**Supplementary material**

**Fig. S1.** Arrangement of genotypes in the multiple tree plot design. Different colors shows different genotypes from half-, full-sib, or clonal families.

****

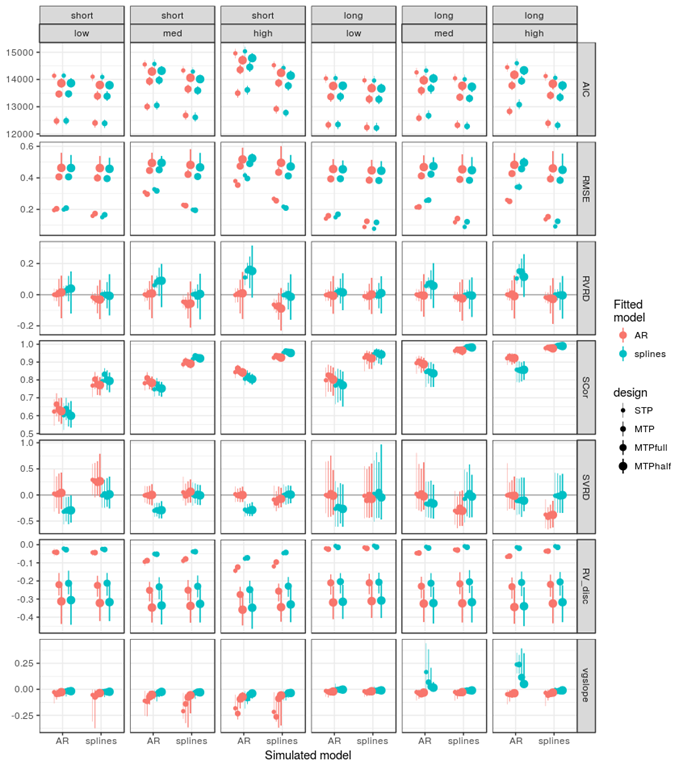
**Fig. S2.** Random realizations of examples of simulated spatial patterns with the bidimensional separable first-order autoregressive (AR) and bidimensional spline regression (Splines) models at short and long spatial ranges. The data are presented in a 70 × 70 grid.

****

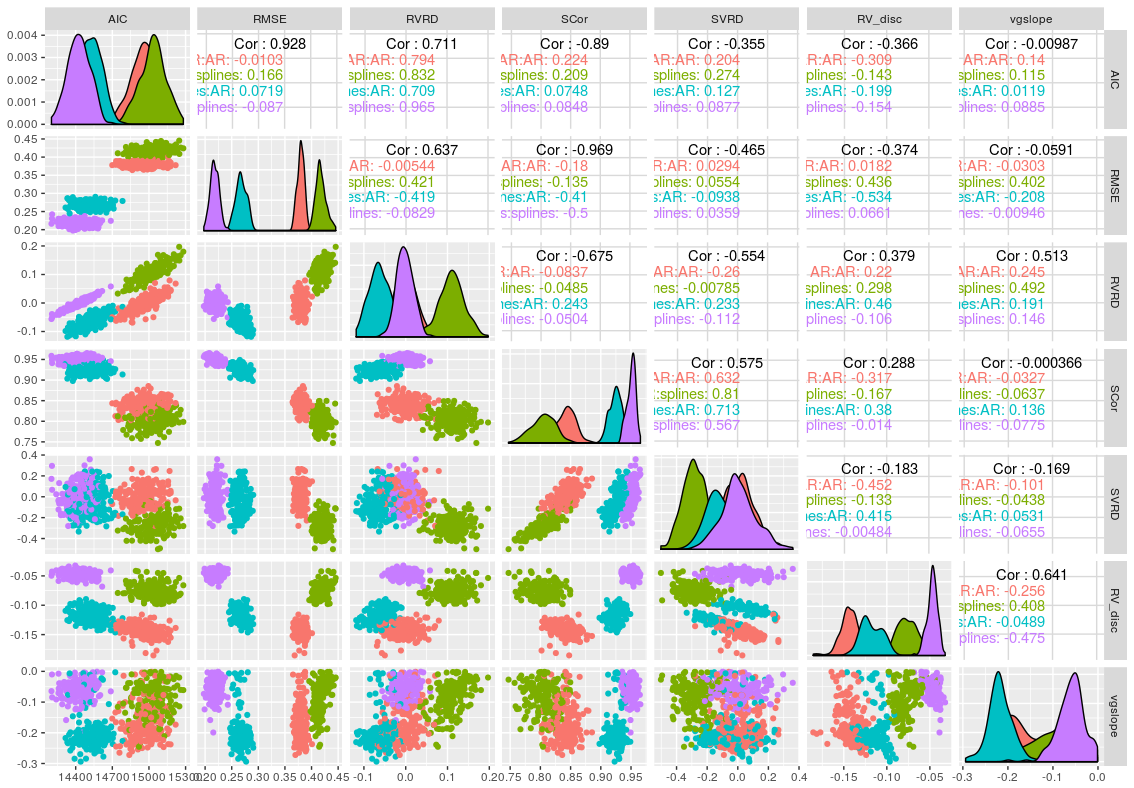
**Fig. S3.** Empirical semivariograms of realized spatial effects from the bidimensional separable first-order autoregressive (AR) and bidimensional spline regression (Splines) models at short and long spatial range.

|  |  |
| --- | --- |
| **AR** | **Splines** |
|  |  |

**Fig. S4.** Median and 90% central quantile of sampling distribution of metrics (Akaike Information Criterion -AIC-, root mean squared error of prediction of the expected phenotype -RMSE-, residual variance relative deviation -RVRD-, correlation between predicted and true simulated spatial effects -Scor-, and spatial variance relative deviation -SVRD-) by simulated model (bidimensional separable first-order autoregressive -AR- and bidimensional spline regression -splines-), spatial range (short and long) and plot structure (single-tree plot -STP- and multiple tree plot -MTP-).



**Fig. S5.** Scatter-plot matrix among metrics (empirical and estimated residual variances differences -RV\_disc-, and slope of the empirical semivariogram of residuals at distance 0 -vgslope-) by generating and fitting model for a short-ranged spatial effect under a single-tree plot design.



**R\_code S1.** R code used to generate the datasets used in the simulation experiment.

#Copyright 2018 Facundo Muñoz.

#This code is free software: you can redistribute it and/or modify

#it under the terms of the GNU General Public License as published by

#the Free Software Foundation, either version 3 of the License, or

#(at your option) any later version.

#This program is distributed in the hope that it will be useful,

#but WITHOUT ANY WARRANTY; without even the implied warranty of

#MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

#GNU General Public License for more details.

#You should have received a copy of the GNU General Public License

#along with this program. If not, see <https://www.gnu.org/licenses/>

## 1. Dataset simulation functions ####

library(ggplot2)

## Get cached file name given a base name

get\_cached <- function(x)

file.path(work.dir, "cache", paste(x, "rds", sep = "."))

## Simulate a list of replicate datasets, given a list of

## simulation parameters and a file name to store the results

simlist <- function(spar, fname) {

## Set up ##

library(breedR)

library(tidyr)

## Design of the experiment

design <- expand.grid(

model = spar$models,

design = spar$design,

plot.str = spar$plot.str,

range = spar$ranges,

sp.vratio = spar$sp.vratios,

KEEP.OUT.ATTRS = FALSE)

siml <- rep(apply(design, 1, as.list),

spar$N.replicates / spar$n.files)

## Simulate the datasets

simdat <- function(x, spar)

within(x, dat <- simparam(model, range, sp.vratio, spar))

siml <- lapply(siml, simdat, spar)

## Save the simulation parameters as an attribute of the siml object

attr(siml, "sim\_parameters") <- spar

## Name instances by order (useful for ldply later)

names(siml) <- seq\_along(siml)

## Save the simulated datasets

saveRDS(siml, fname)

}

## Simulate a dataset from a combination of parameter values

simsp <- function(dim = c(70, 70),

design = c("STP", "MTP"),

plot.str = c("clones", "full-sibs", "half-sibs"),

sp.model = c("AR", "splines"),

sp.param,

sp.vratio) {

param.name <- ifelse(sp.model == "AR", "rho", "n.knots")

sp.names <- c("model", param.name,

"grid.size", "sigma2\_s")

sp.component <- structure(

list(sp.model, sp.param, dim, sp.vratio),

names = sp.names

)

dat <- breedR.sample.phenotype(

fixed = c(mu = 0),

spatial = sp.component,

residual.variance = 1

)

# ## center the spatial surface around zero, and correct the phenotype

# ## NO: if I do this I will be systematically removing variance

# ## specially for splines with few knots!!

# dat$spatial <- dat$spatial - mean(dat$spatial)

# dat$phenotype <- dat$phenotype - mean(dat$spatial)

if (design == "MTP") {

## Include a genetic "clonal" effect in the residuals

## genetic parameters: clonal genotypes with independent values

N.genotypes <- dim[1]

line.size <- 7

add.gen.var <- 1 / 3

res.var <- 2 / 3

stopifnot(dim[2] %% line.size == 0)

N.replicas <- dim[2] / line.size

## I will simulate each BV using the within/between family decomposition

## BV = x1 + x2

## The covariance of members of different families is 0

## while for members of the same family equals

## Cov(BV, BV') = cov(x1+x2, x1'+x2') = cov(x1, x1') = var(x1)

## genetic variance between families

gen.vars <- c(clones = 1, "full-sibs" = .5, "half-sibs" = .25)

gen.var <- gen.vars[plot.str] \* add.gen.var

## simulate N.genotypes genetic values and arrange them into N.replicas

gen.values <- as.vector(rnorm(N.genotypes, sd = sqrt(gen.var)))

gen.arrangement <- replicate(N.replicas, sample(N.genotypes))

## simulate within-family genetic values

inner.gen.values <- rnorm(prod(dim), sd = sqrt(add.gen.var - gen.var))

## vector index in the arrangement for each position in the dataset (same

## order)

gen.idx <- with(dat, (Var1-1) \* N.replicas + (Var2-1) %/% line.size + 1)

## include genetic effect, remove ancient residual from simulated phenotype

## and simulate new residuals with new variance

dat <- transform(

dat,

genotype = gen.arrangement[gen.idx],

genetic = gen.values[gen.arrangement[gen.idx]] + inner.gen.values,

phenotype = phenotype - resid,

resid = as.vector(rnorm(nrow(dat), sd = sqrt(res.var))))

# check

# breedR:::spatial.plot(dat[, c("Var1", "Var2", "genotype")])

## Correct phenotype

dat <- transform(

dat,

phenotype = phenotype + genetic + resid)

}

## save parameter values as attributes

attr(dat, "design") <- design

attr(dat, "plot.str") <- plot.str

attr(dat, "model") <- sp.model

attr(dat, "param") <- sp.param

attr(dat, "vratio") <- sp.vratio

return(dat)

}

## Simulate a dataset from a scenario

simparam <- function(model, range, sp.vratio, spar) {

simsp(

dim = spar$dimensions,

design = spar$design,

plot.str = spar$plot.str,

sp.model = model,

sp.param = rep(switch(model, AR = spar$rho.values[range],

splines = spar$nk.values[[range]]), 2),

sp.vratio = spar$vratio.values[sp.vratio]

)

}

## Select instances from a simulated list

whichsim <- function(model, range, vratio, siml) {

which(sapply(

siml,

function(x) x$model == model && x$range == range && x$sp.vratio == vratio))

}

#### Model fitting functions ####

## Fit a spatial model to a dataset

fitsp <- function(dat, model, param) {

par.names <- c(AR = "rho", splines = "n.knots")

methods <- c(AR = "ai", splines = "em")

sp.names <- c("model", par.names[model], "coordinates")

sp.component <- structure(

list(model, param, dat[, c("Var1", "Var2")]),

names = sp.names)

## Include a genetic component if it was simulated

## We fit only the clonal effects here.

## For half/full-sib families, this component represents only one part of the

## genetic effect BV = x\_f + x\_i.

## The individual part (mendelian seg.) is independent and is accounted for in

## the residuals. var(x\_f) = f\*var(x\_f+x\_i) where f = 0.5 or 0.25 for full and

## half-sib families respectively.

if ("genetic" %in% names(dat)) {

random.component <- ~ genotype

} else {

random.component <- NULL

}

remlf90(

phenotype ~ 1,

random = random.component,

spatial = sp.component,

data = dat,

method = methods[model]

)

}

## Fit a simulated dataset with splines with a (isotropic) grid search of knots

fitspl <- function(dat, nk.values) {

suppressMessages(

ans <- lapply(nk.values, function(y) fitsp(dat, "splines", rep(y, 2)))

)

grid <- data.frame(nk = nk.values,

ll = sapply(ans, logLik))

# ggplot(grid, aes(nk, ll)) + geom\_point() + geom\_line()

idx <- which.max(grid$ll)

ans <- ans[[idx]]

# Include estimation information

ans$grid.search <- grid

return(ans)

}

## Fit a simulated dataset with AR with a (isotropic) grid search of rho

fitar <- function(dat, rho.values) {

suppressMessages(

ans <- lapply(rho.values, function(y) fitsp(dat, "AR", rep(y, 2)))

)

grid <- data.frame(rho = rho.values,

ll = sapply(ans, logLik))

# ggplot(grid, aes(rho, ll)) + geom\_point() + geom\_line()

idx <- which.max(grid$ll)

ans <- ans[[idx]]

# Include estimation information

ans$grid.search <- grid

return(ans)

}

## Define grid-search values around true value

grid\_rho <- function(x)

round(plogis(qlogis(x) + -3:3/2), 2)

grid\_nk <- function(x, min.nk = 2)

round(exp(log(x - min.nk) + -3:3/6) + min.nk)

## Fit a simulated dataset with both AR and splines models using

## a sensible grid search

fitsim <- function(x, spar) {

library(breedR)

## AR grid

current.rho <- spar$rho.values[x$range]

grid.rho <- grid\_rho(current.rho)

## remove potentially duplicated rho values

if(any(idx <- duplicated(grid.rho)))

grid.rho <- grid.rho[!idx]

## splines grid

min.nk <- 2 # minimum number of knots

current.nk <- spar$nk.values[x$range]

grid.nk <- grid\_nk(current.nk, min.nk)

## remove potentially duplicated knot values

if(any(idx <- duplicated(grid.nk)))

grid.nk <- grid.nk[!idx]

ans <- within(x, {

fit.ar <- fitar(dat, grid.rho);

fit.spl <- fitspl(dat, grid.nk)

})

## Ad-hoc corrections for MTP full/half-sibs

if (x$design == "MTP") {

## In these cases, the residuals are accounting for part of the genetic

## effect. Specifically:

## estvar\_a ~ f\*var\_a

## estvar\_e ~ var\_e + (1-f)var\_a

## thus, var\_e = estvar\_e - (1-f)/f\*estvar\_a

## where f = .25 or .5 for half and full sibs respectively.

## This can be extended trivially to clones with f=1.

f <- c("half-sibs" = .25, "full-sibs" = .5, "clones" = 1)[x$plot.str]

fix\_genMTP <- function(x, f) {

estvar\_a <- x$var["genotype", 1]

estvar\_e <- x$var["Residual", 1]

## Correct the estimate of the genetic variance

x$var["genotype", 1] <- estvar\_a/f

## Correct the estimate of the residual variances

stopifnot(

(x$var["Residual", 1] <- estvar\_e - (1-f)/f\*estvar\_a) > 0

)

## Correct the genetic blups

## Need to transform a group-wise random effect to a indinvidual one

prop <- (1-f) \* estvar\_a / (f \* estvar\_e - (1-f) \* estvar\_a)

stopifnot(prop >= 0)

blups\_a <- model.matrix(x)$genotype %\*% ranef(x)$genotype

x$ranef$genotype <- data.frame(value = blups\_a + prop\*resid(x),

"s.e." = NA)

x$effects$genotype$effects[[1]]$incidence.matrix <-

Matrix::Diagonal(length(blups\_a))

return(x)

}

ans$fit.ar <- fix\_genMTP(ans$fit.ar, f)

ans$fit.spl <- fix\_genMTP(ans$fit.spl, f)

}

return(ans)

}

## Fitting simulated datasets with splines and AR models doing grid search of

## knots and autocorrelation parameters respectively

pfitsim <- function(wd = ".", siml.fname, results.fname, ncores) {

library(breedR)

## Auxiliar functions

source(file.path(wd, "src", "functions.R"), local = TRUE)

## Load the simulated datasets

siml <- readRDS(siml.fname)

spar <- attr(siml, "sim\_parameters")

#

# ## Check

# system.time(

# test <- fitsim(siml[[1]], spar)

# ) ## elapsed: 323 sec (~5.4 min) @fmlt

## Setup parallelization

library(foreach)

library(doParallel)

cl <- makeCluster(ncores) # number of cores to use

# cl <- makeCluster("eldorado", user = "fmunoz", type = "PSOCK")

registerDoParallel(cl)

## Go for it!

siml <- foreach(x = iter(siml)) %dopar% fitsim(x, spar)

## Restore attributes

attr(siml, "sim\_parameters") <- spar

## Save

saveRDS(siml, results.fname)

stopCluster(cl)

return(TRUE)

}

## Run fitting job for a given file basename ##

runjob <- function(simfn, ncores, wdir) {

resfn <- gsub("siml", "results", simfn)

cat(paste("\*\*\* Fitting", simfn, "\*\*\*\n"))

pfitsim(wd = wdir,

siml.fname = simfn,

results.fname = resfn,

ncores = ncores - 1)

cat(paste("\*\*\* Saving", resfn, "\*\*\*\n"))

}

#### Extraction of results ####

## Extract metrics and results from fitted models to simulated data

get\_metrics <- function(x) {

t.sv <- attr(x$dat, "vratio") # true spatial variance

t.rv <- c(STP = 1, MTP = 2 / 3)[x$design] # true residual variance

## predicted spatial BLUPs

sp.predict <- function(x)

as.vector(model.matrix(x)$spatial %\*% ranef(x)$spatial)

spl.sp <- sp.predict(x$fit.spl)

ar.sp <- sp.predict(x$fit.ar)

# ## Debug

# data.frame(true = x$dat$spatial,

# splines = spl.sp,

# AR = ar.sp) %>%

# gather(model, pred, -1) %>%

# ggplot(aes(true, pred, col = model)) +

# geom\_point() +

# geom\_abline(intercept = 0, slope = 1)

#

# qplot(spl.sp, ar.sp) + geom\_abline(intercept = 0, slope = 1)

#

# cor(x$dat$spatial, spl.sp) # .60

# cor(x$dat$spatial, ar.sp) # .62

## Root-mean squared error

rmse <- function(x, y) sqrt(mean((x-y)\*\*2))

## relative error

relerr <- function(x, true) (x-true)/true

## Empirical variograms for both approaches

vgs <-

get\_vgs(x) %>%

filter(vg != 'true') %>%

droplevels()

vgfits <-

vgs %>%

split(.$vg) %>%

lapply(vgfit)

slopes <- sapply(vgfits, vgslope)

distest <- sapply(vgfits, vgsill) -

sapply(x[c("fit.ar", "fit.spl")],

function(x) x$var["Residual", "Estimated variances"])

data.frame(

x[c("model", "range", "sp.vratio")],

approach = c("splines", "AR"),

vgslope = slopes[c('splines', 'AR')],

RV\_disc = distest[c('splines', 'AR')],

RVSE = c(NA, x$fit.ar$var["Residual", "S.E."]),

SVRD = c(relerr(x$fit.spl$var["spatial", 1], t.sv),

relerr(x$fit.ar$var["spatial", 1], t.sv)),

RVRD = c(relerr(x$fit.spl$var["Residual", 1], t.rv),

relerr(x$fit.ar$var["Residual", 1], t.rv)),

SCor = c(cor(x$dat$spatial, spl.sp),

cor(x$dat$spatial, ar.sp)),

RMSE = c(rmse(x$dat$phenotype-x$dat$resid, fitted(x$fit.spl)),

rmse(x$dat$phenotype-x$dat$resid, fitted(x$fit.ar))),

AIC = c(extractAIC(x$fit.spl), extractAIC(x$fit.ar)),

logLik = c(logLik(x$fit.spl), logLik(x$fit.ar))

)

}

## Compute quantiles and summary metrics from samples of metrics

summarise\_metrics <- function(x) {

x %>%

## Compute difference splines-AR

group\_by(.id) %>%

spread(approach, value) %>%

mutate(`|SP|-|AR|` = abs(splines) - abs(AR)) %>%

gather(approach, value, AR:`|SP|-|AR|`) %>%

## Compute summary values by scenario, approach and metric

group\_by(model, range, sp.vratio, approach, metric) %>%

summarise(

mean = mean(value, na.rm = TRUE),

sd = sd(value, na.rm = TRUE),

median = median(value, na.rm = TRUE),

q05 = quantile(value, probs = .05, na.rm = TRUE),

q25 = quantile(value, probs = .25, na.rm = TRUE),

q75 = quantile(value, probs = .75, na.rm = TRUE),

q95 = quantile(value, probs = .95, na.rm = TRUE)

) %>%

mutate(liminf = mean-sd, limsup = mean + sd)

}

## Variograms from fitted models

get\_vgs <- function(x) {

true\_vg <- variogram(

coord = x$dat[, c("Var1", "Var2")],

z = x$dat$spatial + x$dat$resid)[["isotropic"]]

ar\_vg <- variogram(x$fit.ar)[["isotropic"]]

spl\_vg <- variogram(x$fit.spl)[["isotropic"]]

cbind(model = x$model,

range = x$range,

sp.vratio = x$sp.vratio,

rbind(

cbind(vg = "true", true\_vg),

cbind(vg = "AR", ar\_vg),

cbind(vg = "splines", spl\_vg)

)

)

}

## Exponential variogram function

## different parameterizations:

## gamma = nugget + psill \* ( 1 - exp( -x / phi ) )

## (too much correlation between nugget and psill)

## = fsill - psill \* exp( -x / phi )

## theta = c(psill, phi, full-sill)

expf <- function(x, theta) {

names(theta) <- c("psill", "phi", "fsill")

theta["fsill"] - theta["psill"] \* exp ( - x / theta["phi"])

}

## Fit an exponential variogram model with possibly negative values

## of nugget and partial sill

vgfit <- function(vg, hessian = FALSE) {

## Loss function

## sum of squared errors

## theta = c(partial-sill, phi, full-sill)

loss.fn <- function(theta, g.l) {

gammaU <- expf(g.l$distance, theta)

loss <- sum((g.l$variogram - gammaU)^2)

if (loss > (.Machine$double.xmax^0.5) | loss == Inf | loss == -Inf | is.nan(loss))

loss <- .Machine$double.xmax^0.5

return(loss)

}

## range ~ 3 phi

## set 1 <= range ~ 3phi -> phi > .34

## set .6 > slope = psill/phi -> psill < .6\*.34 < .6 phi

fit <- optim(par = c(0, 1, 0), loss.fn, method = "L-BFGS-B",

hessian = hessian, lower = c(-.2, .34, -5), upper = c(.2, 30, 5),

g.l = vg)

if (fit$convergence) {

## The variogram fit did not converge

## because it is almost linear

## Fix phi = 1. Then, the full-sill is the intercept

## and partial-sill is the slope of a linear regresion

fit <- list(

par = unname(

c(1, coef(lm(vg$variogram ~ vg$distance)))[c(3, 1, 2)])

)

}

return(fit)

}

## Approximate slope at 0 for an empirical variogram

## fit is the output from vgfit

vgslope <- function(fit) {

par <- fit$par

par[1]/par[2]

}

## (full) sill for an empirical variogram

## fit is the output from vgfit

vgsill <- function(fit) {

par <- fit$par

par[3]

}

## Sample n.sample simulated datasets from each scenario with high sp.vratio

## and one simulated dataset from each of the remaining scenarios

## total = 4\*n.sample + 8

samplesim <- function(siml, n.sample) {

## Examples of simulated data by spatial range and model

## Take 6 out of the 60 replicas from the first set

idx.spl <- list(

short = sample(whichsim("splines", "short", "high", siml), n.sample),

long = sample(whichsim("splines", "long", "high", siml), n.sample))

idx.ar1 <- list(

short = sample(whichsim("AR", "short", "high", siml), n.sample),

long = sample(whichsim("AR", "long", "high", siml), n.sample))

## Take also one example of simulated data from each

## of the remaining scenarios (i.e. sp.variance low and med)

idx.all <- 1:8

return(sort(unlist(c(idx.all, idx.spl, idx.ar1))))

}

## Digest cached results

## Sample six simulated datasets from each scenario with high sp.vratio

## and one simulated dataset from each of the remaining scenarios

## Return in data.frame format

## @param n.sample sample size under the high sp.vratio scenario (default 6)

digest\_sampledat <- function(fname, n.sample = 6) {

## Load results

siml <- readRDS(fname)

## Use names for identifying replicas

names(siml) <- seq\_along(siml)

## Simulation parameters

spar <- attr(siml, "sim\_parameters")

simdat <-

siml[samplesim(siml, n.sample)] %>%

ldply(function(x)

cbind(model = x$model,

range = x$range,

design = x$design,

plot.str = x$plot.str,

sp.vratio = x$sp.vratio,

x$dat)) %>%

group\_by(range) %>%

mutate(rep = dense\_rank(.id))

attr(simdat, "sim\_parameters") <- spar

return(simdat)

}

## Synthesize fitted models from a list of cached results files

## and return a data.frame of the result of applying a function.

## Read results files sequentially to avoid memory issues.

digest\_apply <- function(fnames, fun, sample = FALSE) {

## Load and Process

process\_file <- function(fn) {

siml <- readRDS(fn)

names(siml) <-

paste(seq\_along(siml),

strsplit(basename(fn), split = "-")[[1]][1],

sep = "-")

if(sample) {

siml <- siml[samplesim(siml, sample)]

}

ldply(siml, fun, .id = ".id")

}

res <- ldply(fnames, process\_file)

return(res)

}

digest\_variograms <- function(fnames) {

}

#### Plotting functions ####

## Plot the spatial effect of a simulated dataset

plotsp <- function(x) {

ggplot(x, aes(Var1, Var2, fill = spatial)) +

geom\_raster() + coord\_fixed() + scale\_fill\_viridis()

}

## Plot a empirical variogram from a data frame with variables `distance` and

## `variogram`

plotvg <- function(dat, smooth = TRUE, size = 1.5) {

p <-

ggplot(dat, aes(distance, variogram)) +

geom\_point(size = size) +

geom\_line(size = size / 3) +

labs(y = "semivariance")

if (smooth)

p <- p + stat\_smooth(se = FALSE, method = "loess", size = size \* 2 / 3)

return(p)

}

## ggplot2 proto object for the fitted exponential variogram

StatExp <- ggproto("StatExp", Stat,

required\_aes = c("x", "y"),

compute\_group = function(data, scales, n = 51) {

fit <- vgfit(data.frame(distance = data$x, variogram = data$y))

rng <- range(data$x, na.rm = TRUE)

x <- exp(seq(log(rng[1]), log(rng[2]), length.out = n))

data.frame(x = x,

y = expf(x, fit$par))

}

)

## ggplot2 stat object for the fitted exponential variogram

stat\_exp <- function(mapping = NULL, data = NULL, geom = "line",

position = "identity", na.rm = FALSE, show.legend = NA,

inherit.aes = TRUE, n = 51, ...) {

layer(

stat = StatExp, data = data, mapping = mapping, geom = geom,

position = position, show.legend = show.legend, inherit.aes = inherit.aes,

params = list(n = n, na.rm = na.rm, ...)

)

}

## Scenario-facetted plot for a given metric x

## Can't put everything into one single plot because

## of different scales accross metrics

scatter\_metric <- function(x, dat) {

guide\_intercept <-

c(AIC = NA,

RMSE = NA,

RV\_disc = 0,

RVRD = 0,

SCor = 1,

SVRD = 0,

vgslope = 0)

p <- dat %>%

group\_by(.id) %>%

spread(approach, value) %>%

filter(metric == x) %>%

ggplot(aes(AR, splines)) +

geom\_hline(yintercept = guide\_intercept[x], col = "gray") +

geom\_vline(xintercept = guide\_intercept[x], col = "gray") +

geom\_abline(intercept = 0, slope = 1, col = "darkgray") +

geom\_point(aes(col = model)) +

facet\_grid(metric ~ range + sp.vratio, scales = "free") +

labs(x = NULL, y = NULL) +

theme(

plot.margin = unit(c(0,0,1,1), units = "lines"),

axis.text.y = element\_text(angle=90))

## Facet strips only for the first metric (AIC)

if (x != "AIC")

p <- p +

theme(strip.text.x = element\_blank())

## "Remove" redundant legends

## but keep the spacing by writing them in white

if (x != "RVRD")

p <- p +

theme(

legend.text = element\_text(colour = "white"),

legend.title = element\_text(colour = "white"),

legend.key = element\_rect(colour = "white")

) +

guides(colour = guide\_legend(

override.aes = list(alpha = 0)

))

p

}

## Plot scatterplot matrix of relationships among metrics

## Note: this function takes the range and design from the chunk label

## assuming a format like "something-[else]-range-design"

plot\_corr\_metrics <- function(label = opts\_current$get("label"),

spvr = 'high',

columns = 6:13) {

lab\_components <- strsplit(label, "-")[[1]]

scen <- structure(tail(lab\_components, 2),

names = c("range", "design"))

dat <- paste("res", scen["design"], sep = "\_")

eval(parse(text = dat)) %>%

# filter(metric != "RVSE") %>%

spread(metric, value) %>%

mutate(group = factor(model:approach)) %>%

# filter(model == 'AR', range == 'short', sp.vratio == 'high') %>%

filter(

range == scen["range"],

sp.vratio == spvr) %>%

ggpairs(aes(col = group), columns = columns)

}

## Save a plot for publication

papersave <- function(

fbname, # file basename (no extension)

plot = last\_plot(), # plot to save

device = c("pdf", "jpeg"), # one or more formats

dir = file.path("paper/fig"), # target dir

wd = 1, # in "columns" units

ht = 1, # in "rows" units (roughly)

caption = NULL # caption to be saved in a txt.file

) {

stopifnot(!missing(fbname))

stopifnot(as.logical(length(device)))

plot <- plot + theme\_bw(base\_size = 9)

wl <- c(3.465, 7.205)

hl <- c(2.5, 3.465, 5, 10)

fn <- paste(fbname, device, sep = ".")

for (f in fn) {

ggsave(f, plot, path = dir,

width = wl[wd], height = hl[ht], dpi = 600)

}

## Caption

if (!is.null(caption)) {

cn <- paste(fbname, ".txt")

writeLines(caption, file.path(dir, cn))

}

invisible(TRUE)

}

#### System functions ####

## Return the maximum available number of cores

sys.ncores <- function() {

cmd.ncores <- "cat /proc/cpuinfo | grep '^processor' | wc -l"

cmd.load <- "uptime | awk '{gsub(/.\*:/,\"\"); {gsub(/, /,\" \")}; print $1}'"

num.cores <- system(paste(cmd.ncores), intern = TRUE)

load <- gsub(",", ".", # substitute , by . if necessary

system(paste(cmd.load), intern = TRUE))

return(as.integer(num.cores) - ceiling(as.numeric(load)))

}

## 2. Simulation of datasets under different scenarios

# ## checks

# str(simsp(dim=c(4,4), sp.model = 'AR', sp.param = c(.9, .9), sp.vratio = 1))

# str(simsp(dim=c(4,4), sp.model = 'splines', sp.param = c(2, 2), sp.vratio = 1))

# simparam(sp.model = 'AR', range = 'short', sp.vratio = 'med') %>% str()

# simparam(sp.model = 'splines', range = 'long', sp.vratio = 'high') %>% str()

## Simulation parameters ##

dim <- c(70, 70)

def.nk <- sapply(dim, breedR:::determine.n.knots) # 7

spar <- list(

n.files = 6,

N.replicates = 6\*30,

dimensions = dim,

design = c('STP', 'MTP'),

plot.str = c('clones', 'full-sibs', 'half-sibs'),

models = c('AR', 'splines'),

ranges = c('short', 'long'),

sp.vratios = c('low', 'med', 'high'),

rho.values = c(short = .8, long = .93),

nk.values = c(short = 20, long = 7),

vratio.values= structure(rev(1/2^2^(0:2)),

names = c('low', 'med', 'high'))

)

## In order to keep objects within manegeable sizes we split the simulations of

## STP-clones, MTP-clones, MTP-fullsibs and MTP-halfsibs into different

## simulated lists. Furthermore, we split the 6\*30 = 180 replicates per scenario

## into 6 separate files.

## Setup parallelization

library(foreach)

library(doParallel)

source("functions.R")

work.dir <- normalizePath(".")

dir.create(file.path(work.dir, "cache"))

cl <- makeCluster(sys.ncores()) # number of cores to use

# cl <- makeCluster("eldorado", user = 'fmunoz', type = "PSOCK")

registerDoParallel(cl)

spar$design <- 'STP'

spar$plot.str <- 'clones'

foreach(x = seq\_len(spar$n.files)) %dopar% simlist(spar, get\_cached(paste0('siml', x, '-STP')))

spar$design <- 'MTP'

spar$plot.str <- 'clones'

foreach(x = seq\_len(spar$n.files)) %dopar% simlist(spar, get\_cached(paste0('siml', x, '-MTPclones')))

spar$design <- 'MTP'

spar$plot.str <- 'full-sibs'

foreach(x = seq\_len(spar$n.files)) %dopar% simlist(spar, get\_cached(paste0('siml', x, '-MTPfull')))

spar$design <- 'MTP'

spar$plot.str <- 'half-sibs'

foreach(x = seq\_len(spar$n.files)) %dopar% simlist(spar, get\_cached(paste0('siml', x, '-MTPhalf')))

**R\_code S2.** R code to generate a reduced simulation study.

#Copyright 2018 Facundo Muñoz.

#This code is free software: you can redistribute it and/or modify

#it under the terms of the GNU General Public License as published by

#the Free Software Foundation, either version 3 of the License, or

#(at your option) any later version.

#This program is distributed in the hope that it will be useful,

#but WITHOUT ANY WARRANTY; without even the implied warranty of

#MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

#GNU General Public License for more details.

#You should have received a copy of the GNU General Public License

#along with this program. If not, see <https://www.gnu.org/licenses/>

## 1. Dataset simulation functions ####

## Please, used the R code provided in the Electronic Supplementary Material R\_code S1.

## 2. Simulation of reduced datasets and summarized results ####

library(breedR)

library(dplyr)

library(tidyr)

source("functions.R")

## Setup simulation study parameters

dim <- c(30, 30)

def.nk <- sapply(dim, breedR:::determine.n.knots) # 6

spar <- list(

n.files = 1,

N.replicates = 1,

dimensions = dim,

design = c('STP'),

plot.str = c('clones'),

models = c('AR', 'splines'),

ranges = c('short'),

sp.vratios = c('high'),

rho.values = c(short = .8, long = .93),

nk.values = c(short = 20, long = 7),

vratio.values= structure(rev(1/2^2^(0:2)),

names = c('low', 'med', 'high'))

)

## Simulate N.replicates datasets for all combination of parameters

## above. Store in one file under 'cache'

simlist(spar, 'cache/siml-STP.rds')

## Fit models (AR and splines, conduct grid search)

## pfitsim() expects function.R to be under /src

dir.create("src")

file.copy("functions.R", "src")

pfitsim(wd = ".",

siml.fname = "cache/siml-STP.rds",

results.fname = "cache/results-STP.rds",

ncores = 6)

res <- readRDS("cache/results-STP.rds")

metrics <- lapply(res, get\_metrics)

**Table S1.** Akaike Information Criterion (AIC), estimates of variance components (Genetics, Spatial, Residual), heritabilities (Heritability), and average of the correlation coefficients and root mean square errors (RMSE) from the cross-validation analyses for each of the three spatial mixed models studied (see text for models’ abbreviations). Standard errors are in brackets. The corresponding traits analyzed are total height (TH) for the sites 2.707.3, 2.708.1, 3.704.2, 3.713.1, and 3.713.3, and diameter at breast height (DAP) for the sites 2.701.1, 3.713.1, 3.713.2, and 3.713.3. The lowest AIC and RMSE values, and highest correlations values are highlighted in bold. Models are described in the text. Trials are complementary to those shown in Table 2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Trial (Trait)** |  | **Model** | | |
|  | **block** | **AR** | **splines** |
| **2.707.1 (DAP)** | **AIC** | 51772 | **51408** | 51593 |
| **Genetic** | 270 (28) | 265 (38) | 260 (78) |
| **Spatial** | 179 (30) | 534 (24) | 628 (31) |
| **Residual** | 1271 (37) | 1009 (26) | 1216 (26) |
| **Heritability** | 0.17 (0.02) | 0.21 (0.02) | 0.18 (0.02) |
| **Correlation coefficients** | 0.26 (0.06) | **0.29 (0.06)** | 0.28 (0.06) |
| **RMSE** | 43.46 (1.9) | **43 (1.99)** | 43.13 (1.98) |
| **2.707.3 (TH)** | **AIC** | 68451 | **68188** | 68223 |
| **Genetic** | 1819 (144) | 1820 (139) | 1796 (226) |
| **Spatial** | 1733 (213) | 1892 (151) | 1951 (130) |
| **Residual** | 4711 (107) | 3952 (108) | 4431 (99) |
| **Heritability** | 0.28 (0.02) | 0.32 (0.02) | 0.29 (0.02) |
| **Correlation coefficients** | 0.46 (0.03) | **0.51 (0.03)** | 0.49 (0.02) |
| **RMSE** | 22.95 (0.56) | **22.37 (0.58)** | 22.52 (0.59) |
| **2.708.1 (TH)** | **AIC** | 80793 | **80499** | 80617 |
| **Genetic** | 2173 (210) | 2170 (179) | 2151 (138) |
| **Spatial** | 662 (128) | 2092 (191) | 1107 (135) |
| **Residual** | 7804 (176) | 6333 (196) | 7449 (141) |
| **Heritability** | 0.22 (0.02) | 0.26 (0.02) | 0.22 (0.01) |
| **Correlation coefficients** | 0.36 (0.05) | **0.43 (0.03)** | 0.4 (0.04) |
| **RMSE** | 28.5 (0.9) | **27.6 (0.67)** | 28.05 (0.82) |
| **3.704.2 (TH)** | **AIC** | **33049** | 33050 | 33053 |
| **Genetic** | 2728 (278) | 2653 (88) | 2656 (87) |
| **Spatial** | 518 (106) | 748 (187) | 484 (211) |
| **Residual** | 6665 (309) | 6518 (170) | 6782 (187) |
| **Heritability** | 0.29 (0.03) | 0.29 (0.02) | 0.28 (0.02) |
| **Correlation coefficients** | **0.28 (0.07)** | 0.28 (0.07) | 0.28 (0.07) |
| **RMSE** | **27.51 (1.7)** | 27.55 (1.61) | 27.56 (1.65) |
| **3.713.1 (TH)** | **AIC** | 48074 | **47826** | 47907 |
| **Genetic** | 2756 (284) | 2834 (345) | 2782 (480) |
| **Spatial** | 1400 (281) | 3066 (179) | 2043 (180) |
| **Residual** | 7978 (265) | 6005 (155) | 7257 (171) |
| **Heritability** | 0.26 (0.02) | 0.32 (0.02) | 0.28 (0.02) |
| **Correlation coefficients** | 0.39 (0.04) | **0.47 (0.04)** | 0.44 (0.04) |
| **RMSE** | 29.07 (1.43) | **27.91 (1.36)** | 28.32 (1.48) |
| **3.713.1 (DAP)** | **AIC** | 47754 | **47697** | 47698 |
| **Genetic** | 6328 (599) | 6515 (219) | 6404 (126) |
| **Spatial** | 734 (161) | 1239 (477) | 912 (507) |
| **Residual** | 10283 (358) | 9694 (308) | 10000 (312) |
| **Heritability** | 0.38 (0.03) | 0.4 (0.02) | 0.39 (0.02) |
| **Correlation coefficients** | 0.33 (0.05) | **0.35 (0.06)** | 0.35 (0.06) |
| **RMSE** | 35.39 (1.23) | **35.13 (1.2)** | 35.14 (1.16) |
| **3.713.2 (DAP)** | **AIC** | 41757 | **41739** | 41746 |
| **Genetic** | 3186 (338) | 3209 (42) | 3190 (51) |
| **Spatial** | 103 (40) | 230 (252) | 148 (280) |
| **Residual** | 7817 (276) | 7693 (164) | 7778 (220) |
| **Heritability** | 0.29 (0.03) | 0.29 (0.02) | 0.29 (0.02) |
| **Correlation coefficients** | 0.3 (0.05) | **0.31 (0.05)** | 0.31 (0.05) |
| **RMSE** | 29.15 (1.94) | **29.05 (1.95)** | 29.1 (1.94) |
| **3.713.3 (TH)** | **AIC** | 17336 | 17302 | **17301** |
| **Genetic** | 972 (254) | 997 (176) | 1009 (319) |
| **Spatial** | 608 (165) | 1493 (192) | 1271 (212) |
| **Residual** | 7752 (365) | 6849 (167) | 7015 (132) |
| **Heritability** | 0.11 (0.03) | 0.13 (0.02) | 0.13 (0.02) |
| **Correlation coefficients** | 0.26 (0.11) | 0.31 (0.09) | **0.31 (0.08)** |
| **RMSE** | 26.94 (1.47) | 26.48 (2.37) | **26.45 (2.17)** |
| **3.713.3 (C)** | **AIC** | 17882 | 17862 | **17858** |
| **Genetic** | 5477 (1403) | 5484 (251) | 5465 (278) |
| **Spatial** | 1107 (317) | 1223 (997) | 1407 (1171) |
| **Residual** | 16235 (978) | 16060 (628) | 16120 (734) |
| **Heritability** | 0.25 (0.05) | 0.25 (0.04) | 0.25 (0.05) |
| **Correlation coefficients** | 0.29 (0.06) | 0.31 (0.07) | **0.32 (0.07)** |
| **RMSE** | 41.65 (2.51) | 41.38 (2.76) | **41.3 (2.75)** |