Main notebook

2023-06-23

R code acknowledgements

The application dataset and a large portion of data preprocessing is provided by Jane Reid. The re-implementation into INLA is also largely based on the work from Stefanie Muff.

Data loading

We start by installing and importing required packages.

```
req.packages <- c("BiocManager", "ggplot2", "latex2exp", "nadiv",
                  "QGglmm", "cowplot", "reshape2", "showtext")
to.install <- req.packages[
  is.na(match(req.packages, installed.packages()[,1]))
  ]
if(length(to.install) > 0L){
  install.packages(to.install)
}
for(pack in req.packages){
  suppressPackageStartupMessages(
    library(pack,character.only = TRUE, quietly = TRUE))
}

if (!require("GeneticsPed", quietly = TRUE)) {
  BiocManager::install("GeneticsPed")
}
if (!require("MCMCglmm", quietly = TRUE)) {
  # rbv patch
  install.packages("../MCMCglmm-rbv-patch.tar.gz")
}
if (!require("INLA", quietly = TRUE)) {
  install.packages(
    "INLA", repos=c(getOption("repos"),
                    INLA="https://inla.r-inla-download.org/R/stable"),
    dep=TRUE)

}
library(MCMCglmm)
library(MASS)
library(bdsmatrix)
library(INLA)

library(GeneticsPed) #

# Plotting libraries and settings
```

```
library(grid)
texfont <- "CMU Serif"
showtext_auto()
font.paths(file.path(
  Sys.getenv("LOCALAPPDATA"), "Microsoft",
  "Windows", "Fonts"
))
font_add(texfont, regular = "cmunrm.ttf")
theme_set(theme_bw() + theme(text = element_text(family = texfont,
                                                 size = 20)))

# Dataset import
qg.data.gg.inds <- read.table("../data/qg.data.gg.inds.steffi.txt",
  header = TRUE
)
d.ped <- read.table("../data/ped.prune.inds.steffi.txt",
  header = TRUE
)
d.Q <- read.table("../data/Q.data.steffi.txt", header = TRUE)

qg.data.gg.inds$natalyr.id <- qg.data.gg.inds$natalyr.no
```

Some global settings:

```
SAVE.PLOT <- TRUE
n.samples <- 10000
# iid N(0,1) noise in songsparrow formula:
FORMULA.EXTRA.IID.NOISE <- FALSE
```

Below we do a couple more preprocessing steps, namely:

- build a pedigree structure from the denormalized table with `prepPed`.
- assign new IDs to each individual starting at 1.
- keep a mapping record between the original IDs (ninecode) and the new 1-indexed IDs.
- use this mapping to transform the IDs for each individual's Dam and Sire to the same.
- compute the inverse of the relatedness matrix.

```
# Scale the continuous variances for stability
qg.data.gg.inds$f.coef.sc <- scale(qg.data.gg.inds$f.coef,
                                   scale = FALSE)
qg.data.gg.inds$g1.sc <- scale(qg.data.gg.inds$g1,
                               scale = FALSE)
qg.data.gg.inds$natalyr.no.sc <- scale(qg.data.gg.inds$natalyr.no,
                                       scale = FALSE)
qg.data.gg.inds$brood.date.sc <- scale(qg.data.gg.inds$brood.date,
                                       scale = FALSE)

# Binarize `sex` covariate
qg.data.gg.inds$sex <- qg.data.gg.inds$sex.use.x1 - 1
```

Deriving A

For INLA we need IDs that run from 1 to the number of individuals

```
d.ped <- nadiv::prepPed(d.ped)
d.ped$id <- seq_len(nrow(d.ped))
```

```r
# Maps to keep track of the Ninecode to ID relations
d.map <- d.ped[, c("ninecode", "id")]
d.map$g1 <- d.Q[match(d.map$ninecode, d.Q$ninecode), "g1"]
d.map$foc0 <- d.Q[match(d.map$ninecode, d.Q$ninecode), "foc0"]

# Give mother and father the id
d.ped$mother.id <- d.map[match(d.ped$gendam, d.map$ninecode), "id"]
d.ped$father.id <- d.map[match(d.ped$gensire, d.map$ninecode), "id"]

# A can finally be constructed using `nadiv`
Cmatrix <- nadiv::makeAinv(
  d.ped[, c("id", "mother.id", "father.id")])$Ainv

# Stores ID twice (to allow for extra IID random effect)

qg.data.gg.inds$id <- d.map[
  match(qg.data.gg.inds$ninecode, d.map$ninecode),
  "id"
]
qg.data.gg.inds$u <- seq_len(nrow(qg.data.gg.inds))
```

INLA

The general INLA formula is provided below, where `f()` encode the random effect:

```r
formula.inla.scaled <- surv.ind.to.ad ~ f.coef.sc + g1.sc +
  natalyr.no.sc + brood.date.sc + sex +
  f(nestrec, model = "iid", hyper = list(
    prec = list(
      initial = log(1 / 0.05),
      prior = "pc.prec",
      param = c(1, 0.05)
    ) # PC priors
  )) +
  f(natalyr.id, model = "iid", hyper = list(
    prec = list(
      initial = log(1 / 0.25),
      prior = "pc.prec",
      param = c(1, 0.05)
    ) # PC priors
  )) +
  f(id,
    model = "generic0", # Here we need to specify the covariance matrix
    Cmatrix = Cmatrix, #    via the inverse (Cmatrix)
    constr = FALSE,
    hyper = list(
      prec = list(initial = log(1 / 10), prior = "pc.prec",
                  param = c(1, 0.05))
    ) # PC priors
  )
if (FORMULA.EXTRA.IID.NOISE) {
  formula.inla.scaled <- update(
    formula.inla.scaled,
```

```
    ~ . + f(u,
      model = "iid", constr = TRUE,
      hyper = list(prec = list(
        initial = log(1),
        fixed = TRUE
      ))
    )
  )
}
```

Now we call INLA models. Note that we pass some control arguments to the function call. We compute DIC (Deviance information criterion) for all models with `dic` flag in `control.compute`. For the binomial models, we want to be able to use `QGglmm` and average over all fixed effects. This is done by supplying the *latent marginal predicted values*, which are not computed unless you pass the `return.marginals.predictor` flag set to true. We also want to set the `CPO` flag to true in the Gaussian model, so we can look a bit at its "residuals" (PIT values). Lastly, the `control.family` argument is used to pass the link functions for binomial models.

```
fit.inla.probit <- inla(
  formula = formula.inla.scaled, family = "binomial",
  data = qg.data.gg.inds,
  control.compute = list(dic = TRUE,
                         return.marginals.predictor = TRUE),
  control.family = list(link = "probit"),
)


fit.inla.gaussian <- inla(
  formula = formula.inla.scaled, family = "gaussian",
  data = qg.data.gg.inds,
  control.compute = list(dic = TRUE, cpo = TRUE)
)

data.frame(
  Gaussian = fit.inla.gaussian$dic$dic,
  Probit = fit.inla.probit$dic$dic,
  row.names = "Deviance Information Criteria"
)
```

Latent heritability

Below is a function to get h^2_{obs}, h^2_{lat} and h^2_{Φ} based on a fitted model.

```
get.h2 <- function(inla.fit, n, use.scale = FALSE, model = NA,
                   include.fixed = FALSE) {
  #' Get heritability
  #'
  #' Get n samples of heritability (h^2) from INLA object
  #' @param inla.fit fitted model
  #' @param n number of samples
  #' @param use.scale flag for adding link variance to denominator
  #' @param model string representation of model type
  #' @param include.fixed flag for including fixed effects variance
  #' @return n-sized vector of heritability samples
```

```
  samples <- inla.hyperpar.sample(n = n, inla.fit)
  denominator <- 0
  for (cname in colnames(samples)) {
    denominator <- denominator + 1 / samples[, cname]
  }
  if (include.fixed) {
    # Grab variance from summary object in INLA fit
    denominator <- denominator + sum(inla.fit$summary.fixed[, "sd"]^2)
  }

  if (use.scale) {
    scales.dictionary <- list(
      binom1.probit = 1, binom1.logit = pi^2 / 3,
      round = 0.25
    )
    scale.param <- get(model, scales.dictionary)
    denominator <- denominator + scale.param
  }

  h2.inla <- (1 / samples[, "Precision for id"]) / denominator
  return(h2.inla)
}

threshold.scaling.param <- function(p) {
  # h^2_liab = threshold.scaling.param * h^2_obs
  p * (1 - p) / (dnorm(qnorm(p)))^2
}
```

Simulation data

We implement a general simulation method that, based on the passed arguments, generates simulation data and fits animal models onto the dichotomized response.

```
simulated.heritability <- function(NeNc = 0.5, idgen = 100, nGen = 9,
                                   sigmaA = 0.8, linear.predictor = NA,
                                   simulated.formula = NA,
                                   dichotomize = "round",
                                   pc.prior = NA, probit.model = FALSE,
                                   simulated.formula.probit = NA,
                                   DIC = FALSE) {
  #' Simulate and fit animal model
  #'
  #' Generate pedigree, fit Gaussian (INLA) model and provide
  #' heritability estimate.
  #' @param NeNc Effective/Census population mean, used to determine
  #' number of fathers and mothers per generation,
  #' @param idgen Number of individuals per generation in pedigree
  #' @param nGen Number of generations in pedigree
  #' @param sigmaA:  Additive genetic variance
  #' @param linear.predictor Callable function of two parameters, the
  #' first 'u' (breeding values), the second for the data
  #' @param simulated.formula Formula expression using the response
  #' name `simulated.response`, param `id` and `Cmatrix`, both of
```

```r
#' which are defined locally in this method.
#' @param dichotomize Dichotomization method (round, binomial, etc)
#' @param pc.prior (optional) parameters for PC prior
#' @param probit.model (optional) flag to fit binomial probit model
#' in addition to the Gaussian model.
#' @param simulated.formula.probit (opt) Formula for probit model
#' @param DIC (optional) Flag for computing DIC for the models
#'
#' @return A list with the following items:
#' heritability: Posterior latent heritability samples
#' summary: List of mean, standard deviation and quantiles of h^2
#' p: Portion of `TRUE` observations in simulated response
#' simulated.response:  The observed values in simulation dataset
#' fit: Gaussian fitted model
#' fit.probit: Probit fitted model, if `probit.model=F`, is `NULL`.
ped0 <- generatePedigree(
  nId = idgen, nGeneration = nGen,
  nFather = idgen * NeNc, nMother = idgen * NeNc
)
# Set correct format for pedigree
pedigree <- ped0[, c(1, 3, 2, 5)]
names(pedigree) <- c("id", "dam", "sire", "sex")

# Generate random breeding values
# The following will CRASH if you don't
# use the patched MCMCglmm package!
u <- rbv(pedigree[, c(1, 2, 3)], sigmaA)

simulated.d.ped <- nadiv::prepPed(pedigree, gender = "sex")
# Binarize from (1,2) to (0,1)
simulated.d.ped$sex <- simulated.d.ped$sex - 1
simulated.Cmatrix <- nadiv::makeAinv(pedigree[, c(1, 2, 3)])$Ainv
# Make index to allow iid noise random effect
simulated.d.ped$ind <- seq_len(nrow(simulated.d.ped))

# Generating "true" y_i

if (dichotomize == "binom1.logit") {
  simulated.response <- rbinom(length(u),
    size = 1,
    prob = pnorm(linear.predictor(u, simulated.d.ped))
  )
} else if (dichotomize == "round") {
  # This assumes mean of \eta_i is 0
  simulated.response <- ifelse(
    linear.predictor(u, simulated.d.ped) <= 0, 0, 1
  )
} else if (dichotomize == "round_balanced") {
  # Get balanced residuals for unbalanced linear predictor
  cutoff <- mean(linear.predictor(u, simulated.d.ped))
  simulated.response <- ifelse(
    linear.predictor(u, simulated.d.ped) <= cutoff, 0, 1
  )
```

```
} else if (is.numeric(dichotomize)) {
  stopifnot(dichotomize >= 0 & dichotomize <= 1)
  eta_values <- linear.predictor(u, simulated.d.ped)
  cutoff <- quantile(eta_values, 1 - dichotomize)
  # e.g. dichotomize=0.1, p should be about 0.1
  simulated.response <- ifelse(eta_values <= cutoff, 0, 1)
} else {
  stop(paste0(
    "Unknown dichotomization method '", dichotomize, "'. ",
    "Consider using 'binom1.logit' or 'round'."
  ))
}
p <- mean(simulated.response) # portion of true responses


# Model fitting LMM for binary trait
# First reload formula environment to access local variables
environment(simulated.formula) <- environment()
environment(simulated.formula.probit) <- environment()

simulated.fit.inla <- inla(
  formula = simulated.formula, family = "gaussian",
  data = simulated.d.ped, control.compute = list(dic = DIC)
)

# Checks for error status in INLA fit,
if (simulated.fit.inla$mode$mode.status != 0) {
  cat("\n[WARNING], INLA status",
      simulated.fit.inla$mode$mode.status, "\n")
}
heritability <- get.h2(simulated.fit.inla, 10000)

# Also fit probit if specified
if (probit.model) {
  fit.probit <- inla(
    formula = simulated.formula.probit, family = "binomial",
    data = simulated.d.ped,
    control.compute = list(return.marginals.predictor = TRUE,
                           dic = DIC)
  )
} else {
  fit.probit <- NULL
}
list(
  heritability = heritability,
  summary = list(
    mean = mean(heritability),
    standard.deviation = sd(heritability),
    quantiles = quantile(heritability, probs = c(0.025, 0.5, 0.975))
  ),
  p = p,
  simulated.response = simulated.response,
  fit = simulated.fit.inla,
```

```
    fit.probit = fit.probit
  )
}
```

Now we try to run it through the model pipeline. Here we try $\eta_i = u_i + e_i$ (extra iid effect) and $y_i = u_i + \varepsilon_i$.

```
simulated.formula <- simulated.response ~  f(id,
  model = "generic0",
  Cmatrix = simulated.Cmatrix,
  constr = FALSE,
  hyper = list(
    prec = list(initial = log(1 / 10), prior = "pc.prec",
                param = c(1, 0.05))
  ))
simulated.formula.probit <- simulated.response ~  f(id,
  model = "generic0",
  Cmatrix = simulated.Cmatrix,
  constr = FALSE,
  hyper = list(
    prec = list(initial = log(1 / 10), prior = "pc.prec",
                param = c(1, 0.05))
  )) + f(ind, model="iid", hyper=list(prec=list(
    prior="pc.prec", param=c(1,0.05))))

get.simulated.threshold.value <- function(
    NeNc, idgen, nGen, sigmaA, linear.predictor, simulated.formula,
    Vp = NA) {
  #' Get h^2 statistics from simulation
  #'
  #' Makes a dataframe showing observation h^2 and
  #' liability h^2 with 95% confidence interval. Most parameters are
  #' for generating pedigree and not covered in this docstring.
  #' @param Vp Total phenotypic variance, used to get "true" h^2
  #' @return A dataframe with said statistics
  stopifnot("Vp required to get true h^2" = !is.na(Vp))
  sim.result <- simulated.heritability(
    NeNc, idgen, nGen, sigmaA,
    linear.predictor, simulated.formula
  )
  threshold.scaled.h2 <- threshold.scaling.param(sim.result$p) *
    sim.result$heritability
  simulation.h2.true <- sigmaA / (Vp)

  data.frame(
    Simulation = c(
      simulation.h2.true, mean(threshold.scaled.h2),
      paste0("(", paste(format(
        quantile(threshold.scaled.h2, probs = c(0.025, 0.975)),
        digits = 4
      ), collapse = ", "), ")"),
      mean(sim.result$summary$mean)
    ),
    row.names = c(
      "True h^2", "Estimated h^2_obs, mean",
```

```
      "95% Confidence interval", "Estimated latent mean"
    )
  )
}

get.simulated.threshold.value(
  NeNc = 0.5, idgen = 100, nGen = 9, sigmaA = 0.05,
  function(u, .) {
    u + rnorm(length(u))
  },
  simulated.formula,
  Vp = 0.05 + 1
)
```

Further, we quantitatively look into estimates for different σ_A^2:

Performance over varying V_A

```
plot.h2.deviation <- function(
    dichotomize = "round",
    title = "Simulation heritability",
    SAVE.PLOT = TRUE, plot.fn = NA, sigma.scale = "log",
    lin.pred = NULL,
    dynamic.priors = FALSE, simulated.formula = NULL, Ve = NULL,
    fixedeffects = FALSE) {
  #' Plot h^2 estimate, alongside true value, for a series of V_A
  #'
  #' For each V_A, generate simulation and fit a Gaussian model.
  #' Then, plot the obtained h^2 for observation and liability scale,
  #' alongside the true value.
  #' @param dichotomize Dichotomization method, used for simulation
  #' @param title ggplot legend title used as title for all (sub)plots
  #' @param SAVE.PLOT flag for saving plot to disk
  #' @param plot.fn (optional) String to add to the end of the
  #' filename, before file extension, when saving the plot.
  #' @param sigma.scale either "log" or "small", deciding what values,
  #' and the spacing between values of V_A to be iterated over.
  #' @param lin.pred linear predictor for simulation
  #' @param dynamic.priors Flag for changing model priors based on V_A
  #' @param simulated.formula Formula for simulatin
  #' @param Ve Residual variance, or fixed effects variance
  #' for that model
  #' @param fixedeffects Flag to determine if model has fixed effects
  #' @return List with item "p" for the ggplot object.

  if (sigma.scale == "log") {
    sigmaA.list <- c(1:10 %o% 10^(-3:3)) # Log scale[10^-3,  10^3]
  } else if (sigma.scale == "small") { # Linear scale [10^-3, 0.259]
    sigmaA.list <- seq(0.001, 0.26, by = 0.01)
  } else {
    stop("Unrecognized scale for sigmaA.")
  }
  pc.U.list <- c(rep(10, 10) %o% 10^(-3:3))
```

```
# pc.U.list <- 2*sigmaA.list # Alternative dynamic prior

estimates <- c()
latent <- c()
true.vals <- c()
est.CI.u <- c()
est.CI.l <- c()
plist <- c()
iter.num <- 0
Ve0 <- Ve
for (sigmaA in sigmaA.list) {
  cat(">")
  if (dynamic.priors) {
    iter.num <- iter.num + 1
    if (sigmaA < 1) {
      pc.prior <- c(1, 0.05)
    } else {
      pc.prior <- c(pc.U.list[iter.num], 0.05)
    }
  } else {
    pc.prior <- c(1, 0.05)
  }
  if (is.null(simulated.formula)) { # Defaults eta = a_i + e
    simulated.formula <- simulated.response ~  f(id,
      model = "generic0",
      Cmatrix = simulated.Cmatrix,
      constr = FALSE,
      hyper = list(
        prec = list(initial = log(sigmaA), prior = "pc.prec",
                    param = pc.prior)
      )
    )
  }
  if (is.null(lin.pred)) { # The 'usual' linear predictor
    lin.pred <- function(u, .) u + rnorm(length(u))
  }
  result <- simulated.heritability(
    NeNc = 0.5, idgen = 100, nGen = 9, sigmaA = sigmaA,
    linear.predictor = lin.pred,
    simulated.formula = simulated.formula,
    dichotomize = dichotomize,
    pc.prior = pc.prior
  )

  if (is.null(Ve)) {
    # Fallback residual variance
    Ve <- 1
  }
  if (fixedeffects) {
    # beta^2 * Var(x_fixedeffect):
    Ve <- Ve * var(result$simulated.response)
    posterior <- get.h2(result$fit, 10000, include.fixed = TRUE)
  } else {
```

```
    posterior <- result$heritability
  }

  simulation.h2.true <- sigmaA / (sigmaA + Ve)

  latent <- c(latent, mean(posterior)) # Observatoin-level
  threshold.scaled.h2 <- threshold.scaling.param(result$p)*posterior

  true.vals <- c(true.vals, simulation.h2.true)
  estimates.CI <- quantile(threshold.scaled.h2,
                           probs = c(0.025, 0.975))
  estimates <- c(estimates, mean(threshold.scaled.h2))
  est.CI.l <- c(est.CI.l, estimates.CI[1])
  est.CI.u <- c(est.CI.u, estimates.CI[2])
  plist <- c(plist, result$p)
  # Reset Ve
  Ve <- Ve0
}
res <- data.frame(
  estimates = estimates,
  true.vals = true.vals, latent = latent,
  est.CI.l = est.CI.l, est.CI.u = est.CI.u,
  sigmaA = sigmaA.list,
  plist = plist
)
# Plotting
p <- ggplot(data = res, aes(x = sigmaA)) +
  geom_ribbon(aes(ymin = est.CI.l, ymax = est.CI.u), alpha = 0.1) +
  geom_line(aes(y = true.vals, color = "atrue"), size = 1.5) +
  geom_line(aes(y = estimates, color = "bliab"), size = 1.5) +
  geom_line(aes(y = latent, color = "obs"), size = 1.5) +
  xlab(TeX("$\\sigma_A^2$")) +
  ylab(TeX("$h^2$")) +
  scale_x_log10() +
  scale_color_manual(
    name = title,
    values = c(
      "atrue" = "darkred", "bliab" = "steelblue",
      "obs" = "chartreuse3"
    ),
    labels = c(
      expression("True " * h["liab"]^2),
      expression("Fitted " * h["liab"]^2),
      expression("Fitted " * h["obs"]^2)
    )
  ) +
  theme(text = element_text(size = 18), legend.text.align = 0)
if (SAVE.PLOT) {
  ggsave(
    paste0(
      "../figures/simulation_deviance_",
      if (!is.na(plot.fn)) plot.fn, ".pdf"
    ), p + theme(legend.position = "none"),
```

```
      width = 20, height = 20,
      units = "cm"
    )
    # Save legend as separate plot
    p.legend <- cowplot::get_legend(p)
    pdf(paste0(
      "../figures/simulation_deviance",
      if (fixedeffects) "_fixedeffects", "_legend.pdf"
    ), width = 7.87402, height = 7.87402)
    grid.newpage()
    grid.draw(p.legend)
    dev.off()
  }
  return(list(p = p))
}
```

We also implement a method to average over several runs, reducing stochasticity. This should be ran on a remote node rather than locally, as just one run takes several minutes.

```
multiple.h2.dev <- function(sigma.scale, ntimes,
                            title = "Simulation heritability", ...){
  #' Run plot_h2_deviation `ntimes`
  #'
  #' Repeats the runs several times, and outputs error plot with
  #' mean +- SD
  #' @param sigma.scale Sigma values to test, either 'log' or 'small'
  #' @param ntimes Number of repeated runs
  #' @param title Legend title, charcter
  #' @param ... Additional parameters for plot_h2_deviation
  #' @return Dataframe with all info to plot the errorplots

  if (sigma.scale == "log") {
    sigmaA.list <- c(1:10 %o% 10^(-3:3))
  } else if (sigma.scale == "small") {
    sigmaA.list <- seq(0.001, 0.26, by = 0.01)
  } else {
    stop("Unrecognized scale for sigmaA.")
  }
  n.sigma <- length(sigmaA.list)
  # Initialize containers: each col is one run
  all.h2obs <- matrix(ncol = ntimes, nrow = n.sigma)
  all.h2liab <- matrix(ncol = ntimes, nrow = n.sigma)
  all.truevals <- matrix(ncol = ntimes, nrow = n.sigma)
  for(i in 1:ntimes){
    cat(paste0("\n [Run ", i ,"/", ntimes, "]\n"))
    res <- plot.h2.deviation(SAVE.PLOT = FALSE,
                             sigma.scale = sigma.scale, ...)
    cat("Latent:", res$p$data$latent)
    all.h2obs[, i] <- res$p$data$latent
    all.h2liab[, i] <- res$p$data$estimate
    all.truevals[, i] <- res$p$data$true.vals
  }
  plot.data <- data.frame(
    model = rep(c("h2obs", "h2liab", "true"),
```

```
                each = n.sigma, times = 2),
    top = c(rowMeans(all.h2obs) + apply(all.h2obs, 1, sd),
            rowMeans(all.h2liab) + apply(all.h2liab, 1, sd),
            rowMeans(all.truevals) + apply(all.truevals, 1, sd)
    ),
    mid = c(rowMeans(all.h2obs), rowMeans(all.h2liab),
            rowMeans(all.truevals)
    ),
    btm = c(rowMeans(all.h2obs) - apply(all.h2obs, 1, sd),
            rowMeans(all.h2liab) - apply(all.h2liab, 1, sd),
            rowMeans(all.truevals) - apply(all.truevals, 1, sd)
    ),
    xax = rep(sigmaA.list, times=3)
  )
  return(plot.data)
}
```

We test out a single run over several values below.

```
### Plots for sigmaA in (10^-3, 10^3)

simulation.res <- plot.h2.deviation(
  plot.fn = "round",
  SAVE.PLOT = TRUE, dynamic.priors = TRUE
)
simulation.res$p

### Plot for small values og sigmaA, but finer grid
simulation2.res <- plot.h2.deviation(
  SAVE.PLOT = TRUE, sigma.scale = "small",
  plot.fn = "small", dynamic.priors = TRUE
)
simulation2.res$p

### Standard plotting with different dichotomization techniques
plot.h2.deviation(
  dichotomize = "binom1.logit",
  title = "Simulation heritability",
  plot.fn = "binom", dynamic.priors = TRUE
)
```

If multiple runs have been done on remote server, loads and plots the results here,

```
# Multiple h^2 deviation plotter
markov.plotter <- function(df, legend.name=""){
  ggplot(df, aes(x=xax)) +
    geom_pointrange(aes(ymax=top, ymin=btm, y=mid, color=model)) +
    geom_line(aes(y=mid, color=model)) +
    xlab(TeX("$\\sigma_A^2$")) +
    ylab(TeX("$h^2$")) +
    scale_x_log10() +
    scale_color_manual(
      name = legend.name,
      values = c(
        "true" = "darkred", "h2liab" = "steelblue",
```

```
        "h2obs" = "chartreuse3"
      ),
      labels = c(
        expression("Fitted " * h["liab"]^2),
        expression("Fitted " * h["obs"]^2),
        expression("True " * h["liab"]^2)
      )
    ) +
    theme(text = element_text(size = 18), legend.position = "none")
}
load("markovh2dev_50_runs.Rdata")
mp1 <- markov.plotter(markov.result1)
mp2 <- markov.plotter(markov.result2)
mp3 <- markov.plotter(markov.result3,
                      legend.name="Simulation heritability")
ggsave("../figures/simulation_deviance_round.pdf", mp1)
ggsave("../figures/simulation_deviance_small.pdf", mp2)
ggsave("../figures/simulation_deviance_binom.pdf", mp3)
# Store legend separately
mp3.legend <- cowplot::get_legend(
  mp3 +theme(legend.position = "right", legend.text.align = 0))
pdf("../figures/simulation_deviance_legend.pdf",
    width = 7.87402, height = 7.87402)
grid.newpage()
grid.draw(mp3.legend)
dev.off()
```

Without residual in linear predictor

We also check what happens if we rerun with $\eta_i = a_i$, i.e. without residuals on underlying scale. Similar results.

```
get.simulated.threshold.value(
  0.5, 100, 9, 10, function(u, .) u,
  simulated.formula, 10
)
```

Now we want to look into how **A** (the relatedness matrix) looks like.

```
plot.A.matrix <- function(pedigree, title.append = NULL) {
  #' Plot histogram of relatedness from pedigree
  #'
  #' Compute relatedness matrix from pedigree and output
  #' histogram of its off-diagonal values.
  #' @param pedigree Pedigree dataframe/object
  #' @param title.append (optional) extra text in plot title
  #' @return ggplot object of histogram
  A.matrix <- nadiv::makeA(pedigree)
  A.diag <- diag(A.matrix)
  A.nondiag <- A.matrix
  diag(A.nondiag) <- NA
  ggplot(data = data.frame(values = A.nondiag@x)) +
    geom_histogram(aes(x = values, y = ..density..),
                   binwidth = 0.005) +
    ylim(c(0, 9)) +
```

```
    xlim(c(0, 0.4)) +
    labs(
      x = "Relatedness value", y = "Density",
      title = paste0("Off-diagonal values", title.append)
    )
}

# For the simulation data:
ped0 <- generatePedigree(
  nId = 100, nGeneration = 24,
  nFather = 0.5 * 100, nMother = 0.5 * 100
)
pedigree <- ped0[, c(1, 3, 2)]
names(pedigree) <- c("id", "dam", "sire")
simulated.d.ped <- nadiv::prepPed(pedigree)
plot.A.matrix(simulated.d.ped,
              title.append = ", 24 generation simulation")
if (SAVE.PLOT) {
  ggsave("../figures/relatedness-offdiagonal-sim.pdf",
    width = 20, height = 20, units = "cm"
  )
}
# Song sparrow data
plot.A.matrix(d.ped[, c("id", "mother.id", "father.id")],
              ", Song sparrow data")
if (SAVE.PLOT) {
  ggsave("../figures/relatedness-offdiagonal-songsparrow.pdf",
    width = 20, height = 20, units = "cm"
  )
}
```

Residual analyses on Gaussian model

- The first plot are the sorted PIT values over quantiles, analogous to a Q-Q plot in frequentist data. It shows a clear non-linear trend but rather a sigmoid-like curve.
- The second plot shows the PIT values across the different posterior fitted value means. Here we expect no clear pattern for well-behaved models, which is not the case in our model.
- The third plot is the residuals $y_i - \hat{y}_i$ with 95% credible interval. Here, we see a clear separation of those

```
pit.g <- fit.inla.gaussian$cpo$pit # PIT-values

# <Plot 1> Analogous to QQ-plot so should be linear
# --------
ggplot(data = data.frame(
  Quantiles = seq_along(1:length(pit.g)) / (length(pit.g) + 1),
  PIT = sort(pit.g)
)) +
  geom_point(aes(x = Quantiles, y = PIT)) +
  ggtitle("Sorted PIT values for Gaussian model") +
  theme(text = element_text(size = 14))
if (SAVE.PLOT) ggsave("../figures/PIT-sorted.pdf")

# <Plot 2> Posterior mean fitted values as a function of PIT values
```

```
# --------    analagous to "Residuals vs fitted"

ggplot(
  cbind(fit.inla.gaussian$summary.fitted.values, pit.g),
  aes(x = mean, y = pit.g)
) +
  geom_point() +
  geom_smooth() +
  labs(
    title = "PIT values over posterior mean fitted values",
    x = "Posterior fitted values (mean)",
    y = "PIT value"
  ) +
  theme(text = element_text(size = 14))
if (SAVE.PLOT) ggsave("../figures/PIT-over-fitted.pdf")


# <Plot 3> Plot of 'residuals', i.e. difference in true data and the
# --------    mean of the fitted values
df.resid <- qg.data.gg.inds$surv.ind.to.ad -
  fit.inla.gaussian$summary.fitted.values
rownames(df.resid) <- seq_len(nrow(df.resid))
df.resid$class <- qg.data.gg.inds$surv.ind.to.ad

ggplot(
  data = df.resid,
  aes(
    x = as.numeric(row.names(df.resid)),
    y = mean, color = factor(class)
  )
) +
  geom_errorbar(aes(ymin = `0.025quant`, ymax = `0.975quant`),
    color = "darkgrey"
  ) +
  geom_point() +
  scale_color_manual(
    name = "Juvenile survival",
    values = c("darkred", "steelblue")
  ) +
  labs(title = "Residuals of Gaussian model", x = "Index",
       y = "Residuals") + theme(text=element_text(size=14))
if (SAVE.PLOT) ggsave("../figures/Residuals-gaussian.pdf")
```

Transformations of heritability

Before developing methods for transformed heritability, we need to be able to sample from the marginal fitted values on latent scale.

```
marginal.latent.mode <- function(fit) {
  #' Helper function
  #'
  #' Get mode for each marginal linear predictor in `fit`
  #' @param fit Fitted INLA object
```

```
  #' @return vector of modes
  modes <- c()
  iter <- 1
  for (predictor in names(fit$marginals.linear.predictor)) {
    xy <- get(predictor, fit$marginals.linear.predictor)
    modes[iter] <- xy[, "x"][which.max(xy[, "y"])]
    iter <- iter + 1
  }
  modes
}


marginal.latent.samples <- function(fit, nsamples) {
  #' Sample values from each predictor in fit
  #'
  #' Rather than only using mode for each predictor, we use samples
  #' from its posterior.
  #' @param fit Fitted INLA object
  #' @param nsamples Number of samples
  #' @return A list of `nsamples` elements, with each element in the
  #' list being a vector of the predictor size
  #' (i.e., number of observations in data)
  out.transpose <- matrix(
    nrow = nsamples,
    ncol = length(fit$marginals.linear.predictor)
  )
  for (i in seq_along(fit$marginals.linear.predictor)) {
    xy <- get(
      names(fit$marginals.linear.predictor)[i],
      fit$marginals.linear.predictor
    )
    out.transpose[, i] <- inla.rmarginal(nsamples, xy)
  }

  # We want list where each list element is one sample (transposed)
  out <- list()
  for (i in 1:nsamples) {
    out[[i]] <- out.transpose[i, ]
  }
  out
}


report.max.skewness <- function(posterior) {
  #' Get skewness for all predictors
  #'
  #' Computes skewness and prints maximum and minimum skew with index
  #' @param posterior list of predictors, assumed form [x, y]
  #' @return None (invisible `NULL`)
  library(e1071)
  iter <- 1
  posterior.skews <- c()
  for (predictor in names(posterior)) {
```

```
    posterior.skews[iter] <- skewness(get(
      predictor, posterior)[, "x"])
    iter <- iter + 1
  }
  cat(
    "Minimum skew for list no.", which.min(posterior.skews),
  "with skewness",
    min(posterior.skews), "and max for list no.",
  which.max(posterior.skews),
    "with skewness", max(posterior.skews), ".\n"
  )
}
```

We can now define methods to obtain heritability on the different scales. The first function computes h^2 on latent scale, or using the direct transformation by including link variance in denominator. The second method is more comprehensive and uses the library `QGglmm` to obtain estimates on data scale. So far we've only used `QGglmm` without averaging over fixed effects. This takes considerably more time to process, but we still do that one time to compare the results. Then we compare the heritability on four different scales

- Using 10k samples from the *marginal linear predictor*, and using this to average over
- Grabbing the mode for each *marginal linear predictor*, and passing the mode for each 10k sample
- No averaging, i.e. use intercept value instead.
- Use direct scaling method with link variance.

```
get.h2.from.qgparams <- function(inla.fit,
                                 modelname,
                                 n,
                                 averaging = FALSE,
                                 averaging.mode.only = FALSE) {
  #' Get heritability using QGglmm::QGParams()
  #'
  #' Computes a posterior of data-scale heritability using QGParams
  #' @param inla.fit Fitted INLA model
  #' @param modelname string specifying model type
  #' @param n Number of samples for posterior distribution
  #' @param averaging Flag for averaging over fixed effects
  #' @param averaging.mode.only Other flag to determine what helper
  #' to call to.
  #' @return List of n observation-scale heritabilities

  stopifnot(modelname %in% c("Gaussian", "binom1.probit",
                             "binom1.logit"))
  samples.posterior <- inla.hyperpar.sample(n = n, inla.fit)
  vp.samples <- 0
  for (cname in colnames(samples.posterior)) {
    vp.samples <- vp.samples + 1 / samples.posterior[, cname]
  }

  if (!averaging) {
    mu <- inla.fit$summary.fixed$mean[1] # Intercept
    va.samples <- 1 / samples.posterior[, "Precision for id"]
    kwargs <- list(verbose = FALSE)

    h2.getter <- function(...) {
      get("h2.obs", suppressWarnings(QGparams(...)))
```

```
    }
    posterior <- mapply(h2.getter, mu, va.samples, vp.samples,
                        modelname, MoreArgs = kwargs
    )
    return(posterior)
  } else {
    # Average over fixed effects
    vp.samples <- 0
    df <- data.frame(
      va = as.vector(1 / samples.posterior[, "Precision for id"]),
      vp = as.vector(vp.samples)
    )
    if (!averaging.mode.only) {
      # Bayesian approach
      df$predict <- marginal.latent.samples(inla.fit, n)

      posterior <- do.call("rbind", apply(df, 1, function(row) {
        QGparams(
          predict = row[["predict"]], var.a = row[["va"]],
          var.p = row[["vp"]],
          model = modelname, verbose = FALSE
        )
      }))
    } else {
      predict.argument <- marginal.latent.mode(inla.fit)
      posterior <- do.call("rbind", apply(df, 1, function(row) {
        QGparams(
          predict = predict.argument, var.a = row[["va"]],
          var.p = row[["vp"]],
          model = modelname, verbose = FALSE
        )
      }))
    }
    return(posterior$h2.obs)
  }
}
```

Compute the 3 different approaches with `QGglmm` (this is quite slow). We also include time estimates here.

```
ti <- Sys.time()
h2.psi.sparrow <- data.frame(
  bayesian =
    get.h2.from.qgparams(fit.inla.probit, "binom1.probit",
      n.samples,
      averaging = TRUE,
      averaging.mode.only = FALSE
    )
)
cat(
  "\n-\nRuntime for Bayesian:",
  difftime(Sys.time(), ti, units = "secs"), "secs.\n"
)
ti <- Sys.time()
h2.psi.sparrow$frequentist <- get.h2.from.qgparams(fit.inla.probit,
```

```
  "binom1.probit",
  n.samples,
  averaging = TRUE,
  averaging.mode.only = TRUE
)
cat(
  "\n-\nRuntime for Frequentist:",
  difftime(Sys.time(), ti, units = "secs"), "secs.\n"
)
ti <- Sys.time()
h2.psi.sparrow$noavg <- get.h2.from.qgparams(fit.inla.probit,
  "binom1.probit",
  n.samples,
  averaging = FALSE
)
cat(
  "\n-\nRuntime for No averaging:",
  difftime(Sys.time(), ti, units = "secs"), "secs.\n"
)
```

We do the same for simulation data

```
# We store an instance of a gaussian and probit model with V_A = 1
tmp <- simulated.heritability(
  sigmaA = 1, linear.predictor = function(u, .) u + rnorm(length(u)),
  simulated.formula = simulated.formula,
  probit.model = TRUE,
  simulated.formula.probit = simulated.formula.probit
)
fit.sim.probit <- tmp$fit.probit

ti <- Sys.time()
h2.psi.sim1 <- data.frame(
  bayesian =
    get.h2.from.qgparams(fit.sim.probit, "binom1.probit",
      n.samples,
      averaging = TRUE,
      averaging.mode.only = FALSE
    )
)
cat(
  "\n-\nRuntime for Bayesian:",
  difftime(Sys.time(), ti, units = "secs"), ".\n"
)
ti <- Sys.time()
h2.psi.sim1$frequentist <- get.h2.from.qgparams(fit.sim.probit,
  "binom1.probit",
  n.samples,
  averaging = TRUE,
  averaging.mode.only = TRUE
)
cat(
  "\n-\nRuntime for Frequentist:",
  difftime(Sys.time(), ti, units = "secs"), ".\n"
```

```
)
ti <- Sys.time()
h2.psi.sim1$noavg <- get.h2.from.qgparams(fit.sim.probit,
  "binom1.probit",
  n.samples,
  averaging = FALSE
)
cat(
  "\n-\nRuntime for No averaging:",
  difftime(Sys.time(), ti, units = "secs"), ".\n"
)

# Also fit for smaller V_A, i.e. 0.1
tmp <- simulated.heritability(
  sigmaA = 0.1,
  linear.predictor = function(u, .) u + rnorm(length(u)),
  simulated.formula = simulated.formula,
  probit.model = TRUE,
  simulated.formula.probit = simulated.formula.probit
)

fit.sim.probit <- tmp$fit.probit

h2.psi.sim2 <- data.frame(
  bayesian =
    get.h2.from.qgparams(fit.sim.probit, "binom1.probit",
      n.samples,
      averaging = TRUE,
      averaging.mode.only = FALSE
    )
)
h2.psi.sim2$frequentist <- get.h2.from.qgparams(fit.sim.probit,
  "binom1.probit",
  n.samples,
  averaging = TRUE,
  averaging.mode.only = TRUE
)
h2.psi.sim2$noavg <- get.h2.from.qgparams(fit.sim.probit,
  "binom1.probit",
  n.samples,
  averaging = FALSE
)
```

```
plot.qgglmm.heritability <- function(h2.psi, dataset, SAVE.PLOT,
                                     plot.title = NA,
                                     fn.append = NULL) {
  #' Plot heriability density
  #'
  #' Compares posterior heritability using different transformations
  #' @param h2.psi Dataframe of n rows and a column for each
  #' back-transformation technique
  #' (bayesian, frequentist, no averaging, phi).
  #' @param dataset Either 'application' or 'simulation' specifying
  #' which dataset is used
```

```
  #' @param SAVE.PLOT Flag to store plot to disk
  #' @param plot.title (optional) title for plot. No title if unused.
  #' @param fn.append (optional) string to append to filename
  #' @return ggplot object
  color.map <- c(application = "Dark2", simulation = "Spectral")
  stopifnot(dataset %in% names(color.map))
  p <- ggplot(data = melt(h2.psi)) +
    geom_density(aes(x = value, fill = variable), alpha = 0.5) +
    scale_fill_brewer(
      palette = color.map[dataset],
      labels = c(
        expression(h[Psi]^2 * ", Bayesian"),
        expression(h[Psi]^2 * ", Frequentist"),
        expression(h[Psi]^2 * ", No averaging")
      )
    ) +
    ylab("Density") +
    xlab("Heritability") +
    theme(legend.text.align = 0, legend.title = element_blank()) +
    {
      if (!is.na(plot.title)) ggtitle(plot.title)
    } +
    {
      if (dataset == "simulation") xlim(c(0,quantile(
        melt(h2.psi)$value,0.99)))
    }

  if (SAVE.PLOT) {
    set_null_device(cairo_pdf)
    p.legend <- cowplot::get_legend(p)
    pdf(paste0("../figures/qgglmm-comparison-", dataset,
               "-legend.pdf"),
      width = 3, height = 3
    )
    grid.newpage()
    grid.draw(p.legend)
    dev.off()
    ggsave(
      paste0(
        "../figures/qgglmm-comparison-",
        dataset, fn.append, ".pdf"
      ),
      p + if (dataset == "simulation") theme(legend.position = "none"),
      width = 20, height = 10, units = "cm"
    )
  }
  p
}

plot.qgglmm.heritability(h2.psi.sparrow, "application",
  SAVE.PLOT,
  plot.title = NA
)
```

```
plot.qgglmm.heritability(h2.psi.sim1, "simulation",
  SAVE.PLOT,
  plot.title = NA,
  fn.append = "va1"
)
plot.qgglmm.heritability(h2.psi.sim2, "simulation",
  SAVE.PLOT,
  plot.title = NA,
  fn.append = "va0.1"
)
```

Compare different scales

First, we fit the simulation probit model and get simulation-based heritability on all scales. Now we compute the heritability on the different scales - for song sparrow data. First we make a function to help us obtain all heritability scales in a dataframe.

```
get.all.heritabilities <- function(fit.gaussian, fit.probit, p, n,
                                   fixed = FALSE) {
  #' Get h^2 for all scales
  #'
  #' For a Gaussian and probit fit, computes heritability on all scales
  #' @param fit.gaussian Fitted Gaussian model
  #' @param fit.probit Fitted Probit model
  #' @param p Phenotypic mean for the data, used in threshold formula
  #' @param n Number of samples
  #' @param fixed Flag for including fixed effects variance
  #' @return Dataframe of `n` rows with columns 'gaussian',
  #' 'guassian.liability', 'probit.latent', 'probit.scaled',
  #' 'probit.qgglmm'
  out <- data.frame(gaussian = get.h2(fit.gaussian, n,
                                      include.fixed = fixed))
  out$gaussian.liability <- threshold.scaling.param(p) * out$gaussian
  out$probit.latent <- get.h2(fit.probit, n, include.fixed = fixed)
  out$probit.scaled <- get.h2(fit.probit, n,
    model = "binom1.probit",
    use.scale = TRUE, include.fixed = fixed
  )
  out$probit.qgglmm <- get.h2.from.qgparams(fit.probit,
                                            "binom1.probit", n,
    averaging = TRUE
  )
  out
}
```

```
# Application data
heritability <- get.all.heritabilities(
  fit.inla.gaussian, fit.inla.probit,
  mean(qg.data.gg.inds$surv.ind.to.ad),
  n.samples,
  fixed = FALSE
)
```

```
simulation.res2 <- simulated.heritability(0.5, 100, 9,
  sigmaA = 0.5,
```

```
  linear.predictor = function(u, .) u + rnorm(length(u)),
  simulated.formula = simulated.formula,
  probit.model = TRUE, DIC = TRUE,
  simulated.formula.probit = simulated.formula.probit
)
heritability.sim <- get.all.heritabilities(
  simulation.res2$fit, simulation.res2$fit.probit,
  simulation.res2$p, n.samples,
  fixed = FALSE
)
```

Method to export heritability estimates in a TeX table

```
get.mode <- function(vec) {
  #' General helper to get mode of a vector
  d <- density(vec)
  d$x[which.max(d$y)]
}


print.one.metric <- function(fit, param, digits) {
  #' Helper for heritability table, rounding estiamtes
  paste(
    round(mean(get(param, fit)), digits), " & ",
    round(get.mode(get(param, fit)), digits), " & ",
    round(sd(get(param, fit)), digits),
    sep = ""
  )
}


print.heritability.table <- function(digits, h2, simulation = T) {
  #' Output LaTeX table of heritability
  #'
  #' Writes table of heritability with posterior mean, posterior mode
  #' and standard deviation, to a TeX file. Works for both datasets.
  #' @param digits Number of significant digits
  #' @param h2 Heritability DF with different scales
  #' @param simulation Simulation flag for the table's filename
  filename <- ifelse(simulation, "heritability simulation",
      "heritability application")
  header <- paste(
    "% TABLE FROM R:", format(Sys.time(), "%a %b %d %X %Y"), "\n",
    "\\begin{tabular}{lccc}\n",
    "\\hline\n",
    "Model & Mean & Mode & Standard deviation  \\\\ \n",
    "\\hline \n"
  )
  main <- paste(
    " Gaussian $h^2_\\text{obs}$ &",
    print.one.metric(h2, "gaussian", digits), "\\\\ \n",
    "Probit $h^2_{\\Psi}$ &",
    print.one.metric(h2, "probit.qgglmm", digits), "\\\\ \n",
    " & & & \\\\ \n",
```

```
    "Gaussian $h^2_\\text{liab}$ &",
    print.one.metric(h2, "gaussian.liability", digits), "\\\\ \n",
    "Probit $h^2_{\\Phi}$ &",
    print.one.metric(h2, "probit.scaled", digits), "\\\\ \n",
    "\\bottomrule"
  )
  footer <- "\\end{tabular}"

  write(paste(header, main, footer, sep = "\n"),
        file=paste0("../figures/", filename, ".tex"))
}
print.heritability.table(3, heritability, FALSE)
print.heritability.table(3, heritability.sim, TRUE)
```

Code for part of discussion:

```
write(paste0("$\\hat p=", round(simulation.res2$p,2),
             "$, that the true heritability of the observation scale",
             " would be $",
             round(1/3*1/threshold.scaling.param(simulation.res2$p),3),
             "$."),
             file="../figures/trueh2discussion.tex")
```

The table gives some indication, but we also want to look qualitatively on the densities. We start by just plotting a grid to compare each Gaussian vs. Probit scale two by two.

```
plot_grid_of_heritability <- function(heritability, SAVE.PLOT,
                                      plot.fn, colorscheme = NA) {
  #' Plot 3x2 grid of h^2 comparisons
  #'
  #' Compare density of posterior heritability between all scales for
  #' Gaussian model to all scales of the probit model. First row is
  #' observation-scale compares to latent, psi and phi, respectively.
  #' Second is the same for liability scale in the Gaussian case,
  #' and the same three cases for probit.
  #' @param heritability DF of all heritability estimates
  #' @param SAVE.PLOT Flag for storing plot to disk
  #' @param plot.fn Filename, must include file extension
  #' @param colorscheme (optional) Color palette to use for density
  #' @return List of all individual plots, as well as the grid plot
  p1 <- ggplot() +
    geom_density(
      data = melt(heritability[, c("gaussian", "probit.latent")]),
      aes(x = value, fill = variable), alpha = 0.5
    ) +
    theme(legend.position = "none", axis.title = element_blank()) +
    labs(title = TeX("$h^2_{obs}$ vs. $h^2_{lat}$")) +
    if (!is.na(colorscheme)) scale_fill_brewer(palette = colorscheme)
  p2 <- ggplot() +
    geom_density(
      data = melt(heritability[, c(
        "gaussian.liability",
        "probit.latent"
      )]),
      aes(x = value, fill = variable), alpha = 0.5
```

```
  ) +
  theme(legend.position = "none", axis.title = element_blank()) +
  labs(title = TeX("$h^2_{liab}$ vs. $h^2_{lat}$")) +
  if (!is.na(colorscheme)) scale_fill_brewer(palette = colorscheme)
p3 <- ggplot() +
  geom_density(
    data = melt(heritability[, c(
      "gaussian",
      "probit.qgglmm"
    )]),
    aes(x = value, fill = variable), alpha = 0.5
  ) +
  theme(legend.position = "none", axis.title = element_blank()) +
  labs(title = TeX("$h^2_{obs}$ vs. $h^2_{\\Psi}$")) +
  if (!is.na(colorscheme)) scale_fill_brewer(palette = colorscheme)
p4 <- ggplot() +
  geom_density(
    data = melt(heritability[, c(
      "gaussian.liability",
      "probit.qgglmm"
    )]),
    aes(x = value, fill = variable), alpha = 0.5
  ) +
  theme(legend.position = "none", axis.title = element_blank()) +
  labs(title = TeX("$h^2_{liab}$ vs. $h^2_{\\Psi}$")) +
  if (!is.na(colorscheme)) scale_fill_brewer(palette = colorscheme)
p5 <- ggplot() +
  geom_density(
    data = melt(heritability[, c(
      "gaussian",
      "probit.scaled"
    )]),
    aes(x = value, fill = variable), alpha = 0.5
  ) +
  theme(legend.position = "none", axis.title = element_blank()) +
  labs(title = TeX("$h^2_{obs}$ vs. $h^2_{\\Phi}$")) +
  if (!is.na(colorscheme)) scale_fill_brewer(palette = colorscheme)
p6 <- ggplot() +
  geom_density(
    data = melt(heritability[, c(
      "gaussian.liability",
      "probit.scaled"
    )]),
    aes(x = value, fill = variable), alpha = 0.5
  ) +
  theme(legend.position = "none", axis.title = element_blank()) +
  labs(title = TeX("$h^2_{liab}$ vs. $h^2_{\\Phi}$")) +
  if (!is.na(colorscheme)) scale_fill_brewer(palette = colorscheme)
set_null_device(cairo_pdf)
p <- plot_grid(p1, p3, p5, p2, p4, p6, ggplot() +
  theme_void(),
get_legend(
  ggplot(data.frame(v = c("Gaussian", "Probit"), x = c(0, 0))) +
```

```
      geom_density(aes(x = x, fill = v), alpha = 0.5) +
      scale_fill_discrete(breaks = c("Gaussian", "Probit")) +
      theme(legend.title = element_blank()) +
      if (!is.na(colorscheme)) scale_fill_brewer(palette =
                                                   colorscheme)
  ),
  axis = "tblr"
  )
  if (SAVE.PLOT) ggsave(paste0("../figures/", plot.fn), p)
  list(p1 = p1, p2 = p2, p3 = p3, p4 = p4, p5 = p5, p6 = p6, p = p)
}
plot.h2.appl <- plot_grid_of_heritability(
  heritability, SAVE.PLOT, "grid_application_gaussian_vs_binom.pdf",
  "Dark2"
)
# For simulation:
plot.h2.sim <- plot_grid_of_heritability(
  heritability.sim, SAVE.PLOT,"grid_simulation_gaussian_vs_binom.pdf",
  "Spectral"
)

plot.h2.appl$p
plot.h2.sim$p
```

Key takeaways:

- The Gaussian model to liability scale doesn't fit well with the other latent models.
- The scalings from binomial latent onto data scale fit well with the Gaussian one

The ones of greatest importance are plots $(1,2)$ (p3) and $(2,3)$ (p6), so we extract them in particular

```
plot.h2.appl$p3 +
  theme(legend.position = "right", legend.title = element_blank()) +
  scale_fill_brewer(palette = "Dark2",labels = c(
    TeX("Gaussian $h^2_{obs}$"), TeX("Probit $h^2_\\Psi$"))
    ) +
  theme(legend.text.align = 0, legend.position = "bottom",
        legend.text = element_text(size = 23)) + ggtitle("")

if (SAVE.PLOT) ggsave(
  "../figures/heritability_application_obsscale.pdf")

plot.h2.appl$p6 +
  theme(legend.position = "right", legend.title = element_blank()) +
  scale_fill_brewer(palette = "Dark2", labels = c(
    TeX("Gaussian $h^2_{liab}$"), TeX("Probit $h^2_\\Phi$"))) +
  theme(legend.text.align = 0, legend.position = "bottom",
        legend.text = element_text(size = 23)) + ggtitle("")
if (SAVE.PLOT) ggsave(
  "../figures/heritability_application_liabscale.pdf")

plot.h2.sim$p3 +
  theme(legend.position = "right", legend.title = element_blank()) +
  scale_fill_brewer(palette = "Spectral", labels = c(
    TeX("Gaussian $h^2_{obs}$"), TeX("Probit $h^2_\\Psi$"))
```

```
    ) +
  theme(legend.text.align = 0, legend.position = "bottom",
        legend.text = element_text(size = 23)) + ggtitle("")
if (SAVE.PLOT) ggsave(
  "../figures/heritability_simulation_obsscale.pdf")
plot.h2.sim$p6 +
  theme(legend.position = "right", legend.title = element_blank()) +
  scale_fill_brewer(palette = "Spectral", labels = c(
    TeX("Gaussian $h^2_{liab}$"),
    TeX("Probit $h^2_\\Phi$"))) +
  theme(legend.text.align = 0, legend.position = "bottom",
        legend.text = element_text(size = 23)) + ggtitle("")
if (SAVE.PLOT) ggsave(
  "../figures/heritability_simulation_liabscale.pdf")
```

Finally, we want to look at DIC values for simulation model. It can't be compared to the song sparrow data directly, but how much the Gaussian and probit differ can be compared.

```
data.frame(
  Gaussian = simulation.res2$fit$dic$dic,
  Probit = simulation.res2$fit.probit$dic$dic,
  row.names = "Deviance Information Criteria"
)
```

Fixed effects for simulation

We now add a sex covariate to the linear predictor. We use that

$$\mathrm{Var}[\beta_{\mathrm{sex}}\mathbf{x}_{\mathrm{sex}}] = \beta_{\mathrm{sex}}^2\sigma_{\mathrm{sex}}^2$$

```
linear_predictor_fixedeffects <- function(u, simulated.d.ped) {
  #' \Tilde{\eta} = a + N(0, varE) + betaSex x_{sex}
  varE <- 1
  betaSex <- 100
  out <- c()
  intercept <- 0
  residuals <- rnorm(length(u), mean = 0, sd = sqrt(varE))
  for (idx in seq_along(u)) {
    out <- c(
      out,
      intercept + betaSex * simulated.d.ped$sex[idx] + u[idx] +
        residuals[idx]
    )
  }
  out
}
simulated.formula.fixedeffects <- simulated.response ~ sex +
  f(id,
    model = "generic0",
    Cmatrix = simulated.Cmatrix,
    constr = FALSE,
    hyper = list(
      prec = list(
        initial = log(1 / 10), prior = "pc.prec",
```

```
        param = c(1, 0.05)
      ) # PC priors
    )
  )

# u = a + 100*x_sex + N(0,1) - balanced binary trait
m <- plot.h2.deviation(
  dichotomize = "round_balanced", title = TeX("$\\beta_{sex}=100$"),
  SAVE.PLOT = SAVE.PLOT, plot.fn = "fixedeffects_beta100",
  sigma.scale = "log",
  lin.pred = linear_predictor_fixedeffects,
  simulated.formula = simulated.formula.fixedeffects,
  Ve = 100^2, fixedeffects = TRUE, dynamic.priors=TRUE
)
m$p + xlim(c(1,10^4)) + theme(legend.position = "none")
if(SAVE.PLOT){
  ggsave("../figures/simulation_deviance_fixedeffects_beta100.pdf")
  } # Re-save figure with specified x-lim.

# Re-run with smaller magnitude for fixed effect
linear_predictor_fixedeffects <- function(u, simulated.d.ped) {
  varE <- 1
  betaSex <- 10
  out <- c()
  intercept <- 0 #-4.5
  residuals <- rnorm(length(u), mean = 0, sd = sqrt(varE))
  for (idx in seq_along(u)) {
    out <- c(
      out,
      intercept + betaSex * simulated.d.ped$sex[idx] + u[idx] +
        residuals[idx]
    )
  }
  out
}

# u = a + 10*x_sex + N(0,1) - balanced binary trait
m2 <- plot.h2.deviation(
  dichotomize = "round_balanced", title = TeX("$\\beta_{sex}=10$"),
  SAVE.PLOT = SAVE.PLOT, plot.fn = "fixedeffects_beta10",
  sigma.scale = "log",
  lin.pred = linear_predictor_fixedeffects,
  simulated.formula = simulated.formula.fixedeffects,
  Ve = 10^2, fixedeffects = TRUE, dynamic.priors = TRUE
)
m2$p

# u = a + 10*x_sex + N(0,1) - somewhat unbalanced
m3 <- plot.h2.deviation(
  dichotomize = 0.1,
  title = "Simulation heritability for \nfixed effects model",
  SAVE.PLOT = SAVE.PLOT, plot.fn = "fixedeffects_beta10_unbalanced",
  sigma.scale = "log",
```

```
  lin.pred = linear_predictor_fixedeffects,
  simulated.formula = simulated.formula.fixedeffects,
  Ve = 10^2, fixedeffects = TRUE, dynamic.priors = TRUE
)
m3$p

# How unbalanced is the response?
summary(m3$p$data$plist)
```

Similar to the case without fixed effects, we provide code for plotting based on results form remote server.

```
load("markovfixed_50_runs.Rdata")
mp4 <- markov.plotter(res.fixed1)
mp5 <- markov.plotter(res.fixed2)
mp6 <- markov.plotter(res.fixed3,
  legend.name="Simulation heritability for\nfixed effects model")
fixed.fn <- "../figures/simulation_deviance_fixedeffects_"
ggsave(paste0(fixed.fn, "beta100.pdf"),
       mp4+xlim(c(1,10^4)))
ggsave(paste0(fixed.fn, "beta10.pdf"), mp5)
ggsave(paste0(fixed.fn, "beta10_unbalanced.pdf"), mp6)
mp6.legend <- cowplot::get_legend(
  mp6 +theme(legend.position = "right", legend.text.align = 0))
pdf("../figures/simulation_deviance_fixedeffects_legend.pdf",
    width = 7.87402, height = 7.87402)
grid.newpage()
grid.draw(mp6.legend)
dev.off()
```

For sufficiently large choice of β corresponding to sex, we get progressively worse results as is expected.

Fixed effect model performance

Another aspect we can examine, is how the grid plots of heritability scales would look like if we use a Gaussian and probit model with (somewhat dominating) fixed effect. This is implemented below.

```
plot.fixedeffects.h2 <- function(sA, .dichotomize, include.fixed = T,
                                 sE = 1, beta = 10, SAVE.PLOT = T,
                                 plot.legend=F) {
  #' Plot h2 density of gaussian and backtransformed probit model,
  #'
  #' Fit simulation models with fixed effects, compute h2 for
  #' Gaussian and probit case, backtransform probit h2 and plot
  #' @param sA Additive genetic variance sigma^2_A
  #' @param .dichotomize character denoting dichotomization method
  #' @param include.fixed Wether or not to include in denom. of h2
  #' @param sE Error variance sigma^2_E
  #' @param beta Weight for fixed effect in linear predictor
  #' @param SAVE.PLOT Flag for storing plot to disk
  #' @param plot.legend Flag for including legend in saved plot
  .pc.prior <- c(10^(ceiling(log10(sA))), 0.05)
  .linear_predictor_fixedeffects <- function(u, simulated.d.ped) {
  out <- c()
  intercept <- 0
  residuals <- rnorm(length(u), mean = 0, sd = sqrt(sE))
```

```
  for (idx in seq_along(u)) {
    out <- c(
      out,
      intercept + beta * simulated.d.ped$sex[idx] + u[idx] +
        residuals[idx]
    )
    }
  out
  }
  fits <- simulated.heritability(
    idgen = 100, dichotomize = .dichotomize, pc.prior = .pc.prior,
    sigmaA = sA, linear.predictor = .linear_predictor_fixedeffects,
    simulated.formula = simulated.formula.fixedeffects,
    probit.model = TRUE,
    simulated.formula.probit = simulated.formula.probit
  )
  h2.sim.fixed <- get.all.heritabilities(fits$fit, fits$fit.probit,
                                         fits$p, n.samples,
                                         fixed = FALSE)

  p <- ggplot(melt(h2.sim.fixed[, c("gaussian", "probit.qgglmm")])) +
    geom_density(aes(x = value, fill = variable), alpha = 0.5) +
    scale_fill_discrete(
      name = "",
      labels = c(TeX("Gaussian $h^2_{obs}"),TeX("Probit $h^2_\\Psi$"))
    ) +
    theme(legend.text.align = 0) +
    xlab("Heritability") +
    ylab("Density") + theme(legend.position = "bottom")
  if (SAVE.PLOT) {
    p.legend <- cowplot::get_legend(p)
    pdf("../figures/fixedeffects_gaussian_probit_legend.pdf",
        width = 5.5, height = 1)
    grid.newpage()
    grid.draw(p.legend)
    dev.off()
    legend.pos <- if(plot.legend) "right" else "none"
    plot.height <- if(plot.legend) 10 else 20
    fn_append <- if(plot.legend) "_wide" else NULL
    fn_append <- if(beta != 10) paste0(fn_append, "_beta_", beta) else
      fn_append
    ggsave(
      paste0("../figures/fixedeffects_gaussian_probit_sA", sA,
      "_p_", 10*round(fits$p,1), fn_append, ".pdf"),
      p+theme(legend.position = legend.pos), width = 20,
      height = plot.height, units = "cm"
    )
  }
}
plot.fixedeffects.h2(10, 0.1,plot.legend=T)
plot.fixedeffects.h2(10, 0.1)
plot.fixedeffects.h2(10, "round_balanced")
plot.fixedeffects.h2(500, 0.1)
```

```
plot.fixedeffects.h2(500,"round_balanced")

# For appendix:
for(beta_sex in c(1,5)){
  plot.fixedeffects.h2(sA=10, .dichotomize = "round_balanced",
                       beta=beta_sex)
  plot.fixedeffects.h2(sA=500, .dichotomize = "round_balanced",
                       beta=beta_sex)
}
```

IID noise to probit simulation

```
alternative.probit.sim <- function(sigmaA, linear.predictor,
                                   fit.gaussian = NULL) {
  #' Simulate and fit model with and without extra noise
  #'
  #' Modified version of `simulated_heritability()` to fit probit
  #' model, one with an extra IID noise in INLA formula,
  #' and one without.
  #' @param sigmaA Additive genetic variance
  #' @param linear.predictor Callable of two variables, for
  #' simulating response
  #' @param fit.gaussian Flag for also fitting Gaussian model.
  #' Will fit as long
  #' as it's not `NULL`.
  #' @return List of two probit fits, gaussian fit (or `NULL`) and p,
  #' the simulation's phenotypic mean.

  # Init
  idgen <- 100
  NeNc <- 0.5
  nGen <- 9

  # Generate pedigree
  ped <- generatePedigree(
    nId = idgen, nGeneration = nGen, nFather = idgen * NeNc,
    nMother = idgen * NeNc
  )
  ped <- ped[, c(1, 3, 2, 5)]
  names(ped) <- c("id", "dam", "sire", "sex")
  u <- rbv(ped[, c(1, 2, 3)], sigmaA)
  simulated.d.ped <- nadiv::prepPed(ped, gender = "sex")
  simulated.Cmatrix <- nadiv::makeAinv(ped[, c(1, 2, 3)])$Ainv
  simulated.d.ped$ind <- seq_len(nrow(simulated.d.ped))

  # Generate binary response
  simulated.response <- ifelse(
    linear.predictor(u, simulated.d.ped) <= 0, 0, 1)
  p <- mean(simulated.response)

  # INLA fitting
  formula.overdisp <- simulated.response ~ f(id,
```

```
    model = "generic0", Cmatrix = simulated.Cmatrix,
    constr = FALSE,
    hyper = list(prec = list(
      initial = log(1 / 10), prior = "pc.prec",
      param = c(1, 0.05)
    ))
  ) +
    f(ind,
      model = "iid", constr = TRUE
    )
  formula.standard <- simulated.response ~ f(id,
    model = "generic0", Cmatrix = simulated.Cmatrix,
    constr = FALSE,
    hyper = list(prec = list(
      initial = log(1 / 10), prior = "pc.prec",
      param = c(1, 0.05)
    ))
  )


  fit.overdisp <- inla(
    formula = formula.overdisp, family = "binomial",
    data = simulated.d.ped,
    control.compute = list(return.marginals.predictor = TRUE)
  )
  fit.standard <- inla(
    formula = formula.standard, family = "binomial",
    data = simulated.d.ped,
    control.compute = list(return.marginals.predictor = TRUE)
  )
  if (!is.null(fit.gaussian)) {
    # Also compute Gaussian model
    fit.gaussian <- inla(
      formula = formula.standard, family = "gaussian",
      data = simulated.d.ped
    )
  }


  list(
    fit.overdisp = fit.overdisp,
    fit.standard = fit.standard,
    fit.gaussian = fit.gaussian,
    overdisp.p = p
  )
}
```

Here, we fit a probit with the formula $y_i = \beta_0 + a_i + \gamma_{0,i}$, where the last is a random iid effect. The simulated data has overdisperion in its data via the residual vector being $\mathcal{N}(0, 3)$.

```
overdisperion_wrapper <- function(nsamps, vA, vE, SAVE.PLOT) {
  #' Wrapper for running overdispersion tests
  #'
  #' Wrapper for calling alternative model fitting with extra noise,
```

```
  #' and for
  #' plotting thee results.
  #' @param nsamps Number of samples for posterior heritability
  #' @param vA Additive genetic variance
  #' @param vE Additional noise (should be more than 1)
  #' @param SAVE.PLOT Flag for storing plot to disk

  list2env( # Loads fit.overdisp, fit.standard, fit.gaussian, p
    alternative.probit.sim(vA, function(u, .) u + rnorm(length(u),
                                                        0, sqrt(vE)),
      fit.gaussian = TRUE
    ), .GlobalEnv
  )
  df.probit.comp <- data.frame(
    Overdispersion = get.h2.from.qgparams(fit.overdisp,
                                          "binom1.probit",
      nsamps,
      averaging = TRUE
    ),
    Standard = get.h2.from.qgparams(fit.standard, "binom1.probit",
      nsamps,
      averaging = TRUE
    ),
    Gaussian = get.h2(fit.gaussian, nsamps)
  )

  curr.plot <- ggplot(data = melt(df.probit.comp)) +
    geom_density(aes(x = value, fill = variable), alpha = 0.5) +
    scale_fill_discrete(name = "", labels = c(
      TeX("$h^2_\\Psi$ with iid effect"),
      TeX("$h^2_\\Psi$ without iid effect"),
      TeX("$h^2_{obs}$")
    )) +
    xlab("Heritability") +
    ylab("Density") +
    theme(legend.position = "bottom") +
    xlim(c(0, quantile(melt(df.probit.comp)$value, 0.95)))
  if (SAVE.PLOT) {
    p.legend <- cowplot::get_legend(curr.plot)
    pdf("../figures/overdisperions_legend.pdf", width = 9, height = 1)
    grid.newpage()
    grid.draw(p.legend)
    dev.off()
    ggsave(
      paste0("../figures/overdispersion_vE-vA_", vE, "-", vA, ".pdf"),
      curr.plot + theme(legend.position = "none")
    )
  }
}

overdisperion_wrapper(10000, 0.5, 2, SAVE.PLOT)
overdisperion_wrapper(10000, 0.5, 5, SAVE.PLOT)
overdisperion_wrapper(10000, 0.5, 10, SAVE.PLOT)
```

```
overdisperion_wrapper(10000, 10, 10, SAVE.PLOT)
```

Illustrative figures

Addendum - Illustrative figure for fitted values in a probit model. This is intended to demonstrate the different scales you get when using GLMMs.

```
mod <- simulated.heritability(
  linear.predictor = function(u, .) u + rnorm(length(u)),
  simulated.formula = simulated.formula,
  probit.model = T,
  simulated.formula.probit = simulated.formula.probit
)
pmod <- mod$fit.probit
eta_samples <- marginal.latent.samples(pmod, 20)

# Plotting
sample_ids <- sort(c(LETTERS, letters))[1:40]

eta_df <- data.frame(
  elem = c(
    rep(sample_ids[1:20], each = 900), # Latent eta
    rep(sample_ids[21:40], each = 900)), # Phi(eta)
  value = c(
    unlist(eta_samples),
    pnorm(unlist(eta_samples))
  )
)
custom_palette <- c(
  colorRampPalette(c("pink", "darkred"))(20),
  colorRampPalette(c("lightblue", "darkblue"))(20), "darkgreen"
)

ggplot(eta_df) +
  geom_line(aes(x = value, color = elem), stat = "density",
            alpha = 0.25, size = 2) +
  # True values
  geom_vline(xintercept=0,size=2,color='green4') +
  geom_vline(xintercept=1, size=2, color='green4') +
  scale_color_manual(values = custom_palette) +
  theme(legend.position = "none") +
  xlab("(Predicted) response") +
  ylab("Density")
if (SAVE.PLOT) {
  ggsave("../figures/illustration_probit_scales_fitted_values.pdf",
    width = 20, height = 10, units = "cm"
  )
  # Generate legend
  plegend <- ggpubr::get_legend(
    ggplot(melt(data.frame(r = rnorm(1), b = rnorm(1),
                           t = rnorm(1)))) +
      geom_line(aes(x = value, color = variable), stat = "density",
                size = 2, alpha = 0.9) +
```

```
      scale_color_manual(
        name = "",
        labels = c(TeX("$\\eta$"), TeX("$\\Phi(\\eta)$"),
                   "True observations"),
        values = c(r = "darkred", b = "darkblue", t = "green4")
      ) +
      theme(legend.text.align = 0, legend.position = "bottom",
            text = element_text(family = texfont)
      )
  )
  pdf("../figures/illustration_probit_scales_fitted_values_legend.pdf",
    width = 4.2, height = 1
  )
  grid.newpage()
  grid.draw(plegend)
  dev.off()
}
```

The second is an illustrative figure for binomial vs linear regression in general.

```
data(mtcars)
library(ggplot2)
library(cowplot)
p1 <- ggplot(mtcars, aes(x = hp, y = vs)) +
  geom_point(alpha = .5) +
  ggtitle("Binomial regression") +
  stat_smooth(method = "glm", se = F,
              method.args = list(family = binomial)) +
  ylim(c(-0.2, 1.01)) +
  theme(title = element_text(size = 16))
p2 <- ggplot(mtcars, aes(x = hp, y = vs)) +
  geom_point(alpha = .5) +
  stat_smooth(method = "lm", se = F) +
  ggtitle("Linear regression") +
  ylim(c(-0.2, 1.01)) +
  theme(title = element_text(size = 16))

plot_grid(p1, p2, ncol = 2, align = "v", axis = "tb")
if (SAVE.PLOT) {
  ggsave("../figures/linear-vs-logistic-example.pdf",
    width = 20, height = 10, units = "cm"
  )
}
```

The final plot is an illustration of the threshold model.

```
threshold.illustration <- function(){
  x <- seq(-3, 3, length.out = 100)
  threshold <- -0.4
  df <- data.frame(x = x, y = dnorm(x))
  df$samps <- c(rep(NA,10), runif(80, 0, dnorm(x[11:90])), rep(NA,10))
  df$sampscol <- ifelse(df$x >= threshold, "a", "b")
  cols <- c(hcl(h=seq(15,375,length=3), l=65, c=100)[1:2])
  # Create the ggplot
  ggplot(df, aes(x = x, y = y)) +
```

```
    geom_point(aes(x = x, y = samps, colour = sampscol)) +
    geom_line(linewidth=0.7) + ylab("") + xlab("") +
    geom_vline(aes(xintercept=threshold, linetype='Threshold M'),
               linewidth=0.7) +
    theme_classic(24) + theme(text=element_text(family=texfont)) +
    scale_color_manual(name="", values=cols,
                       labels=c("Phenotype 2","Phenotype 1")) +
    scale_linetype_manual(name="", values=c('Threshold M'=2))
  ggsave("../figures/illustration_thresholdmodel.pdf",
         width=20, height=10, units="cm")
}
threshold.illustration()
```