MEMgene effect size analysis

Contents

1. Introduction

This file documents the analysis of simulated datasets from Lotterhos & Whitlock (2015) with the same random sample of 90 sites for all datasets.

The analysis is based on the subsets of 20 or 6 individuals per sampling location that were exported by Lotterhos & Whitlock (2015), i.e., the samples with n=6 individuals are not sampled.

Data sources:

- The genetic data from Lotterhos and Whitlock (2015) are available on Dryad: Lotterhos, Katie E.; Whitlock, Michael C. (2015), Data from: The relative power of genome scans to detect local adaptation depends on sampling design and statistical method, Dryad, Dataset, https://doi.org/10.5061/dryad.mh67v
- The above link will download an archive `dryad.zip`, which contains two folders. The files needed here are in folder `SimFilesLFMM`.
- The file `SchemeRandom1.txt` with the spatial coordinates of the 90 random sampling sites is also available on Dryad: Wagner, Helene H.; Chávez-Pesqueira, Mariana; Forester, Brenna R. (2017), Data from: Spatial detection of outlier loci with Moran eigenvector maps (MEM), Dryad, Dataset, https://doi.org/10.5061/dryad.b12kk

2. Preparations

Load packages (more are loaded later for producing figures)

```
library(hierfstat)
```

```
## Registered S3 method overwritten by 'spdep':
##   method   from
##   plot.mst ape
```

```
library(memgene)
library(here)
```

```
## here() starts at /Users/helene/OneDrive - University of Toronto/R/MEMgene_paper
```

```
library(parallel)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(spdep)
```

```
## Loading required package: sp
```

```
## Loading required package: spData
```

```
## Loading required package: sf
```

```
## Linking to GEOS 3.7.2, GDAL 2.4.2, PROJ 5.2.0
```

a) Import spatial coordinates

```
Coords <- list()
Coords$Pairs <- list()
Coords$Transect <- list()
Coords$Random <- list()
Coords$Random$E453 <- Coords$Random$E988 <- Coords$Random$E950 <-
  read.table("SchemeRandom1.txt") # upload this file

for(i in 1:length(Coords))
{
  if(length(Coords[[i]]) > 0)
  {
    for(k in 1:length(Coords[[i]]))
    {
      b <- order(Coords[[i]][[k]][,3], Coords[[i]][[k]][,2]) # Sort by y, then x
      Coords[[i]][[k]] <- Coords[[i]][[k]][b,]  # correct order!!
    }
  }
}
```

b) Parameter space

Genetic data: find all file names in the folder "SimFilesLFMM" that contain 'lfmm':

```
Filenames.lfmm <- list.files(paste0(here::here(), "/dryad//SimFilesLFMM/"), pattern="lfmm")

test <- Reduce(rbind, strsplit(Filenames.lfmm, split="[_=]"))
test <- cbind(test, Reduce(rbind, strsplit(test[,ncol(test)], split="[.]")))
tmp <- strsplit(test[,2], split="[.xs]")
test2 <- matrix(NA, nrow(test), 3)
for(i in 1:nrow(test)) test2[i,1:length(tmp[[i]])] <- tmp[[i]]
test2 <- gsub("[A-Z, a-z]","", test2)
test <- cbind(test, test2)
dimnames(test) <- list(NULL, c("Demography", "Design", "ID", "Env", "ID2",
                               "V6", "NumPops", "V8", "V9", "NumInd", "Design2",
                               "NumPops2", "NumTrans", "NumInd2"))
```

Design matrix (parameter space): each row is one combination of parameter settings:

- Demography: single refugium (1R), two refugia (2R), isolation by distane (IBD), island model (IM)
- Design: sampling design (R, P, T; see below) and number of pops sampled
- Env: which of the three replicate landscapes '453', '950', '988'
- NumPops: how many populations are sampled
- NumInd: how many individuals sampled per pop
- Type: random (R), pairs (P), transects (T)

```
Design <- as.data.frame(test[,c(1,2,4,7,10,13,14)])
Design$Env <- ordered(Design$Env, levels=c(453,988,950))
Design$Type <- ordered(substr(Design$Design, 1, 1), levels=c("P", "T", "R"))
Design$Design <- as.character(Design$Design)
Design$Design[Design$Design == "T30.T3x10"] <- "T30.3x10s"
Design$Design[Design$Design == "T30.T6x5s"] <- "T30.6x5s"
Design$NumPops <- as.numeric(as.character(Design$NumPops))
```

c) Create replicate random samples of 30 or 60 sites

R subsamples of n = 30 sites

```
R = 10
Sites.30 <- list()
set.seed(19)
for(r in 1:R)
{
  Sites.30[[r]] <- sort(sample(1:90, 30, replace=FALSE))
}

saveRDS(Sites.30, paste0(here::here(), "/output/Sites.30.rds"))
```

R subsamples of n = 60 sites

```
R = 10
Results.drop <- rep( list(list()), R)

set.seed(297)
Drop <-lapply(c(1:R), function(r) sort(sample(1:90, 30, replace=FALSE)))

saveRDS(Drop, paste0(here::here(), "/output/Drop.rds"))
```

3. Genetic data

a) Select datasets

Select all data files with the largest sample size (90) and random sampling (R)

```
Sites.R.90 <- c(1:nrow(Design))[Design$NumPops==90 & Design$Type=="R"]
```

Check parameters for selected datasets:

```
Design[Sites.R.90,]
```

```
##     Demography Design Env NumPops NumInd NumTrans NumInd2 Type
## 32          1R    R90 453      90     20     <NA>    <NA>    R
## 33          1R    R90 453      90      6     <NA>    <NA>    R
## 34          1R    R90 988      90     20     <NA>    <NA>    R
## 35          1R    R90 988      90      6     <NA>    <NA>    R
## 36          1R    R90 950      90     20     <NA>    <NA>    R
## 37          1R    R90 950      90      6     <NA>    <NA>    R
## 93          2R    R90 453      90     20     <NA>    <NA>    R
## 94          2R    R90 453      90      6     <NA>    <NA>    R
## 95          2R    R90 988      90     20     <NA>    <NA>    R
## 96          2R    R90 988      90      6     <NA>    <NA>    R
## 97          2R    R90 950      90     20     <NA>    <NA>    R
## 98          2R    R90 950      90      6     <NA>    <NA>    R
## 153        IBD    R90 453      90     20     <NA>    <NA>    R
## 154        IBD    R90 453      90      6     <NA>    <NA>    R
## 155        IBD    R90 988      90     20     <NA>    <NA>    R
## 156        IBD    R90 988      90      6     <NA>    <NA>    R
## 157        IBD    R90 950      90     20     <NA>    <NA>    R
## 158        IBD    R90 950      90      6     <NA>    <NA>    R
## 213         IM    R90 453      90     20     <NA>    <NA>    R
## 214         IM    R90 453      90      6     <NA>    <NA>    R
## 215         IM    R90 988      90     20     <NA>    <NA>    R
## 216         IM    R90 988      90      6     <NA>    <NA>    R
## 217         IM    R90 950      90     20     <NA>    <NA>    R
## 218         IM    R90 950      90      6     <NA>    <NA>    R
```

b) Run function 'getGenData' to extract the genetic data

This function:

- extracts the genetic data,
- selects the 9900 neutral loci,
- randomizes the order of the neutral loci (allows sampling loci),
- adds a first column 'pop' with site as a factor (format for functions in the package `hierfstat`), and
- randomizes the alleles of each SNP within each population (allows sampling individuals, though this was not used in the present study as we worked with the sets of 20 and 6 individuals per deme provided as separate datasets by Lotterhos & Whitlock 2015).

Define function:

```
getGenData <- function(j)
{
  # Select the sites that need to be sampled for this run:
  i=Sites.R.90[j]
  cat("j:", j, ", i:", i, "\n")

  # Each file has NumPops x NumInd rows (sampled individuals) and up to 10000 columns (loci)
  tmp <- read.table(paste0(here::here(),"/dryad/SimFilesLFMM/", Filenames.lfmm[[i]]))

  # Drop non-neutral loci
```

```
  tmp <- tmp[,1:9900]

  # Randomize order of neutral loci
  tmp <- tmp[,sample(1:9900)]

  # Site: create vector of sites = pops (for each row = individual)
  Site <- rep(1:Design$NumPops[i],each=as.numeric(as.vector(Design$NumInd)[i]))

  #Data.hierfstat <- data.frame(pop=factor(Site), tmp)
  Data.hierfstat <- data.frame(pop=Site, tmp)

  # NEW Dec 2020: Randomize alleles within and pops (will allow subsampling individuals)
  Data.hierfstat <- Reduce(rbind, lapply(split(Data.hierfstat, Site),
                                         function(p) data.frame(sapply(p , sample))))

  return(Data.hierfstat)
}
```

Run in parallel and save results:

```
start_time <- Sys.time()

Index.j <- 1:length(Sites.R.90)

Data.R.90 <- mclapply(Index.j, function(j) getGenData(j),
                      mc.cores=detectCores())
names(Data.R.90) <- Sites.R.90

end_time <- Sys.time()
end_time - start_time
cat("This job took", (end_time - start_time)/length(Index.j), "per dataset.")

saveRDS(Data.R.90, paste0(here::here(), "/output/Data.R.90.rds"))
```

Results are stored in a list (one element per site j with 90 samples), with each element:

- Data.frame with first column "pop" (factor of site IDs) and 9900 columns of neutral loci

c) Run function 'getDgen' to calculate sample Fst and genetic distances

Define function:

```
#Data.R.90 <- readRDS(paste0(here::here(), "/output/Data.R.90.rds"))

getDgen <- function(j, Data=Data.R.90, dist.method = c("Fst", "Dch"), nLoci=9900, sim=FALSE)
{
  # Randomize pop if sim=TRUE
  if(sim==TRUE) {Data = lapply(Data, function(ls) data.frame(pop=sample(ls$pop), ls[,-1]))}

  Fst <- basic.stats(Data[[j]][,1:(nLoci+1)])$overall

  Fst.30 <- t(sapply(1:length(Sites.30), function(s)
```

```
    basic.stats(filter(Data[[j]][, 1:(nLoci+1)], is.element(pop, Sites.30[[s]])))$overall))

  D <- list()
  for(k in 1:length(dist.method))
  {
    D[[k]] <- genet.dist(Data[[j]][,1:(nLoci+1)], method = dist.method[k])
  }
  names(D) <- dist.method

  return(list(Fst=Fst, Fst.30 = Fst.30, Dgen=D))
}
```

Run for different numbers of loci, create list object

```
loci <- c(9900, 3300, 500, 100, 50)

Dgen.R.90 <- list()

for(s in length(loci):1)
{
  cat(s)
  Dgen.R.90[[s]] <- mclapply(Index.j, function(j) getDgen(j, Data=Data.R.90, nLoci=loci[s],
                                                          sim=FALSE, dist.method=c("Fst", "Dch")),
                  mc.cores=detectCores())
  names(Dgen.R.90[[s]]) <- Sites.R.90
}
names(Dgen.R.90) <- paste0("L", loci)

saveRDS(Dgen.R.90, paste0(here::here(), "/output/Dgen.R.90.rds"))
```

Dgen.R.90 is a list, with one element per number of loci (e.g.: 9900, 3300, 500, 100, 50). Each list element is again a list with one element per site j with 90 samples. Each of these contains the following:

- One list element per number of loci
  - One list element per dataset (site j with 90 samples)
    - Fst: vector with 'overall' statistics returned by `basic.stats` function
    - Fst.30: matrix with statistics for ten subsets of n = 30 sites
    - Dgen: list of pairwise genetic distance matrices
      - Fst (Fst)
      - Cavalli-Sforza and Edwards Chord distance (Dch)

d) Create randomized data (for standardized effect size)

This takes long! Only run for 500 loci and for Fst distance method.

```
R = 200

Dgen.sim <- list()

for(r in 1:R)
{
```

```
  cat("\n", r, ": ")
  Dgen <- list(list(), list(), list(), list(), list())
  names(Dgen) <- paste0("L", loci)
  #for(s in length(loci):1)
  for(s in c(3))
  {
    cat(s)
    Dgen[[s]] <- mclapply(Index.j, function(j) getDgen(j, Data=Data.R.90, nLoci=loci[s],
                                                       sim=TRUE, dist.method=c("Fst")),
                          mc.cores=detectCores())
    names(Dgen[[s]]) <- Sites.R.90
  }
  Dgen.sim[[r]] <- Dgen
}

saveRDS(Dgen.sim, paste0(here::here(), "/output/Dgen.sim.rds"))
```

e) Run function 'getMEMgene' to obtain RsqAdj and Moran's I

This function extracts the grid coordinates, performs MEMgene analysis with forward selection to obtain the adjusted Rsquared value. In addition, it derives the unadjusted Rsquared value using the same set of selected MEM eigenvectors. It also calculates Moran's I for the genetic data by deriving a scalogram S (Rsquared for each MEM vector, which sum to 1 across all MEM vectors) and multiplying it with the rescaled MEM eigenvectors (Moran's I for each vector). Finally, it returns the limits for Moran's I of the genetic data (min and max of Moran's I values of MEM vectors).

Argument 'subset' specifies a subset of sites to be used (meta-analysis mode, performing MEM with subset only). Argument 'drop' specifies which sites should be dropped (for comparative mode, keeping MEM of full dataset).

Define function:

```
getMEMgene <- function(j, Dgen = Dgen.R.90[[1]], subset=NULL, drop=NULL)
{
  i=Sites.R.90[j]
  cat("j:", j, ", i:", i, "\n")

  # Extract the grid coordinates of the sampled sites
  coord <- data.matrix(Coords[[as.numeric(Design$Type[i])]]
                       [[as.numeric(Design$Env[i])]][,2:3])
  if(length(subset) > 0) {coord <- coord[subset,]}
  if(length(drop) > 0) { coord.drop <- coord[-drop,]}

  Res <- list()
  for(k in 1: length(Dgen[[j]]$Dgen))
  {

    Y <- as.matrix(Dgen[[j]]$Dgen[[k]])
    if(length(drop) > 0) {Y = Y[-drop, -drop]}
    if(length(subset) > 0) {Y <- Y[subset, subset]}

    # memGene
```

```
MEM <- mgMEM(dist(coord))

if(length(drop) > 0) {MEM$vectorsMEM <- MEM$vectorsMEM[-drop,]}

Positive <- mgForward(Y, MEM$vectorsMEM[, MEM$valuesMEM > 0])

RsqAdjPos = 0
RsqPos = 0
if(!is.na(Positive$selectedRsqAdj))
{
  RsqAdjPos = Positive$selectedRsqAdj
  # Get R square (unadjusted)
  MEM$analysis <- mgRDA(Y, MEM$vectorsMEM[, Positive$selectedMEM], full=TRUE)
  RsqPos <- sum(diag(MEM$analysis$pred)) / (sum(diag(MEM$analysis$pred)) +
                                            sum(diag(MEM$analysis$resid)))
}

if(length(drop) > 0)
{
  MEM.drop <- mgMEM(dist(coord.drop))

  Positive <- mgForward(Y, MEM.drop$vectorsMEM[ , MEM.drop$valuesMEM > 0])
  RsqAdjPos.drop = 0
  if(!is.na(Positive$selectedRsqAdj)) { RsqAdjPos.drop = Positive$selectedRsqAdj}
}

# Centre distance matrix
n <- nrow(Y)
row.wt = rep(1, nrow(Y))
col.wt = rep(1, ncol(Y))
st <- sum(col.wt)
sr <- sum(row.wt)
row.wt <- row.wt/sr
col.wt <- col.wt/st
Y <- -0.5 * (Y * Y)
row.mean <- apply(row.wt * Y, 2, sum)
col.mean <- apply(col.wt * t(Y), 2, sum)
col.mean <- col.mean - sum(row.mean * col.wt)
Y <- sweep(Y, 2, row.mean)
G <- t(sweep(t(Y), 2, col.mean))

# Get Rsq for each MEM vector from a separate dbRDA for each vector m
X <- MEM$vectorsMEM

S <- rep(0, ncol(X))
for(m in 1:ncol(X))
{
  if(var(X[,m]) > 0)
  {
    p = 1
    H <- X[,m] %*% solve(t(X[,m]) %*% X[,m]) %*% t(X[,m])
    I <- diag(n)
    res <- (I - H) %*% G %*% (I - H)
```

```
        S[m] <- 1 - sum(diag(res))/sum(diag(G))
      }
    }

    S.tot = sum(S)
    S.min = min(S)
    S[S < 0] <- 0

    Values <- MEM$valuesMEM / abs(sum(MEM$valuesMEM))
    Range <- range(Values)
    Morans.I <- as.vector(S %*% Values)

    Res[[k]] <- list(RsqAdjPos=RsqAdjPos, RsqPos=RsqPos, Morans.I=Morans.I, Range=Range)
  }
  names(Res) <- names(Dgen[[1]]$Dgen)

  return(Res)
}
```

Run across all levels of number of loci

```
#Dgen.R.90 <- readRDS(paste0(here::here(), "/output/Dgen.R.90.rds"))

start_time <- Sys.time()

Index.j <- 1:length(Sites.R.90)

Results.R.90 <- list()
for(s in length(loci):1)
{
  cat(s)
  Results.R.90[[s]] <- mclapply(Index.j, function(j)
    getMEMgene(j, Dgen = Dgen.R.90[[s]], subset=NULL),
    mc.cores=detectCores())
  names(Results.R.90[[s]]) <- Sites.R.90
}
names(Results.R.90) <- paste0("L", loci)

end_time <- Sys.time()
end_time - start_time
cat("This job took", (end_time - start_time)/length(Index.j)/length(loci), "per dataset.")

saveRDS(Results.R.90, paste0(here::here(), "/output/Results.R.90.rds"))
```

Results.R.90 is a list, with one element per number of loci (e.g.: 9900, 3300, 500, 100, 50). Each list element is again a list with one element per site j with 90 samples. Each of these contains the following:

- One list element per number of loci
  - One list element per dataset (site j with 90 samples)
    - One list element per genetic distance measure
      - RsqAdjPos: adjusted Rsquare from memgene, based on MEM with positive eigenvalues
      - Morans.I: Moran's I for the genetic data

- Range: range (minimum and maximum) of Moran's I of MEM vectors (limits for Moran's I of genetic data)

Run for simulated data (only for s = 3, i.e., 500 loci)

```
#Dgen.sim <- readRDS(paste0(here::here(), "/output/Dgen.sim.rds"))

start_time <- Sys.time()

Index.j <- 1:length(Sites.R.90)

Results.sim <- list()
#for(r in 1:length(Dgen.sim))
for(r in 1:length(Dgen.sim))
{
  cat(r, " ")
  Results.sim[[r]] <- list()

  s = 3

  Results.sim[[r]] <- mclapply(Index.j, function(j)
                               getMEMgene(j, Dgen = Dgen.sim[[r]][[s]], subset=NULL),
                               mc.cores=detectCores())
  names(Results.sim[[r]]) <- Sites.R.90
}

end_time <- Sys.time()
end_time - start_time
cat("This job took", (end_time - start_time)/length(Index.j)/length(loci)/R, "per dataset.")

saveRDS(Results.sim, paste0(here::here(), "/output/Results.sim.rds"))
```

4. Compile results for n = 90

```
Results.R.90 <- readRDS(paste0(here::here(), "/output/Results.R.90.rds"))
Results.sim <- readRDS(paste0(here::here(), "/output/Results.sim.rds"))
Dgen.R.90 <- readRDS(paste0(here::here(), "/output/Dgen.R.90.rds"))
```

a) Function to extract response variables

```
getRes <- function(Results=Results.R.90[[1]], k = 1)
{
  RsqAdjPos = sapply(Results, function(ls) ls[[k]]$RsqAdjPos)
  RsqPos = sapply(Results, function(ls) ls[[k]]$RsqPos)
  Morans.I = sapply(Results, function(ls) ls[[k]]$Morans.I)
  I.max = sapply(Results, function(ls) max(ls[[k]]$Range))
  I.min = sapply(Results, function(ls) min(ls[[k]]$Range))
  I.scaled = Morans.I / I.max
  res <- data.frame(RsqAdjPos=RsqAdjPos, RsqPos=RsqPos, Morans.I=Morans.I, I.scaled=I.scaled)
}
```

b) Combine response variables with Design matrix

```
Results.table <- list()

for(s in length(loci):1)
{
  cat(s)
  Ftab <- data.frame(Fst.90 = sapply(Dgen.R.90[[s]], function(ls) ls$Fst[7]),
                     Fst.30.m = sapply(Dgen.R.90[[s]], function(ls) mean(ls$Fst.30[,7])),
                     Fst.30.s = sapply(Dgen.R.90[[s]], function(ls) sd(ls$Fst.30[,7])))

  res <- lapply(c(1:length(Results.R.90[[s]][[1]])), function(k) getRes(Results.R.90[[s]], k))
  names(res) <- names(Results.R.90[[s]][[1]])

  res.combined <- Reduce(cbind, res)
  names(res.combined) <- paste(rep(names(Results.R.90[[s]][[1]]), each=ncol(res[[1]])),
                               names(res[[1]]), "90", sep=".")
  Results.table[[s]] <- data.frame(Design[Sites.R.90,], Ftab, res.combined)
}
names(Results.table) <- paste0("L", loci)

saveRDS(Results.table, paste0(here::here(), "/output/Results.table.rds"))
```

Analyze according to the following factors:

- Demography (1R, 2R, IBD, IM)
- NumInd: 20, 6
- Genetic distance measure (Fst, Dch)
- Number of loci (9900, 3300, 500, 100, 50)

Response variables:

- RsqAdjPos: as RsqAdj in memgene output with default settings (only positive MEM)
- RsqPos: unadjusted (not reported in memgene)
- Morans.I: (not reported in memgene) assuming population is sampled. Correct with (n-1)/n (where n = 90) for estimating population Moran's I
- Morans.I.scaled: this is experimental. Divide Morans.I by the maximum value (max(Range)).

Notes:

- Env: for each combination, there are three replicate datasets (Env: 453, 988, 950). Env could be used as a blocking variable, though this was not done here.

c) Standardized effect sizes

Extract simulated responses, calculate mean and sdev

```
Results.table.sim <- list()

for(r in 1:length(Results.sim))
```

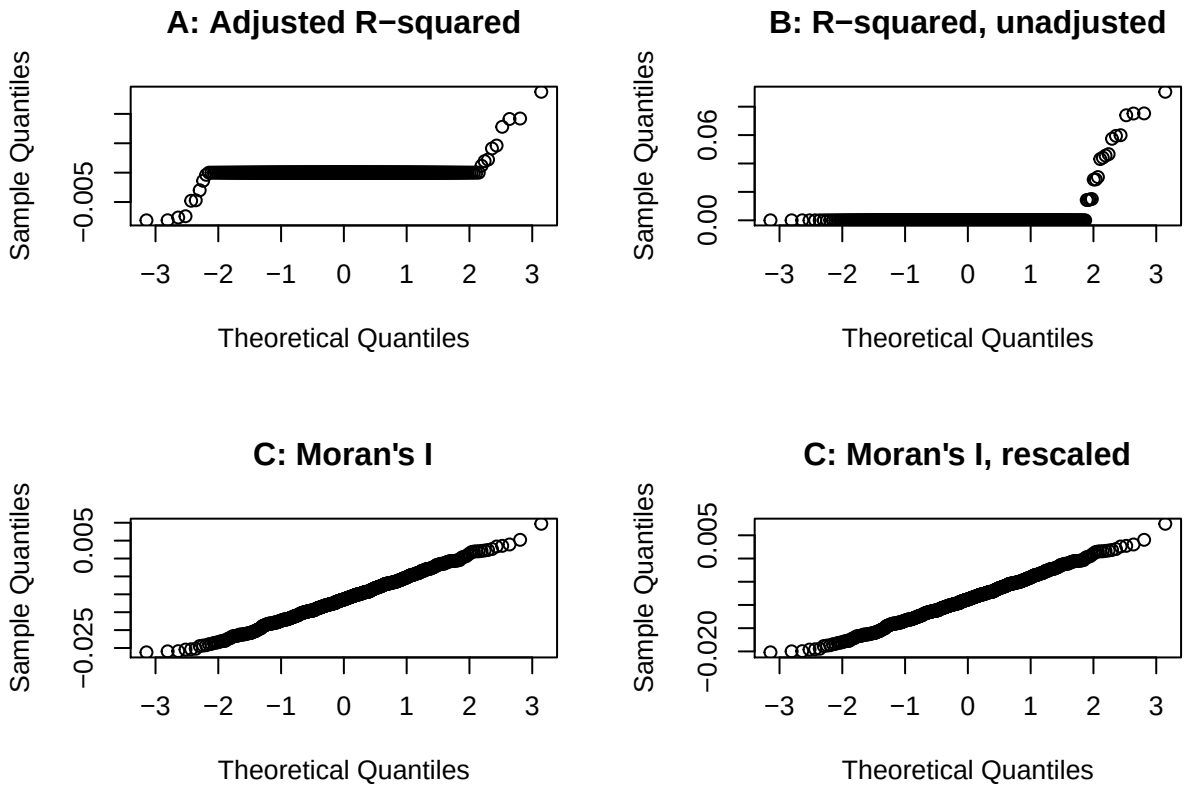
```
{
  Results.table.sim[[r]] <- lapply(c(1:length(Results.sim[[r]][[1]])),
                                   function(k) getRes(Results.sim[[r]], k))
}
RsqAdjPos.sim.Fst <- data.frame(t(sapply(Results.table.sim, function(ls) ls[[1]]$RsqAdjPos)))
RsqPos.sim.Fst <- data.frame(t(sapply(Results.table.sim, function(ls) ls[[1]]$RsqPos)))
Morans.I.sim.Fst <- data.frame(t(sapply(Results.table.sim, function(ls) ls[[1]]$Morans.I)))
I.scaled.sim.Fst <- data.frame(t(sapply(Results.table.sim, function(ls) ls[[1]]$I.scaled)))

Sim.distributions <- data.frame(RsqAdjPos.mean = apply(RsqAdjPos.sim.Fst, 2, mean),
           RsqAdjPos.sd = apply(RsqPos.sim.Fst, 2, sd),
           RsqPos.mean = apply(RsqPos.sim.Fst, 2, mean),
           RsqPos.sd = apply(RsqAdjPos.sim.Fst, 2, sd),
           Morans.I.mean = apply(Morans.I.sim.Fst, 2, mean),
           Morans.I.sd = apply(Morans.I.sim.Fst, 2, sd),
           I.scaled.mean = apply(I.scaled.sim.Fst, 2, mean),
           I.scaled.sd = apply(I.scaled.sim.Fst, 2, sd))
row.names(Sim.distributions) <- names(Results.sim[[1]])

saveRDS(Sim.distributions, paste0(here::here(), "/output/Sim.distributions.rds"))
```

Check visually for normal distribution:

```
par(mfrow=c(2,2))
qqnorm(data.matrix(RsqAdjPos.sim.Fst[,c(1,3,5)])[], main="A: Adjusted R-squared")
qqnorm(data.matrix(RsqPos.sim.Fst[,c(1,3,5)])[], main="B: R-squared, unadjusted")
qqnorm(data.matrix(Morans.I.sim.Fst[,c(1,3,5)])[], main="C: Moran's I")
qqnorm(data.matrix(I.scaled.sim.Fst[,c(1,3,5)])[], main="C: Moran's I, rescaled")
```




```
par(mfrow=c(1,1))
```

For each response, subtract mean of simulated values and divide by their sdev (Fst only)

```
getSES <- function(Table=Results.table[[3]], Sim=Sim.distributions)
{
  Table$SES.RsqAdjPos.90 = (Table$Fst.RsqAdjPos.90 - Sim$RsqAdjPos.mean) / Sim$RsqAdjPos.sd
  Table$SES.RsqPos.90 = (Table$Fst.RsqPos.90 - Sim$RsqPos.mean) / Sim$RsqPos.sd
  Table$SES.Morans.I.90 = (Table$Fst.Morans.I.90 - Sim$Morans.I.mean) / Sim$Morans.I.sd
  Table$SES.I.scaled.90 = (Table$Fst.I.scaled.90 - Sim$I.scaled.mean) / Sim$I.scaled.sd
  Table
}
```

```
Results.table <- readRDS(paste0(here::here(), "/output/Results.table.rds"))

s = 3

Results.table[[s]] <- getSES(Table=Results.table[[s]], Sim=Sim.distributions)

saveRDS(Results.table, paste0(here::here(), "/output/Results.table.rds"))
```

5. Analyze data for subsets with 30 pops

a) Repeat MEMgene with n = 30

The R = 10 subsets of n = 30 sites for each dataset with n = 90 sites were defined above already in object `Sites.30`.

```
start_time <- Sys.time()

Index.j <- 1:length(Sites.R.90)
RR = length(Sites.30)
Results.R.30 <- list()

for(s in 1:length(Dgen.R.90))
{
  cat("\n", s, ": ")
  Results.R.30[[s]] <- rep( list(list()), RR)
  for(rr in 1:RR)
  {
    cat(rr)
    Results.R.30[[s]][[rr]] <- mclapply(Index.j, function(j) getMEMgene(j, Dgen = Dgen.R.90[[s]],
                                                                        subset=Sites.30[[rr]]),
                                        mc.cores=detectCores())
    names(Results.R.30[[s]][[rr]]) <- Sites.R.90
  }
}
names(Results.R.30) <- names(Dgen.R.90)


end_time <- Sys.time()
end_time - start_time
cat("This job took", (end_time - start_time)/length(Index.j)/length(Dgen.R.90), "per dataset.")

saveRDS(Results.R.30, paste0(here::here(), "/output/Results.R.30.rds"))
```

Repeat for simulated data (only for 500 loci)

```
start_time <- Sys.time()

Index.j <- 1:length(Sites.R.90)
RR = length(Sites.30)
Results.R.30.sim <- list()

for(r in 1:length(Dgen.sim))
{
  cat(r, " ")
  Results.R.30.sim[[r]] <- list()

  s = 3

  Results.R.30.sim[[r]] <- rep( list(list()), RR)
  for(rr in 1:RR)
  {
```

```
    Results.R.30.sim[[r]][[rr]] <- mclapply(Index.j,
                                            function(j) getMEMgene(j, Dgen = Dgen.sim[[r]][[s]],
                                            subset=Sites.30[[rr]]),
                                            mc.cores=detectCores())
    names(Results.R.30.sim[[r]][[rr]]) <- Sites.R.90
  }
}

end_time <- Sys.time()
end_time - start_time
cat("This job took", (end_time - start_time)/length(Index.j), "per dataset.")

saveRDS(Results.R.30.sim, paste0(here::here(), "/output/Results.R.30.sim.rds"))
```

b) Compile results for n = 30 sites

Extract results for each replicate subsample

```
Results.R.30 <- readRDS(paste0(here::here(), "/output/Results.R.30.rds"))

res.r <- list()
Results.table.30 <- Results.table

for(s in 1:length(Results.R.30))
{
  res.r[[s]] <- lapply(Results.R.30[[s]], function(sub)
      lapply(c(1:length(Results.R.30[[s]][[1]][[1]])), function(k) getRes(sub, k)))

  Results.table.30[[s]]$Fst.RsqAdjPos.30m <- apply(data.frame(t(sapply(res.r[[s]],
                                            function(ls) ls[[1]]$RsqAdjPos))), 2, mean)
  Results.table.30[[s]]$Fst.RsqAdjPos.30s <- apply(data.frame(t(sapply(res.r[[s]],
                                            function(ls) ls[[1]]$RsqAdjPos))), 2, sd)
  Results.table.30[[s]]$Fst.RsqPos.30m <- apply(data.frame(t(sapply(res.r[[s]],
                                            function(ls) ls[[1]]$RsqPos))), 2, mean)
  Results.table.30[[s]]$Fst.RsqPos.30s <- apply(data.frame(t(sapply(res.r[[s]],
                                            function(ls) ls[[1]]$RsqPos))), 2, sd)
  Results.table.30[[s]]$Fst.Morans.I.30m <- apply(data.frame(t(sapply(res.r[[s]],
                                            function(ls) ls[[1]]$Morans.I))), 2, mean)
  Results.table.30[[s]]$Fst.Morans.I.30s <- apply(data.frame(t(sapply(res.r[[s]],
                                            function(ls) ls[[1]]$Morans.I))), 2, sd)
  Results.table.30[[s]]$Fst.I.scaled.30m <- apply(data.frame(t(sapply(res.r[[s]],
                                            function(ls) ls[[1]]$I.scaled))), 2, mean)
  Results.table.30[[s]]$Fst.I.scaled.30s <- apply(data.frame(t(sapply(res.r[[s]],
                                            function(ls) ls[[1]]$I.scaled))), 2, sd)
  Results.table.30[[s]]$Dch.RsqAdjPos.30m <- apply(data.frame(t(sapply(res.r[[s]],
                                            function(ls) ls[[2]]$RsqAdjPos))), 2, mean)
  Results.table.30[[s]]$Dch.RsqAdjPos.30s <- apply(data.frame(t(sapply(res.r[[s]],
                                            function(ls) ls[[2]]$RsqAdjPos))), 2, sd)
  Results.table.30[[s]]$Dch.RsqPos.30m <- apply(data.frame(t(sapply(res.r[[s]],
                                            function(ls) ls[[2]]$RsqPos))), 2, mean)
  Results.table.30[[s]]$Dch.RsqPos.30s <- apply(data.frame(t(sapply(res.r[[s]],
                                            function(ls) ls[[2]]$RsqPos))), 2, sd)
```

```
  Results.table.30[[s]]$Dch.Morans.I.30m <- apply(data.frame(t(sapply(res.r[[s]],
                                            function(ls) ls[[2]]$Morans.I))), 2, mean)
  Results.table.30[[s]]$Dch.Morans.I.30s <- apply(data.frame(t(sapply(res.r[[s]],
                                            function(ls) ls[[2]]$Morans.I))), 2, sd)
  Results.table.30[[s]]$Dch.I.scaled.30m <- apply(data.frame(t(sapply(res.r[[s]],
                                            function(ls) ls[[2]]$I.scaled))), 2, mean)
  Results.table.30[[s]]$Dch.I.scaled.30s <- apply(data.frame(t(sapply(res.r[[s]],
                                            function(ls) ls[[2]]$I.scaled))), 2, sd)
}
```

Repeat for simulated data, get means and sdev (Fst only)

```
Results.table.sim.30 <- list()

  for(r in 1:length(Results.R.30.sim))
  {
    Results.table.sim.30[[r]] <- lapply(Results.R.30.sim[[r]], function(sub) getRes(sub, 1))
  }

  RsqAdjPos.sim.30.Fst <- lapply(Results.table.sim.30, function(ls) sapply(ls,
                                function(x) x$RsqAdjPos))
  RsqPos.sim.30.Fst <- lapply(Results.table.sim.30, function(ls) sapply(ls,
                                function(x) x$RsqPos))
  Morans.I.sim.30.Fst <- lapply(Results.table.sim.30, function(ls) sapply(ls,
                                function(x) x$Morans.I))
  I.scaled.sim.30.Fst <- lapply(Results.table.sim.30, function(ls) sapply(ls,
                                function(x) x$I.scaled))


  RsqAdjPos.30.mean <- Reduce("+", RsqAdjPos.sim.30.Fst)/length(RsqAdjPos.sim.30.Fst)
  tmp <- lapply(RsqAdjPos.sim.30.Fst, function(ls) (ls - RsqAdjPos.30.mean)^2)
  RsqAdjPos.30.sd <- sqrt(Reduce("+", tmp)/(length(tmp) - 1))

  RsqPos.30.mean <- Reduce("+", RsqPos.sim.30.Fst)/length(RsqPos.sim.30.Fst)
  tmp <- lapply(RsqPos.sim.30.Fst, function(ls) (ls - RsqPos.30.mean)^2)
  RsqPos.30.sd <- sqrt(Reduce("+", tmp)/(length(tmp) - 1))

  Morans.I.30.mean <- Reduce("+", Morans.I.sim.30.Fst)/length(Morans.I.sim.30.Fst)
  tmp <- lapply(Morans.I.sim.30.Fst, function(ls) (ls - Morans.I.30.mean)^2)
  Morans.I.30.sd <- sqrt(Reduce("+", tmp)/(length(tmp) - 1))

  I.scaled.30.mean <- Reduce("+", I.scaled.sim.30.Fst)/length(I.scaled.sim.30.Fst)
  tmp <- lapply(I.scaled.sim.30.Fst, function(ls) (ls - I.scaled.30.mean)^2)
  I.scaled.30.sd <- sqrt(Reduce("+", tmp)/(length(tmp) - 1))

  Sim.distributions.30 <- list(RsqAdjPos.30.mean = RsqAdjPos.30.mean,
           RsqAdjPos.30.sd = RsqAdjPos.30.sd,
           RsqPos.30.mean = RsqPos.30.mean,
           RsqPos.30.sd = RsqPos.30.sd,
           Morans.I.30.mean = Morans.I.30.mean,
           Morans.I.30.sd = Morans.I.30.sd,
           I.scaled.30.mean = I.scaled.30.mean,
           I.scaled.30.sd = I.scaled.30.sd)
```

```
saveRDS(Sim.distributions.30, paste0(here::here(), "/output/Sim.distributions.30.rds"))
```

Calculate SES for replicate subsamples

```
Results.subsets.table <- Results.table.30

#for(s in length(Results.table.30):1)
s = 3

  Results.subsets.table[[s]]$SES.RsqAdjPos.30m <-
    apply((t(sapply(res.r[[s]], function(ls) ls[[1]]$RsqAdjPos)) -
       t(Sim.distributions.30$RsqAdjPos.30.mean)) /
       t(Sim.distributions.30$RsqAdjPos.30.sd), 2, mean)
  Results.subsets.table[[s]]$SES.RsqAdjPos.30s <-
    apply((t(sapply(res.r[[s]], function(ls) ls[[1]]$RsqAdjPos)) -
       t(Sim.distributions.30$RsqAdjPos.30.mean)) /
       t(Sim.distributions.30$RsqAdjPos.30.sd), 2, sd)
  Results.subsets.table[[s]]$SES.RsqPos.30m <-
    apply((t(sapply(res.r[[s]], function(ls) ls[[1]]$RsqPos)) -
       t(Sim.distributions.30$RsqPos.30.mean)) /
       t(Sim.distributions.30$RsqPos.30.sd), 2, mean)
  Results.subsets.table[[s]]$SES.RsqPos.30s <-
    apply((t(sapply(res.r[[s]], function(ls) ls[[1]]$RsqPos)) -
       t(Sim.distributions.30$RsqPos.30.mean)) /
       t(Sim.distributions.30$RsqPos.30.sd), 2, sd)
  Results.subsets.table[[s]]$SES.Morans.I.30m <-
    apply((t(sapply(res.r[[s]], function(ls) ls[[1]]$Morans.I)) -
       t(Sim.distributions.30$Morans.I.30.mean)) /
       t(Sim.distributions.30$Morans.I.30.sd), 2, mean)
  Results.subsets.table[[s]]$SES.Morans.I.30s <-
    apply((t(sapply(res.r[[s]], function(ls) ls[[1]]$Morans.I)) -
       t(Sim.distributions.30$Morans.I.30.mean)) /
       t(Sim.distributions.30$Morans.I.30.sd), 2, sd)
  Results.subsets.table[[s]]$SES.I.scaled.30m <-
    apply((t(sapply(res.r[[s]], function(ls) ls[[1]]$I.scaled)) -
       t(Sim.distributions.30$I.scaled.30.mean)) /
       t(Sim.distributions.30$I.scaled.30.sd), 2, mean)
  Results.subsets.table[[s]]$SES.I.scaled.30s <-
    apply((t(sapply(res.r[[s]], function(ls) ls[[1]]$I.scaled)) -
       t(Sim.distributions.30$I.scaled.30.mean)) /
       t(Sim.distributions.30$I.scaled.30.sd), 2, sd)


saveRDS(Results.subsets.table, paste0(here::here(), "/output/Results.subsets.table.rds"))
```

c) Compare subsamples with n = 30 to fulld samples

Paired t-tests (without IM)

Drop IM:

```
Results.subsets.table <- readRDS(paste0(here::here(), "/output/Results.subsets.table.rds"))

a <- which(Results.subsets.table[[3]]$Demography != "IM")
```

Paired t-tests for difference between mean of subsamples with n = 30 and corresponding datasets with n = 90 sites.

```
s = 1

with(Results.subsets.table[[s]][a,], t.test(Fst.RsqAdjPos.30m, Fst.RsqAdjPos.90, paired=TRUE))
```

```
##
##  Paired t-test
##
## data:  Fst.RsqAdjPos.30m and Fst.RsqAdjPos.90
## t = -10.065, df = 17, p-value = 1.409e-08
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.09269467 -0.06056670
## sample estimates:
## mean of the differences
##             -0.07663068
```

```
with(Results.subsets.table[[s]][a,], t.test(Fst.Morans.I.30m, Fst.Morans.I.90, paired=TRUE))
```

```
##
##  Paired t-test
##
## data:  Fst.Morans.I.30m and Fst.Morans.I.90
## t = -17.211, df = 17, p-value = 3.429e-12
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.09968834 -0.07791683
## sample estimates:
## mean of the differences
##             -0.08880259
```

```
with(Results.subsets.table[[s]][a,], t.test(Fst.I.scaled.30m, Fst.I.scaled.90, paired=TRUE))
```

```
##
##  Paired t-test
##
## data:  Fst.I.scaled.30m and Fst.I.scaled.90
## t = -9.5475, df = 17, p-value = 3.039e-08
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.05254855 -0.03352733
## sample estimates:
## mean of the differences
##             -0.04303794
```

Repeat for SES (with 500 loci)

```
s = 3  # For SES

with(Results.subsets.table[[s]][a,], t.test(SES.RsqAdjPos.30m, SES.RsqAdjPos.90, paired=TRUE))
```

```
##
##  Paired t-test
##
## data:  SES.RsqAdjPos.30m and SES.RsqAdjPos.90
## t = 4.9195, df = 17, p-value = 0.0001297
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  34.53706 86.40583
## sample estimates:
## mean of the differences
##                60.47144
```

```
with(Results.subsets.table[[s]][a,], t.test(SES.Morans.I.30m, SES.RsqAdjPos.90, paired=TRUE))
```

```
##
##  Paired t-test
##
## data:  SES.Morans.I.30m and SES.RsqAdjPos.90
## t = -4.2225, df = 17, p-value = 0.0005731
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -38.58810 -12.87429
## sample estimates:
## mean of the differences
##               -25.73119
```

```
with(Results.subsets.table[[s]][a,], t.test(SES.I.scaled.30m, SES.RsqAdjPos.90, paired=TRUE))
```

```
##
##  Paired t-test
##
## data:  SES.I.scaled.30m and SES.RsqAdjPos.90
## t = -4.2225, df = 17, p-value = 0.0005731
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -38.58810 -12.87429
## sample estimates:
## mean of the differences
##               -25.73119
```

Effect sizes (without IM)

Cohen's d for paired t-test.

```
s=1

with(Results.subsets.table[[s]][a,], mean(Fst.RsqAdjPos.30m - Fst.RsqAdjPos.90)/
       sd(Fst.RsqAdjPos.30m - Fst.RsqAdjPos.90))
```

```
## [1] -2.372235
```

```
with(Results.subsets.table[[s]][a,], mean(Fst.Morans.I.30m - Fst.Morans.I.90)/
       sd(Fst.Morans.I.30m - Fst.Morans.I.90))
```

```
## [1] -4.056722
```

```
with(Results.subsets.table[[s]][a,], mean(Fst.I.scaled.30m - Fst.I.scaled.90)/
       sd(Fst.I.scaled.30m - Fst.I.scaled.90))
```

```
## [1] -2.250356
```

6. Meta-analysis vs. comparative mode

Simulations to compare results between full sample and sample with some missing sites. Two possible methods:

- Meta-analysis mode: redo MEMgene (including MEM) for each dataset, based on available sites.
- Comparative mode: keep MEM, do adj R2 based on positive MEM, which avoids overfitting.

a) Leave 30 sites out at a time, two modes

- Comparative mode: use MEM from full dataset.
- Meta-analysis mode: MEM based on subset of 60 sites.

For 500 loci:

```
start_time <- Sys.time()

Index.j <- 1:length(Sites.R.90)
RR = 10
Results.drop <- Results.drop.MEM <- rep( list(list()), R)

set.seed(297)
Drop <-lapply(c(1:R), function(r) sort(sample(1:90, 30)))

for(rr in 1:RR)
{
  cat(rr)
  Results.drop[[rr]] <- mclapply(Index.j, function(j) getMEMgene(j, Dgen = Dgen.R.90[[3]],
                                                    subset=NULL, drop=Drop[[rr]]),
                                 mc.cores=detectCores())
   Results.drop.MEM[[rr]] <- mclapply(Index.j, function(j) getMEMgene(j, Dgen = Dgen.R.90[[3]],
                                                    subset=c(1:90)[-Drop[[rr]]], drop=NULL),
```

```
                                  mc.cores=detectCores())
  names(Results.drop[[rr]]) <- names(Results.drop.MEM[[rr]]) <- Sites.R.90
}

end_time <- Sys.time()
end_time - start_time
cat("This job took", (end_time - start_time)/length(Index.j), "per dataset.")

saveRDS(Results.drop, paste0(here::here(), "/output/Results.drop.rds"))
saveRDS(Results.drop.MEM, paste0(here::here(), "/output/Results.drop.MEM.rds"))
```

Repeat for simulated data: comparative and meta-analysis modes

```
start_time <- Sys.time()

Index.j <- 1:length(Sites.R.90)
RR = 10

Results.drop.sim <- Results.drop.sim.MEM <- list()

for(r in 1:length(Dgen.sim))
{
  cat("\n", r, " ")
  Results.drop.sim[[r]] <- Results.drop.sim.MEM[[r]] <- list()
  for(rr in 1:RR)
  {
    cat(rr)
    Results.drop.sim[[r]][[rr]] <- mclapply(Index.j, function(j)
                                    getMEMgene(j, Dgen = Dgen.sim[[r]][[3]],
                                               subset=NULL, drop=Drop[[rr]]),
                                  mc.cores=detectCores())
    Results.drop.sim.MEM[[r]][[rr]] <- mclapply(Index.j, function(j)
                                    getMEMgene(j, Dgen = Dgen.sim[[r]][[3]],
                                               subset=c(1:90)[-Drop[[rr]]], drop=NULL),
                                  mc.cores=detectCores())
    names(Results.drop.sim[[r]][[rr]]) <-  names(Results.drop.sim.MEM[[r]][[rr]]) <- Sites.R.90
  }
}

end_time <- Sys.time()
end_time - start_time
cat("This job took", (end_time - start_time)/length(Index.j), "per dataset.")

saveRDS(Results.drop.sim, paste0(here::here(), "/output/Results.drop.sim.rds"))
saveRDS(Results.drop.sim.MEM, paste0(here::here(), "/output/Results.drop.MEM.rds"))
```

Get means and sdev for simulated data(500 loci, pairwise Fst only)

```
#Results.drop.sim <- readRDS(paste0(here::here(), "/output/Results.drop.sim.rds"))
#Results.drop.sim.MEM <- readRDS(paste0(here::here(), "/output/Results.drop.MEM.rds"))

Results.table.drop.sim <- list()
Results.table.drop.sim.MEM <- list()
```

```
#for(s in length(loci):1)
s = 3
  for(r in 1:length(Results.drop.sim))
  {
    Results.table.drop.sim[[r]] <- lapply(Results.drop.sim[[r]], function(sub)
                    getRes(sub, 1))
    Results.table.drop.sim.MEM[[r]] <- lapply(Results.drop.sim.MEM[[r]], function(sub)
                    getRes(sub, 1))
  }

  RsqAdjPos.drop.sim.Fst <- lapply(Results.table.drop.sim, function(ls)
                                  sapply(ls, function(x) x$RsqAdjPos))
  RsqPos.drop.sim.Fst <- lapply(Results.table.drop.sim, function(ls)
                                  sapply(ls, function(x) x$RsqPos))
  Morans.I.drop.sim.Fst <- lapply(Results.table.drop.sim, function(ls)
                                  sapply(ls, function(x) x$Morans.I))
  I.scaled.drop.sim.Fst <- lapply(Results.table.drop.sim, function(ls)
                                  sapply(ls, function(x) x$I.scaled))
  RsqAdjPos.drop.sim.MEM.Fst <- lapply(Results.table.drop.sim.MEM, function(ls)
                                  sapply(ls, function(x) x$RsqAdjPos))
  RsqPos.drop.sim.MEM.Fst <- lapply(Results.table.drop.sim.MEM, function(ls)
                                  sapply(ls, function(x) x$RsqPos))
  Morans.I.drop.sim.MEM.Fst <- lapply(Results.table.drop.sim.MEM, function(ls)
                                  sapply(ls, function(x) x$Morans.I))
  I.scaled.drop.sim.MEM.Fst <- lapply(Results.table.drop.sim.MEM, function(ls)
                                  sapply(ls, function(x) x$I.scaled))

  RsqAdjPos.drop.sim.mean <- Reduce("+", RsqAdjPos.drop.sim.Fst)/length(RsqAdjPos.drop.sim.Fst)
  tmp <- lapply(RsqAdjPos.drop.sim.Fst, function(ls) (ls - RsqAdjPos.drop.sim.mean)^2)
  RsqAdjPos.drop.sim.sd <- sqrt(Reduce("+", tmp)/(length(tmp) - 1))

  RsqPos.drop.sim.mean <- Reduce("+", RsqPos.drop.sim.Fst)/length(RsqPos.drop.sim.Fst)
  tmp <- lapply(RsqPos.drop.sim.Fst, function(ls) (ls - RsqPos.drop.sim.mean)^2)
  RsqPos.drop.sim.sd <- sqrt(Reduce("+", tmp)/(length(tmp) - 1))

  Morans.I.drop.sim.mean <- Reduce("+", Morans.I.drop.sim.Fst)/length(Morans.I.drop.sim.Fst)
  tmp <- lapply(Morans.I.drop.sim.Fst, function(ls) (ls - Morans.I.drop.sim.mean)^2)
  Morans.I.drop.sim.sd <- sqrt(Reduce("+", tmp)/(length(tmp) - 1))

  I.scaled.drop.sim.mean <- Reduce("+", I.scaled.drop.sim.Fst)/length(I.scaled.drop.sim.Fst)
  tmp <- lapply(I.scaled.drop.sim.Fst, function(ls) (ls - I.scaled.drop.sim.mean)^2)
  I.scaled.drop.sim.sd <- sqrt(Reduce("+", tmp)/(length(tmp) - 1))

  RsqAdjPos.drop.sim.MEM.mean <- Reduce("+", RsqAdjPos.drop.sim.MEM.Fst)/
                                          length(RsqAdjPos.drop.sim.MEM.Fst)
  tmp <- lapply(RsqAdjPos.drop.sim.MEM.Fst, function(ls) (ls - RsqAdjPos.drop.sim.MEM.mean)^2)
  RsqAdjPos.drop.sim.MEM.sd <- sqrt(Reduce("+", tmp)/(length(tmp) - 1))

  RsqPos.drop.sim.MEM.mean <- Reduce("+", RsqPos.drop.sim.MEM.Fst)/
                                          length(RsqPos.drop.sim.MEM.Fst)
  tmp <- lapply(RsqPos.drop.sim.MEM.Fst, function(ls) (ls - RsqPos.drop.sim.MEM.mean)^2)
  RsqPos.drop.sim.MEM.sd <- sqrt(Reduce("+", tmp)/(length(tmp) - 1))
```

```
  Morans.I.drop.sim.MEM.mean <- Reduce("+", Morans.I.drop.sim.MEM.Fst)/
                                          length(Morans.I.drop.sim.MEM.Fst)
  tmp <- lapply(Morans.I.drop.sim.MEM.Fst, function(ls) (ls - Morans.I.drop.sim.MEM.mean)^2)
  Morans.I.drop.sim.MEM.sd <- sqrt(Reduce("+", tmp)/(length(tmp) - 1))

  I.scaled.drop.sim.MEM.mean <- Reduce("+", I.scaled.drop.sim.MEM.Fst)/
                                          length(I.scaled.drop.sim.MEM.Fst)
  tmp <- lapply(I.scaled.drop.sim.MEM.Fst, function(ls) (ls - I.scaled.drop.sim.MEM.mean)^2)
  I.scaled.drop.sim.MEM.sd <- sqrt(Reduce("+", tmp)/(length(tmp) - 1))

  Sim.drop.distributions <- list(RsqAdjPos.drop.sim.mean = RsqAdjPos.drop.sim.mean,
           RsqAdjPos.drop.sim.sd = RsqAdjPos.drop.sim.sd,
           RsqPos.drop.sim.mean = RsqPos.drop.sim.mean,
           RsqPos.drop.sim.sd = RsqPos.drop.sim.sd,
           Morans.I.drop.sim.mean = Morans.I.drop.sim.mean,
           Morans.I.drop.sim.sd = Morans.I.drop.sim.sd,
           I.scaled.drop.sim.mean = I.scaled.drop.sim.mean,
           I.scaled.drop.sim.sd = I.scaled.drop.sim.sd,

           RsqAdjPos.drop.sim.MEM.mean = RsqAdjPos.drop.sim.MEM.mean,
           RsqAdjPos.drop.sim.MEM.sd = RsqAdjPos.drop.sim.MEM.sd,
           RsqPos.drop.sim.MEM.mean = RsqPos.drop.sim.MEM.mean,
           RsqPos.drop.sim.MEM.sd = RsqPos.drop.sim.MEM.sd,
           Morans.I.drop.sim.MEM.mean = Morans.I.drop.sim.MEM.mean,
           Morans.I.drop.sim.MEM.sd = Morans.I.drop.sim.MEM.sd,
           I.scaled.drop.sim.MEM.mean = I.scaled.drop.sim.MEM.mean,
           I.scaled.drop.sim.MEM.sd = I.scaled.drop.sim.MEM.sd)

  saveRDS(Sim.drop.distributions, paste0(here::here(), "/output/Sim.drop.distributions.rds"))
```

b) Compile results for n = 60, two modes

Determine mean and sdev among the R = 10 replicate subsets for each dataset with 5 sites dropped.

```
#Results.drop <- readRDS(paste0(here::here(), "/output/Results.drop.rds"))
#Results.drop.MEM <- readRDS(paste0(here::here(), "/output/Results.drop.MEM.rds"))
#Results.subsets.table <- readRDS(paste0(here::here(), "/output/Results.subsets.table.rds"))

res.r <- lapply(Results.drop, function(sub)
  lapply(c(1:length(Results.drop[[1]][[1]])), function(k) getRes(sub, k)))
res.r.MEM <- lapply(Results.drop.MEM, function(sub)
  lapply(c(1:length(Results.drop.MEM[[1]][[1]])), function(k) getRes(sub, k)))
```

```
tmp <- list()
for(k in 1:length(res.r[[1]]))
  {
    RsqAdjPos.drop.m <- apply(sapply(res.r, function(sub) sub[[k]]$RsqAdjPos), 1, mean)
    RsqAdjPos.drop.s <- apply(sapply(res.r, function(sub) sub[[k]]$RsqAdjPos), 1, sd)
    RsqPos.drop.m <- apply(sapply(res.r, function(sub) sub[[k]]$RsqPos), 1, mean)
    RsqPos.drop.s <- apply(sapply(res.r, function(sub) sub[[k]]$RsqPos), 1, sd)
    Morans.I.drop.m <- apply(sapply(res.r, function(sub) sub[[k]]$Morans.I), 1, mean)
    Morans.I.drop.s <- apply(sapply(res.r, function(sub) sub[[k]]$Morans.I), 1, sd)
```

```
    I.scaled.drop.m <- apply(sapply(res.r, function(sub) sub[[k]]$I.scaled), 1, mean)
    I.scaled.drop.s <- apply(sapply(res.r, function(sub) sub[[k]]$I.scaled), 1, sd)

    RsqAdjPos.drop.MEM.m <- apply(sapply(res.r.MEM, function(sub) sub[[k]]$RsqAdjPos), 1, mean)
    RsqAdjPos.drop.MEM.s <- apply(sapply(res.r.MEM, function(sub) sub[[k]]$RsqAdjPos), 1, sd)
    RsqPos.drop.MEM.m <- apply(sapply(res.r.MEM, function(sub) sub[[k]]$RsqPos), 1, mean)
    RsqPos.drop.MEM.s <- apply(sapply(res.r.MEM, function(sub) sub[[k]]$RsqPos), 1, sd)
    Morans.I.drop.MEM.m <- apply(sapply(res.r.MEM, function(sub) sub[[k]]$Morans.I), 1, mean)
    Morans.I.drop.MEM.s <- apply(sapply(res.r.MEM, function(sub) sub[[k]]$Morans.I), 1, sd)
    I.scaled.drop.MEM.m <- apply(sapply(res.r.MEM, function(sub) sub[[k]]$I.scaled), 1, mean)
    I.scaled.drop.MEM.s <- apply(sapply(res.r.MEM, function(sub) sub[[k]]$I.scaled), 1, sd)

    tmp[[k]] <- data.frame(RsqAdjPos.drop.m=RsqAdjPos.drop.m, RsqAdjPos.drop.s=RsqAdjPos.drop.s,
                           RsqPos.drop.m=RsqPos.drop.m, RsqPos.drop.s=RsqPos.drop.s,
                           Morans.I.drop.m=Morans.I.drop.m, Morans.I.drop.s=Morans.I.drop.s,
                           I.scaled.drop.m=I.scaled.drop.m, I.scaled.drop.s=I.scaled.drop.s,

                           RsqAdjPos.drop.MEM.m=RsqAdjPos.drop.MEM.m,
                           RsqAdjPos.drop.MEM.s=RsqAdjPos.drop.MEM.s,
                           RsqPos.drop.MEM.m=RsqPos.drop.MEM.m,
                           RsqPos.drop.MEM.s=RsqPos.drop.MEM.s,
                           Morans.I.drop.MEM.m=Morans.I.drop.MEM.m,
                           Morans.I.drop.MEM.s=Morans.I.drop.MEM.s,
                           I.scaled.drop.MEM.m=I.scaled.drop.MEM.m,
                           I.scaled.drop.MEM.s=I.scaled.drop.MEM.s)
}

res.r.combined <- Reduce(cbind, tmp)

names(res.r.combined) <- paste(rep(names(Results.drop[[1]][[1]]), each=ncol(tmp[[1]])),
                               names(tmp[[1]]), sep=".")

Results.subsets.table[[3]] <- data.frame(Results.subsets.table[[3]], res.r.combined)
```

Calculate SES

```
s = 3

  Results.subsets.table[[s]]$SES.RsqAdjPos.drop.m <-
    apply((t(sapply(res.r, function(ls) ls[[1]]$RsqAdjPos)) -
       t(Sim.drop.distributions$RsqAdjPos.drop.sim.mean)) /
       t(Sim.drop.distributions$RsqAdjPos.drop.sim.sd), 2, mean)
  Results.subsets.table[[s]]$SES.RsqAdjPos.drop.s <-
    apply((t(sapply(res.r, function(ls) ls[[1]]$RsqAdjPos)) -
       t(Sim.drop.distributions$RsqAdjPos.drop.sim.mean)) /
       t(Sim.drop.distributions$RsqAdjPos.drop.sim.sd), 2, sd)

  Results.subsets.table[[s]]$SES.RsqPos.drop.m <-
    apply((t(sapply(res.r, function(ls) ls[[1]]$RsqPos)) -
       t(Sim.drop.distributions$RsqPos.drop.sim.mean)) /
       t(Sim.drop.distributions$RsqPos.drop.sim.sd), 2, mean)
  Results.subsets.table[[s]]$SES.RsqPos.drop.s <-
    apply((t(sapply(res.r, function(ls) ls[[1]]$RsqPos)) -
```

```
      t(Sim.drop.distributions$RsqPos.drop.sim.mean)) /
      t(Sim.drop.distributions$RsqPos.drop.sim.sd), 2, sd)

Results.subsets.table[[s]]$SES.Morans.I.drop.m <-
  apply((t(sapply(res.r, function(ls) ls[[1]]$Morans.I)) -
      t(Sim.drop.distributions$Morans.I.drop.sim.mean)) /
      t(Sim.drop.distributions$Morans.I.drop.sim.sd), 2, mean)
Results.subsets.table[[s]]$SES.Morans.I.drop.s <-
  apply((t(sapply(res.r, function(ls) ls[[1]]$Morans.I)) -
      t(Sim.drop.distributions$Morans.I.drop.sim.mean)) /
      t(Sim.drop.distributions$Morans.I.drop.sim.sd), 2, sd)

Results.subsets.table[[s]]$SES.I.scaled.drop.m <-
  apply((t(sapply(res.r, function(ls) ls[[1]]$I.scaled)) -
      t(Sim.drop.distributions$I.scaled.drop.sim.mean)) /
      t(Sim.drop.distributions$I.scaled.drop.sim.sd), 2, mean)
Results.subsets.table[[s]]$SES.I.scaled.drop.s <-
  apply((t(sapply(res.r, function(ls) ls[[1]]$I.scaled)) -
      t(Sim.drop.distributions$I.scaled.drop.sim.mean)) /
      t(Sim.drop.distributions$I.scaled.drop.sim.sd), 2, sd)

Results.subsets.table[[s]]$SES.RsqAdjPos.drop.MEM.m <-
  apply((t(sapply(res.r, function(ls) ls[[1]]$RsqAdjPos)) -
      t(Sim.drop.distributions$RsqAdjPos.drop.sim.MEM.mean)) /
      t(Sim.drop.distributions$RsqAdjPos.drop.sim.MEM.sd), 2, mean)
Results.subsets.table[[s]]$SES.RsqAdjPos.drop.MEM.s <-
  apply((t(sapply(res.r, function(ls) ls[[1]]$RsqAdjPos)) -
      t(Sim.drop.distributions$RsqAdjPos.drop.sim.MEM.mean)) /
      t(Sim.drop.distributions$RsqAdjPos.drop.sim.MEM.sd), 2, sd)

Results.subsets.table[[s]]$SES.RsqPos.drop.MEM.m <-
  apply((t(sapply(res.r, function(ls) ls[[1]]$RsqPos)) -
      t(Sim.drop.distributions$RsqPos.drop.sim.MEM.mean)) /
      t(Sim.drop.distributions$RsqPos.drop.sim.MEM.sd), 2, mean)
Results.subsets.table[[s]]$SES.RsqPos.drop.MEM.s <-
  apply((t(sapply(res.r, function(ls) ls[[1]]$RsqPos)) -
      t(Sim.drop.distributions$RsqPos.drop.sim.MEM.mean)) /
      t(Sim.drop.distributions$RsqPos.drop.sim.MEM.sd), 2, sd)

Results.subsets.table[[s]]$SES.Morans.I.drop.MEM.m <-
  apply((t(sapply(res.r, function(ls) ls[[1]]$Morans.I)) -
      t(Sim.drop.distributions$Morans.I.drop.sim.MEM.mean)) /
      t(Sim.drop.distributions$Morans.I.drop.sim.MEM.sd), 2, mean)
Results.subsets.table[[s]]$SES.Morans.I.drop.MEM.s <-
  apply((t(sapply(res.r, function(ls) ls[[1]]$Morans.I)) -
      t(Sim.drop.distributions$Morans.I.drop.sim.MEM.mean)) /
      t(Sim.drop.distributions$Morans.I.drop.sim.MEM.sd), 2, sd)

Results.subsets.table[[s]]$SES.I.scaled.drop.MEM.m <-
  apply((t(sapply(res.r, function(ls) ls[[1]]$I.scaled)) -
      t(Sim.drop.distributions$I.scaled.drop.sim.MEM.mean)) /
      t(Sim.drop.distributions$I.scaled.drop.sim.MEM.sd), 2, mean)
Results.subsets.table[[s]]$SES.I.scaled.drop.MEM.s <-
```

```
    apply((t(sapply(res.r, function(ls) ls[[1]]$I.scaled)) -
       t(Sim.drop.distributions$I.scaled.drop.sim.MEM.mean)) /
       t(Sim.drop.distributions$I.scaled.drop.sim.MEM.sd), 2, sd)


saveRDS(Results.subsets.table, paste0(here::here(), "/output/Results.subsets.table.rds"))
```

7. Figures

```
library("rcompanion")
```

```
## Registered S3 method overwritten by 'DescTools':
##   method         from
##   reorder.factor gdata
```

```
library("ggpubr")
```

```
## Loading required package: ggplot2
```

```
library("ggplot2")
library("gridExtra")
```

```
##
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
library("reshape2")
library("tidyr")
```

```
##
## Attaching package: 'tidyr'
```

```
## The following object is masked from 'package:reshape2':
##
##     smiths
```

```
library("memgene")

source(paste0(here::here(), "/output/InteractionPlotLegend.R"))
```

a) Figure 1

```
Map.sites <- c(32,93,153,213)[c(4,3,1,2)]
Design[Map.sites,]
```

```
##     Demography Design Env NumPops NumInd NumTrans NumInd2 Type
## 213         IM    R90 453      90     20     <NA>    <NA>    R
## 153        IBD    R90 453      90     20     <NA>    <NA>    R
## 32          1R    R90 453      90     20     <NA>    <NA>    R
## 93          2R    R90 453      90     20     <NA>    <NA>    R
```

Note that alpha=0.1 is used to ensure some variables are selected also for IM.

```
# Extract the grid coordinates of the sampled sites
coord <- data.matrix(Coords[[as.numeric(Design$Type[Map.sites[1]])]]
                     [[as.numeric(Design$Env[Map.sites[1]])]][,2:3])
Dgen.R.90 <- readRDS(paste0(here::here(), "/output/Dgen.R.90.rds"))
Res.mem <- list()
k=1       # pairwise Fst
s=3       # 500 loci
for(j in 1:length(Map.sites))
{
  cat(j, " ")
  i = which(Sites.R.90 == Map.sites[j])
  Y <- as.matrix(Dgen.R.90[[s]][[i]]$Dgen[[k]])

  MEM <- mgMEM(dist(coord))
  MEM$Positive <- mgForward(Y, MEM$vectorsMEM[, MEM$valuesMEM > 0], alpha = 0.1)
  MEM$analysis <- mgRDA(Y, MEM$vectorsMEM[, MEM$Positive$selectedMEM], full=TRUE)
  MEM$AdjRsq.12 <-mgRDA(Y, MEM$analysis$memgene[,1:2])$RsqAdj
  Res.mem[[j]] <- MEM
}
```

```
## 1  2  3  4
```

Figure 1

```
pdf(paste0(here::here(), "/output/Figure1.pdf"), width=5.5, height=12)
par(mfrow=c(4,2), mai=c(0.3,0.3,0.5,0.2))

plot(coord[,1], coord[,2], type="n", xlab="", ylab="", axes=FALSE)
box()
mgMap(coord, Res.mem[[1]]$analysis$memgene[, 1], add.plot=TRUE)
mtext(bquote("a)  Island model: Adj." ~ R^2 ==
               .((round(max(0, Res.mem[[1]]$AdjRsq.12),2)))),
      side=3, adj=0, line=1)
plot(coord[,1], coord[,2], type="n", xlab="", ylab="", axes=FALSE)
box()
mgMap(coord, Res.mem[[1]]$analysis$memgene[, 2], add.plot=TRUE)


plot(coord[,1], coord[,2], type="n", xlab="", ylab="", axes=FALSE)
box()
```

```
mgMap(coord, Res.mem[[2]]$analysis$memgene[, 1], add.plot=TRUE)
mtext(bquote("b)  Isolation by distance: Adj." ~ R^2 ==
               .((round(
                 Res.mem[[2]]$AdjRsq.12,2)))),
      side=3, adj=0, line=1)
plot(coord[,1], coord[,2], type="n", xlab="", ylab="", axes=FALSE)
box()
mgMap(coord, Res.mem[[2]]$analysis$memgene[, 2], add.plot=TRUE)


plot(coord[,1], coord[,2], type="n", xlab="", ylab="", axes=FALSE)
box()
mgMap(coord, Res.mem[[3]]$analysis$memgene[, 1], add.plot=TRUE)
mtext(bquote("c)  One refugium: Adj." ~ R^2 ==
               .((round(Res.mem[[3]]$AdjRsq.12,2)))),
      side=3, adj=0, line=1)
plot(coord[,1], coord[,2], type="n", xlab="", ylab="", axes=FALSE)
box()
mgMap(coord, Res.mem[[3]]$analysis$memgene[, 2], add.plot=TRUE)


plot(coord[,1], coord[,2], type="n", xlab="", ylab="", axes=FALSE)
box()
mgMap(coord, Res.mem[[4]]$analysis$memgene[, 1], add.plot=TRUE)
mtext(bquote("d)  Two refugia: Adj." ~ R^2 ==
               .((round(Res.mem[[4]]$AdjRsq.12,2)))),
      side=3, adj=0, line=1)
plot(coord[,1], coord[,2], type="n", xlab="", ylab="", axes=FALSE)
box()
mgMap(coord, Res.mem[[4]]$analysis$memgene[, 2], add.plot=TRUE)


par(mfrow=c(1,1))
dev.off()
```

```
## pdf
##   2
```

a) Figure 2

With n=90, #IND= 20 & 9900 loci: Difference demographies in response variabe (Fst, adjusted R^2 and Moran's I).

```
Results.subsets.table <- readRDS(paste0(here::here(), "/output/Results.subsets.table.rds"))
Results.table <- Results.subsets.table

s = 1
b <- which(Results.table[[s]]$NumInd == 20)
Results.table[[s]]$Demography <- factor(Results.table[[s]]$Demography,
                                        levels = c("IM", "IBD", "1R", "2R"))

Metrics <- c("Fst.90", "Fst.RsqAdjPos.90", "Fst.Morans.I.90")
```

```
Labels <- c("Fst", expression('Adjusted R'^2), "Moran's I")
ymin = c(0.0, -0.02, -0.02)
ymax = c(0.02, 0.8, 0.8)
Title <- c("(a)", "(b)", "(c)")

Plots <- list()

for(m in 1:3)
{
  Sum1.p <- groupwiseMean(var = Metrics[m], group = "Demography", data = Results.table[[s]][b,],
                          conf   = 0.95, digits = 3)
  Plots[[m]] <- ggplot(Sum1.p, aes(x = Demography, y = Mean)) + ylim(ymin[m], ymax[m]) +
     geom_errorbar(aes(ymin = Trad.lower, ymax = Trad.upper), width = 0.05, size  = 0.5) +
     geom_point(shape = 1, size  = 2) +
     theme_bw() + theme (panel.grid.major = element_blank(), panel.grid.minor = element_blank()) +
      ggtitle(Title[m]) + xlab("Demography") + ylab(Labels[m]) +
      theme(axis.text = element_text(size = 12),
            plot.title = element_text(size=16, face="bold"),
            axis.title.x = element_text(size=12, face="bold"),
            axis.title.y = element_text(size=12, face="bold")
            )
}

g <- arrangeGrob(Plots[[1]], Plots[[2]], Plots[[3]], ncol = 3, nrow = 1) #generates g
 ggsave(file=paste0(here::here(), "/output/Figure2.pdf"), g, width=8, height=3)
```

b) Figure 3

With n=90 and removing IM: Interaction between #Ind (20, 6) and # Loci (9900, 3300, 500) in response variabe (global Fst, adjusted R^2, and Moran's I).

```
loci <- c(9900, 3300, 500, 100, 50)

Results.table <- Results.subsets.table

a <- which(Results.table[[1]]$Demography != "IM")

tmp <- lapply(1:length(Results.table), function(s)
  data.frame(NumLoci=rep(loci[s],length(a)), Results.table[[s]] %>%
             mutate(Demography = as.character(Demography),
                    NumInd = as.character(NumInd)) %>%
             filter(Demography!= "IM") %>%
             select(Demography, NumPops, NumInd, Fst.90, Fst.RsqAdjPos.90,
                    Fst.Morans.I.90, Fst.I.scaled.90)))

Results.sub.long <- Reduce(rbind, tmp)
Results.sub.long <- Results.sub.long %>%
            mutate(NumLoci = factor(NumLoci, levels = c("50", "100", "500", "3300", "9900"))) %>%
            mutate(NumInd = factor(NumInd, levels = c("20", "6")))
names(Results.sub.long)[5:8] <- c("Fst", "Adj.Rsqr.Pos", "Morans.I", "I.scaled")

pdf(file = paste0(here::here(), "/output/Figure3.pdf"), width=8, height=3)
```

```
par(mfcol=c(1,3), mar = c(5, 5, 3, 1))

Metrics <- c("Fst", "Adj.Rsqr.Pos", "Morans.I")
Labels <- c("Mean Fst", expression("Mean adjusted R"^2), "Mean Moran's I")
ymax <- c(0.015, 0.6, 0.6)
Leg <- c(TRUE, FALSE, FALSE)
Title <- c("(a)", "(b)", "(c)")

for(m in 1:3)
{
  with(Results.sub.long, {interaction.plot(NumInd, NumLoci, get(Metrics[m]),
                            col=c("grey70","grey50","grey30", "grey20", "grey1"),
                            ylab= Labels[m], ylim=c(0.0, ymax[m]), lwd= 1.5,
                            xlab="Number of individuals", cex.lab=1.5, cex.axis=1.5,
                            cex.sub=1.5, legend=Leg[m], xleg = "bottomright")})
  title(Title[m], adj = 0, line =1, cex.main=2 )
}

dev.off()
```

```
## pdf
##   2
```

c) Figure 4

Multipanel figure with one plot for each demography, all on the same scale. Response variables: Adj R^2, Moran's I & scaled Moran's I. The figure is based on 500 loci. Grey symbols denote results for SES.

Function for pooling mean and sd from groups. Source: https://www.statstodo.com/CombineMeansSDs_Pgm.php

```
Calc1 <- function(myDat) # myDat is matrix with 3 cols of n, mean, and SD
{
  m = nrow(myDat)  # number of groups
  tn = 0
  tx = 0
  txx = 0
  for(i in 1:m)
  {
    n = myDat[i,1]
    mean = myDat[i,2]
    sd = myDat[i,3]
    x = n * mean
    xx = sd^2*(n - 1) + x^2 / n
    #out<-cat("grp",i," n=",n," mean=",mean," SD=", sd, " Ex=", x, " Exx=",xx, "\n")
    tn = tn + n
    tx = tx + x
    txx = txx + xx
  }
  tmean = tx / tn
  tsd = sqrt((txx - tx^2/tn) / (tn - 1))
  #out <- cat("Combined","n=",tn," mean=",tmean," SD=", tsd, " Ex=", tx, " Exx=",txx,"\n")
```

```
  c(tn,tmean,tsd)
}
```

Fig. 4

```
Results.table <- Results.subsets.table[[3]] %>% filter(NumInd == 20)
Results.table <- split(Results.table, Results.table$Demography)

#Plots.data <- list()
Plots <- list()

demo <- rep(c("IBD", "1R", "2R"), 3)
y1 <- rep(c("Fst.RsqAdjPos", "Fst.Morans.I", "Fst.I.scaled"), each=3)
y2 <- rep(c(NA, "SES.Morans.I", "SES.I.scaled"), each=3)
yLabels <- c(expression('Adjusted R'^2), "", "",
             "Moran's I", "", "", "Scaled Moran's I", "", "")

for(i in 1:9)
{
  tmp1 <- tmp2 <- Results.table[[demo[i]]]
  tmp1 <- tmp1[ ,grep(y1[i], names(tmp1))]
  names(tmp1) <- paste0("Y.", substr(names(tmp1), nchar(y1[i])+2, nchar(names(tmp1))))
  D11<- with(tmp1, data.frame(Comp.60=Y.drop.m / Y.90,
                      Meta.60=Y.drop.MEM.m / Y.90,
                      Meta.30=Y.30m / Y.90))
  # Pool standard deviation across three replicate landscapes
  sds <- with(tmp1, data.frame(Comp.60.s=Y.drop.s / Y.90,
                      Meta.60.s=Y.drop.MEM.s / Y.90,
                      Meta.30.s=Y.30s / Y.90))
  Pooled <- data.frame(rbind(Comp.60=Calc1(data.frame(n=10, means=D11[,1], sd=sds[,1])),
        Meta.60=Calc1(data.frame(n=10, means=D11[,2], sd=sds[,2])),
        Meta.30=Calc1(data.frame(n=10, means=D11[,3], sd=sds[,3]))))
  names(Pooled) <- c("n", "Mean", "SD")
  Pooled <- data.frame(Comparison = factor(rownames(Pooled), levels=c("Comp.60", "Meta.60", "Meta.30")),
                       Conf.level=0.95, Pooled)

  # Recalculate confidence intervals
  Dist = Pooled$Conf.level + (1 - Pooled$Conf.level)/2
  Inty = qt(Dist, df = (Pooled$n - 1)) * Pooled$SD/sqrt(Pooled$n)
  Pooled$Trad.lower <- Pooled$Mean - Inty
  Pooled$Trad.upper <- Pooled$Mean + Inty

  if(is.na(y2[i]))
  {
    Plots[[i]] <- ggplot(Pooled, aes(x = Comparison, y = Mean)) + ylim(0.5,1.1) +
      geom_errorbar(aes(ymin = Trad.lower, ymax = Trad.upper), width = 0, size  = 0.5) +
      geom_point(shape = 1, size  = 2, colour="black") +
      theme_bw() + theme (panel.grid.major = element_blank(), panel.grid.minor = element_blank()) +
       ggtitle(demo[i]) + xlab("") + ylab(yLabels[i]) +
       theme(axis.text = element_text(size = 10),
             plot.title = element_text(size=16, face="bold", hjust = 0.5),
             axis.title.y = element_text(size=12, face="bold")) +
      geom_hline(yintercept=1, size=0.5)
```

```
  }

  if(!is.na(y2[i]))
  {
    tmp2 <- tmp2[ ,grep(y2[i], names(tmp2))]
    names(tmp2) <- paste0("Y.", substr(names(tmp2), nchar(y2[i])+2, nchar(names(tmp2))))
    D11.SES<- with(tmp2, data.frame(Comp.60=Y.drop.m / Y.90,
                      Meta.60=Y.drop.MEM.m / Y.90,
                      Meta.30=Y.30m / Y.90))

    # Pool standard deviation across three replicate landscapes
    sds <- with(tmp2, data.frame(Comp.60.s=Y.drop.s / Y.90,
                      Meta.60.s=Y.drop.MEM.s / Y.90,
                      Meta.30.s=Y.30s / Y.90))
    Pooled.SES <- data.frame(rbind(Comp.60=Calc1(data.frame(n=10, means=D11.SES[,1], sd=sds[,1])),
        Meta.60=Calc1(data.frame(n=10, means=D11[,2], sd=sds[,2])),
        Meta.30=Calc1(data.frame(n=10, means=D11[,3], sd=sds[,3]))))
    names(Pooled.SES) <- c("n", "Mean", "SD")
    Pooled.SES <- data.frame(Comparison = factor(rownames(Pooled.SES),
                               levels=c("Comp.60", "Meta.60", "Meta.30")),
                             Conf.level=0.95, Pooled.SES)

    # Recalculate confidence intervals
    Dist = Pooled.SES$Conf.level + (1 - Pooled.SES$Conf.level)/2
    Inty = qt(Dist, df = (Pooled.SES$n - 1)) * Pooled.SES$SD/sqrt(Pooled.SES$n)
    Pooled.SES$Trad.lower <- Pooled.SES$Mean - Inty
    Pooled.SES$Trad.upper <- Pooled.SES$Mean + Inty

    # Combine observed means and SES:
    Pooled <- data.frame(Type=rep(c("Obs", "SES"),each=3), rbind(Pooled, Pooled.SES))

    # Group by type, dodge means and error bars:
    Plots[[i]] <- ggplot(Pooled, aes(x = Comparison, y = Mean, group=Type, colour=Type)) +
         ylim(0.5,1.1) + scale_color_manual(values=c("#000000", "#999999")) +
     geom_errorbar(aes(ymin = Trad.lower, ymax = Trad.upper), width = 0.05, size  = 0.5,
                   position=position_dodge(width=0.25), show.legend = F) +
     geom_point(shape = 1, size  = 2, position=position_dodge(width=0.25), show.legend = F) +
     theme_bw() + theme (panel.grid.major = element_blank(), panel.grid.minor = element_blank()) +
      ggtitle(demo[i]) + xlab("") + ylab(yLabels[i]) +
      theme(axis.text = element_text(size = 10),
            plot.title = element_text(size=16, face="bold", hjust = 0.5),
            axis.title.y = element_text(size=12, face="bold")) +
     geom_hline(yintercept=1, size=0.5)
  }
}

g <- arrangeGrob(Plots[[1]], Plots[[2]], Plots[[3]], Plots[[4]], Plots[[5]],
                         Plots[[6]], Plots[[7]], Plots[[8]], Plots[[9]], ncol = 3, nrow = 3)
ggsave(file=paste0(here::here(), "/output/Figure4.pdf"), g, width=8.5, height=9)
```