2$VCV)[1:9], 3, 3)
m1 <- m[1:2, 1:2]
m2 <- c(m[1, 3], m[2, 3])
cr <- (solve(m1) %*% m2) * sqrt(diag(m1))/posterior.mode(modbi2$Sol[, 3])
cr[2]
```

```
## [1] -0.2583656
```

Errors-in-variables model

Unlike in our linear example, here we directly model relative fitness, for reasons detailed in the Main Text. We further Equation S3.10 (Supplementary Text S3) to estimate the variance in the quadratic term of the trait (t^2) based on the variance in the trait (t).

$$V_{i_{t2}} = 2(V_i^2 + 2\beta_{0_t}^2 V_i)$$

The calculation of the quadratic selection gradient acknowledges that we fitted relative (rather than absolute) fitness as the response:

$$\gamma_{11} = 2b_2\sqrt{V_{i_{t2}}}$$

Notably, standard deviations are denoted with the greek letter sigma (σ), and the variance as sigma squared (σ^2).

```
"generated quantities {
  real beta_1;
  real gamma;
  real sigma2_i;
  real sigma2_i2;

  sigma2_i = sigma_i^2;
  sigma2_i2 = 2*(sigma2_i^2 + 2*beta_0t^2*sigma2_i);
  beta_1 = b_1*sigma_i;
  gamma = 2*b_2*sqrt(sigma2_i2);
}"
```

We then combine all components to run the errors-in-variables model.

```
   write(
   "data {
      // Define variables in data
      // Number of observations (an integer)
     int<lower=0> N;
     // Number of clusters (an integer)
     int<lower=0> nind;
     // Cluster IDs
     int<lower=0> ID[N];
     // Continuous outcome
      real t[N];
      real rw[nind];
    }

    parameters {
      // Define parameters to estimate
      // Population intercept (a real number)
     real beta_0t;
     real beta_0w;
     real b_1;
     real b_2;

      // Random effects
      real I[nind];
      real<lower=0> sigma_i;
      real<lower=0> sigma_e;
      real<lower=0> sigma_w;
    }

   model {
      real mu_t[N];
      real mu_w[nind];

      for (j in 1:nind) {
        mu_t[j]  =  beta_0t + I[j];
        mu_w[j] = beta_0w + b_1*I[j] + b_2*I[j]^2;
      }
```

```
    sigma_i ~ uniform(0, 10);
    sigma_e ~ uniform(0, 10);
    I ~ normal(0, sigma_i);

    // Likelihood part of Bayesian inference
    for (i in 1:N) {
      t[i] ~ normal(mu_t[ID[i]], sigma_e);
    }

  for (i in 1:nind) {
      rw[i] ~ normal(mu_w[i], sigma_w);
    }
  }

  generated quantities {
   real beta_1;
   real gamma;
   real sigma2_i;
   real sigma2_i2;

   sigma2_i = sigma_i^2;
   sigma2_i2 = 2*(sigma2_i^2 + 2*beta_0t^2*sigma2_i);
   beta_1 = b_1*sigma_i;
   gamma = 2*b_2*sqrt(sigma2_i2);
  }"

  , "ME2.stan")

  dt$ID<-as.numeric(dt$ID)
  nind<-max(dt$ID)

  stan_data <- list(t = dt$t,  nind = nind,
                    N=nrow(dt), ID=dt$ID, rw=d$w/mean(d$w))

  #Initial values
  inits <- function() list(sigma_i = runif(1, 0, 10),
                           sigma_e = runif(1, 0, 10),
                           sigma_w = runif(1, 0, 10))

  #Parameters monitored
  params <- c("sigma_i", "sigma_i2",  "sigma_e", "b_1", "b_2","beta_1", "gamma")
  ## MCMC settingsl
  ni <- 3000
  nt <- 2
  nb <- 1000
  nc <- 1

  ## Call Stan from R
  md <- stan("ME2.stan", data = stan_data, init = inits, pars = params,
  chains = nc, iter = ni, warmup = nb, thin = nt, cores=1, refresh=0)
```

```
## Error in new_CppObject_xp(fields$.module, fields$.pointer, ...) :
##   Exception: mismatch in dimension declared and found in context; processing stage=data initializati
```

```
## failed to create the sampler; sampling not done
```

```
summary(md)$summary[7, 1]
```

```
## Stan model 'ME2' does not contain samples.
```

```
## NULL
```

Correlational selection scenario

Next, extend our simulations to perform correlational selection analyses.

```
# New simulation for correlational selection For simplicity we assume that
# the effect of the two traits on fitness is the same.

n.ind <- 800

V_m.error <- 7
V.trait <- 3
mean.trait <- 0

B1_z <- 1
B2_z2 <- -0.3
B1_z1z2 <- 0.3

B1_t <- B1_z/sqrt(V.trait) * 2
B2_t2 <- B2_z2/sqrt(2 * (V.trait^2 + 2 * V.trait * mean.trait^2))
B1_t1t2 <- B1_z1z2/sqrt((V.trait^2 + 2 * V.trait * mean.trait^2)) * 2

m <- matrix(NA, 2, 2)
m[1, 1] <- m[2, 2] <- V.trait
m[1, 2] <- m[2, 1] <- 0
ts <- mvrnorm(n.ind, c(mean.trait, mean.trait), m)

d <- data.frame(ID = 1:n.ind, t = ts[, 1], x = ts[, 2])
d$t2 <- d$t^2
d$x2 <- d$x^2

d$z <- scale(d$t)
d$z2 <- scale(d$t2)

d$y <- scale(d$x)
d$y2 <- scale(d$x2)

d$w <- 2 + d$t * B1_t + d$t2 * B2_t2 + d$x * B1_t + d$x2 * B2_t2 + B1_t1t2 *
    d$t * d$x + rnorm(n.ind, 0, sqrt(1)) - mean(d$t * B1_t + d$t2 * B2_t2 +
    d$x * B1_t + d$x2 * B2_t2 + B1_t1t2 * d$t * d$x + rnorm(n.ind, 0, sqrt(1)))

d$rw <- d$w/mean(d$w)

d$obs.t1 <- rnorm(n.ind, d$t, sqrt(V_m.error))
d$obs.t2 <- rnorm(n.ind, d$t, sqrt(V_m.error))
d$obs.t3 <- rnorm(n.ind, d$t, sqrt(V_m.error))

d$obs.t12 <- d$obs.t1^2
d$obs.t22 <- d$obs.t2^2
```

```
d$obs.t32 <- d$obs.t3^2

d$obs.x1 <- rnorm(n.ind, d$x, sqrt(V_m.error))
d$obs.x2 <- rnorm(n.ind, d$x, sqrt(V_m.error))
d$obs.x3 <- rnorm(n.ind, d$x, sqrt(V_m.error))

d$obs.x12 <- d$obs.x1^2
d$obs.x22 <- d$obs.x2^2
d$obs.x32 <- d$obs.x3^2

m <- mean(c(d$obs.t1, d$obs.t2, d$obs.t3))
sd <- sd(c(d$obs.t1, d$obs.t2, d$obs.t3))

m2 <- mean(c(d$obs.t12, d$obs.t22, d$obs.t32))
sd2 <- sd(c(d$obs.t12, d$obs.t22, d$obs.t32))

mx <- mean(c(d$obs.x1, d$obs.x2, d$obs.x3))
sdx <- sd(c(d$obs.x1, d$obs.x2, d$obs.x3))

mx2 <- mean(c(d$obs.x12, d$obs.x22, d$obs.x32))
sdx2 <- sd(c(d$obs.x12, d$obs.x22, d$obs.x32))

d$obs.z1 <- (d$obs.t1 - m)/sd
d$obs.z2 <- (d$obs.t2 - m)/sd
d$obs.z3 <- (d$obs.t3 - m)/sd

d$obs.z12 <- (d$obs.t12 - m2)/sd2
d$obs.z22 <- (d$obs.t22 - m2)/sd2
d$obs.z32 <- (d$obs.t32 - m2)/sd2

d$obs.y1 <- (d$obs.x1 - mx)/sdx
d$obs.y2 <- (d$obs.x2 - mx)/sdx
d$obs.y3 <- (d$obs.x3 - mx)/sdx

d$obs.y12 <- (d$obs.x12 - mx2)/sdx2
d$obs.y22 <- (d$obs.x22 - mx2)/sdx2
d$obs.y32 <- (d$obs.x32 - mx2)/sdx2

d$mean.t <- apply(d[, c("obs.t1", "obs.t2", "obs.t3")], 1, mean)
d$mean.t2 <- d$mean.t^2
d$mean.z <- scale(d$mean.t)
d$mean.z2 <- scale(d$mean.t2)

d$mean.x <- apply(d[, c("obs.x1", "obs.x2", "obs.x3")], 1, mean)
d$mean.x2 <- d$mean.x^2
d$mean.y <- scale(d$mean.x)
d$mean.y2 <- scale(d$mean.x2)
```

Expected correlational selection gradient

Here we now have t and its standarized value z and trait x and its standardized value y as well as their squared terms.

```
mod <- lm(rw ~ z + z2 + y + y2 + y * z, data = d)
coef(mod)[6]
```

```
##       z:y
## 0.2574669
```

Correlational selection analyses with one value per individual

As above, we can estimate correlational selection using one trait value per indivdiuals, which should result in an attenuated estimated because the geometric repeatability of the two traits is below the value one.

```
mod1 <- lm(rw ~ obs.z1 + obs.z12 + obs.y1 + obs.y12 + obs.y1 * obs.z1, data = d)
coef(mod1)[6]
```

```
## obs.z1:obs.y1
##    0.06981752
```

Here, we can correct the attenuated estimate using the formula printed in Table 1 (Main Text).

```
dx <- gather(d, obs, x, obs.x1:obs.x3, factor_key = TRUE)
modx <- lmer(x ~ 1 + (1 | ID), data = dx)
VIx <- VarCorr(modx)$ID[1, 1]
VRx <- attr(VarCorr(modx), "sc")^2

dt <- gather(d, obs, t, obs.t1:obs.t3, factor_key = TRUE)
modt <- lmer(t ~ 1 + (1 | ID), data = dt)
VIt <- VarCorr(modt)$ID[1, 1]
VRt <- attr(VarCorr(modt), "sc")^2

Rtx <- (VIt * VIx)/((VIt + VRt) * (VIx + VRx))
coef(mod1)[6]/Rtx
```

```
## obs.z1:obs.y1
##     0.8559126
```

Correlational selection analyses using a mean of three values per individual

We assess the estimates of correlational selection derived from fitting individual-mean trait values.

```
mod.mean <- lm(rw ~ mean.z + mean.z2 + mean.y + mean.y2 + mean.z * mean.y, data = d)
coef(mod.mean)[6]
```

```
## mean.z:mean.y
##     0.1603608
```

Here, we can correct the attenuated estimate using the formula printed in Table 1 (Main Text).

```
Rtxm <- (VIt * VIx)/((VIt + VRt/3) * (VIx + VRx/3))
coef(mod.mean)[6]/Rtxm
```

```
## mean.z:mean.y
##     0.5396775
```

Multivariate mixed-effects model

Here we extend the trivariate mixed-effect model approach to estimate correlational selection gradients. Constraints imposed on the residual variance-covariance matrix, and details on the usage of partial regression coefficients or provided in the quadratic example above (see also Supplementary Text S6).

```
dw <- d[, c("ID", "w")]
dw$measure <- "w"
dw$type <- "s"
dw$time <- NA
colnames(dw)[2] <- "t"

dt <- gather(d, obs, t, obs.t1:obs.t3, factor_key = TRUE)
dt <- dt[, c("ID", "t")]
dt$measure <- "t1"
dt$type <- "r"
dt$time <- rep(1:3, each = 800)

dt2 <- gather(d, obs, t, obs.t12:obs.t32, factor_key = TRUE)
dt2 <- dt2[, c("ID", "t")]
dt2$measure <- "t2"
dt2$type <- "r"
dt2$time <- rep(1:3, each = 800)

dx <- gather(d, obs, t, obs.x1:obs.x3, factor_key = TRUE)
dx <- dx[, c("ID", "t")]
dx$measure <- "u1"
dx$type <- "r"
dx$time <- rep(1:3, each = 800)

dx2 <- gather(d, obs, t, obs.x12:obs.x32, factor_key = TRUE)
dx2 <- dx2[, c("ID", "t")]
dx2$measure <- "u2"
dx2$type <- "r"
dx2$time <- rep(1:3, each = 800)

dxt <- dx
dxt$t <- dx$t * dt$t
dxt$measure <- "ut"
dxt$type <- "r"


d3 <- rbind(dw, dt, dt2, dx, dx2, dxt)
d3$measure <- as.factor(d3$measure)
d3$type <- as.factor(d3$type)


prior0 <- list(R = list(R1 = list(V = diag(6), nu = 5.0002, covu = TRUE), R2 = list(V = diag(5),
    nu = 2e-04)))
modbi2 <- MCMCglmm(t ~ measure - 1, random = ~us(at.level(type, "r"):measure):ID,
    rcov = ~us(at.level(type, "s")):ID + idh(at.level(type, "r"):measure):ID:time,
    data = d3, prior = prior0, nitt = 105000, burnin = 5000, thin = 100, verbose = FALSE)
```

```
## Warning in buildZ(rmodel.terms[r], data = data): missing values in random
## predictors
```

```
m <- matrix(posterior.mode(modbi2$VCV)[1:36], 6, 6)
m1 <- m[1:5, 1:5]
m2 <- c(m[1, 6], m[2, 6], m[3, 6], m[4, 6], m[5, 6])
cr <- (solve(m1) %*% m2) * sqrt(diag(m1))/posterior.mode(modbi2$Sol[, 6])
```

```
cr[5]
```

```
## [1] -0.1381048
```

Errors-in-variables model

Finally, the updated errors-in-variables model, as above, models relative fitness and uses the formulas provided in Supplementary Text (S4) to estimate the variance of the product of two traits.

```
write(

   "data {
   // Define variables in data
   // Number of observations (an integer)
  int<lower=0> N;
  // Number of clusters (an integer)
  int<lower=0> nind;
  // Cluster IDs
  int<lower=0> ID[N];
  // Continuous outcome
   real z[N];
   real w[nind];
   real x[N];
 }

 parameters {
   // Define parameters to estimate
   // Population intercept (a real number)
  real beta_0z;
  real beta_0x;
  real beta_0w;
  real beta_Nz;
  real beta_Nz2;
  real beta_Nx;
  real beta_Nx2;
  real beta_Nzx;

   // Random effects
  real Iz[nind];
  real Ix[nind];

   // Standard deviation of random effects
   real<lower=0> sigma_Iz;
   real<lower=0> sigma_Ix;

   // Population standard deviation
   real<lower=0> sigma_Ex;
   real<lower=0> sigma_Ez;
   real<lower=0> sigma_w;
 }

model {
   real zI[nind];
   real xI[nind];
   real mu_z[N];
```

```
   real mu_x[N];
   real mu_w[nind];

   for (j in 1:nind) {
     zI[j]  = beta_0z + Iz[j];
     xI[j]  = beta_0x + Ix[j];
     mu_w[j] = beta_0w + beta_Nz*Iz[j] + beta_Nz2*Iz[j]^2 + beta_Nx*Ix[j]
      + beta_Nx2*Ix[j]^2 + beta_Nzx*Iz[j]*Ix[j];
   }

   // Prior part of Bayesian inference
   // Flat prior for mu (no need to specify if non-informative)
   sigma_Iz ~ uniform(0, 10);
   sigma_Ez ~ uniform(0, 10);
   sigma_Ix ~ uniform(0, 10);
   sigma_Ex ~ uniform(0, 10);

   // Random effects distribution
      Iz ~ normal(0, sigma_Iz);
   Ix ~ normal(0, sigma_Ix);

   // Likelihood part of Bayesian inference
   for (i in 1:N) {
     z[i] ~ normal(zI[ID[i]], sigma_Ez);
     x[i] ~ normal(xI[ID[i]], sigma_Ex);
   }

 for (i in 1:nind) {
     w[i] ~ normal(mu_w[i], sigma_w);
   }
 }

 generated quantities {
  real beta_Nzz;
  real beta_Nxz;
  real beta_Nzxz;
  real sigma2_Iz;
  real sigma2_Ix;
  real sigma2_Izx;

  sigma2_Iz = sigma_Iz^2;
  sigma2_Ix = sigma_Ix^2;
  sigma2_Izx = sigma2_Iz*sigma2_Ix;

  beta_Nzz = beta_Nz*sigma_Iz;
  beta_Nxz = beta_Nx*sigma_Ix;
  beta_Nzxz = beta_Nzx*sqrt(sigma2_Izx);
}

" , "ME3.stan")

dt$ID<-as.numeric(dt$ID)
nind<-max(dt$ID)
```

```
stan_data <- list(z = dt$t,  x=dx$x,
                  nind = nind, ID=dt$ID,
                  w=dw$w/mean(dw$w),
                  N=nrow(dt))

#Parameters monitored
params <- c("sigma_Iz", "sigma_Ix", "beta_Nzz", "beta_Nxz" ,"beta_Nzxz")
## MCMC settingsl
ni <- 3000
nt <- 2
nb <- 1000
nc <- 3

## Call Stan from R
md3 <- stan("ME3.stan", data = stan_data, pars = params,
                    chains = nc, iter = ni, warmup = nb, thin = nt,
                        cores=3)
summary(md3)$summary[5,1]
```

```
## [1] 0.2754118
```

Note that the model printed above assumes that the two traits are uncorrelated within- and among-individuals. To extend the model to apply to correlated traits, would we need to estimate the among-individual correlation to be able to estimate the variance of the product of two traits (see Equation S4.5 & derived quantities part of the model code).

```
write(

   "data {
   // Define variables in data
   // Number of observations (an integer)
  int<lower=0> N;
  int<lower=1>    K;
  // Number of clusters (an integer)
  int<lower=0> nind;
  // Cluster IDs
  int<lower=0> ID[N];
  // Continuous outcome
    vector[K]  t[N];
    vector[nind] w ;

   }

parameters {
   // Define parameters to estimate
   // Population intercept (a real number)

   vector[2] b_0t;
   vector[5] b_t;
   real b_0w;
   vector<lower=0, upper = 1>[K] R;
   cholesky_factor_corr[K] L_u;
   cholesky_factor_corr[K] L_e;
   matrix[2,nind] I_t;
```

```
   vector<lower=0>[2] sigma ;
   real<lower=0> sigma_w;
}

transformed parameters {
   vector<lower=0>[2] sigma_I ;
   vector<lower=0>[2] sigma_E ;
   matrix[2,nind] I;

   for(k in 1:K){
   sigma_I[k] = sigma[k] * R[k];
   sigma_E[k] = sigma[k] * (1-R[k]);
   }

   I = diag_pre_multiply(sigma_I, L_u) * I_t;
}

model {
   vector[K] mu_ti[nind];
   matrix[K, K] Sigma_E;
   vector[K]     mu_t[N];
   vector[nind]  mu_w;

   Sigma_E=diag_pre_multiply(sigma_E, L_e);
   L_e ~ lkj_corr_cholesky(4); // LKJ prior for the  correlation matrix
   L_u ~ lkj_corr_cholesky(4); // LKJ prior for the  correlation matrix
   to_vector(I_t) ~ normal(0,1);
   to_vector(b_t) ~ normal(0,1);

  R ~ beta(2, 4);
  sigma ~ normal(0,1);

   for (j in 1:nind)
     mu_ti[j]  = b_0t + I[,j];

   for (i in 1:N)
     mu_t[i]=mu_ti[ID[i]];

    t ~ multi_normal_cholesky(mu_t, Sigma_E);

   for (j in 1:nind)
      mu_w[j] = b_0w + b_t[1]*I[1,j] + b_t[2]*I[1,j]^2 + b_t[3]*I[2,j]
      + b_t[4]*I[2,j]^2 + b_t[5]*I[1,j]*I[2,j];

   w ~  normal(mu_w, sigma_w);
}

generated quantities {

    real sigma2_it1;
    real sigma2_it2;
    matrix[2, 2]  Omega_I;
    real cov_I;
```

```
    real sigma2_it12;
    real sigma2_it22;
    real sigma2_it21;

    real beta_1t1;
    real beta_1t2;
    real gamma_11;
    real gamma_22;
    real gamma_12;

    sigma2_it1 = sigma_I[1]^2;
    sigma2_it2 = sigma_I[2]^2;
    Omega_I = L_u * L_u';
    cov_I = Omega_I[1,2]*sqrt(sigma2_it2*sigma2_it1);

    sigma2_it12 = 2*(sigma2_it1^2 + 2*b_0t[1]^2*sigma2_it1);
    sigma2_it22 = 2*(sigma2_it2^2 + 2*b_0t[2]^2*sigma2_it2);

   sigma2_it21 = cov_I^2 + 2*b_0t[1]*b_0t[2]*sqrt(sigma2_it1*sigma2_it2)
   + (b_0t[1]^2 + sigma2_it1)*(b_0t[2]^2 + sigma2_it2)
   -(cov_I+b_0t[1]*b_0t[2])^2 ;

   beta_1t1 =  b_t[1]*sqrt(sigma2_it1);
   beta_1t2 =  b_t[3]*sqrt(sigma2_it2);

   gamma_11 = b_t[2]*sqrt(sigma2_it12)*2;
   gamma_22 = b_t[4]*sqrt(sigma2_it22)*2;

   gamma_12 = b_t[5]*sqrt(sigma2_it21);
}

" , "/home/yi/Dropbox/SelectionNiels/ME4.stan")

stan_data <- list(t = cbind(dt$t, dx$x), f= matrix(1,nind,2),
                  nind = nind,
                  w=dw$w/mean(dw$w),
                  N=nrow(dt), ID=dt$ID, K=2)

#Parameters monitored
params <- c("sigma2_it1","sigma2_it2", "cov_I", "sigma2_it21",
                  "beta_1t1", "beta_1t2","gamma_11", "gamma_22", "gamma_12")

## MCMC settingsl
ni <- 3000
nt <- 2
nb <- 1000
nc <- 2


## Call Stan from R
md3b <- stan("/home/yi/Dropbox/SelectionNiels/ME4.stan",
                        data = stan_data, pars = params,
                        chains = nc, iter = ni, warmup = nb, thin = nt,
```

```
                          cores=2)
summary(md3b)$summary[9, 1]
```

```
## [1] 0.2692142
```