```
 1: ##########################################################################
 2: ##
 3: ## File Name: 'SampleScript.R'
 4: ##
 5: ## Author:    FHRB Toledo < f.toledo@cgiar.org >
 6: ##
 7: ## Date:      Wed May 15, 2019
 8: ##
 9: ## Source:    Basic pipeline to play with isqg
10: ##            supplemental material for the G3 paper
11: ##
12: ##            ... several -- read comments (# in R and // in C++)
13: ##
14: ##########################################################################
15:
16: rm(list = objects()); ls() # CLEAR 'workspace'             (R  environment)
17:
18: ## NOTE for windows and mac users:
19: ## Be sure you have installed the compilers and libraries
20: ## Please follow the instruction on these links:
21: ## windows: https://cran.r-project.org/bin/windows/Rtools/installer.html
22: ## mac: https://cran.r-project.org/bin/macosx/tools/
23:
24: ## needed packages
25: packs <- c('isqg', 'reshape2', 'plyr', 'Rcpp', 'ggplot2')
26:
27: ## get non installed
28: toinstall <- names(which(!sapply(packs,
29:                                  function(pack) nzchar(system.file(package = pack)))))
30:
31: ## install non installed packages
32: if(length(toinstall) != 0)
33:     install.packages(pkgs = toinstall,
34:                      repos = "https://cloud.r-project.org",
35:                      dependencies = TRUE,
36:                      type = 'source',
37:                      verbose = TRUE)
38:
39: ## load libraries
40: sapply(packs, require, character.only = TRUE)
41:
42: ## load simple map
43: data(ToyMap) # example of input data
44: head(ToyMap, 10) # 2 chromosomes, 42 markers
45:
46: ## start a standard _in silico_ specie
47: ## meiosis follow a count location process:
48: ## ref: Karlin & Liberman (1978) [Proc. Natl. Acad. Sci. 75(12):6332--6336]
49: spp <- set_specie(ToyMap)
50:
51: ## start some 'founders' individuals from specie
52: AA <- spp$founder(code = 'AA')
53: aa <- spp$founder(code = 'aa')
54: ## alternative to generate heterozigous individuals
55: Aa <- spp$founder(code = 'Aa')
56: aA <- spp$founder(code = 'aA')
57:
58: ## sampling a 'random' individual -- can be your real data
59: real <- sample(c('2 2', '2 1', '1 2', '1 1'),
60:                size = nrow(ToyMap), replace = TRUE)
61: names(real) <- ToyMap$snp
62:
63: ## start an individual from outside
64: Gidi <- import(spp, real)
65:
```

```
## start an individual as the complete oposite from someone
Gidj <- Gidi$mirror()

## making some crosses (cross, selfcross and dh)
## are accessible by standalone function and methods
F1 <- cross(n = 1, AA, aa)  # the hybrid
F2 <- F1$selfcross(n = 1E3) # segregation population
RILS <- F1$dh(n = 1E3)      # inbreeds

## chainable application
F1$selfcross(n = 1, replace = TRUE)$selfcross(n = 1, replace = TRUE) # now it is F3!

## retrieving the genotypic data
Mrils <- genotype(RILS)                    # as numeric
Gid_wphase <- Gidj$genotype(phase = TRUE) # as character (phased)

## look for specific locus
## maybe useful for mutation or gene edit
F1$look('s1', phase = TRUE) # with phase
F1$look('s22')              # as numeric [-1/0/1]

## define a infinitesimal trait:
## all loci with the same additive and dominance values
infty <- set_infty(spp, m = 0, a = 1, d = .5) # partial dominance

## sampling some loci to define another trait (quantitative)
genes <- data.frame(snp = sample(ToyMap$snp, size = 10),
                    add = rnorm(10), dom = rnorm(10))

## define a quantitative trait
## given a table with additive and dominance values for each locus
quant <- set_quant(spp, m = 10, data = genes)

## evaluating breeding values regarding traits
## as a method of specimen
alphas_infty <- sapply(F2, function(x) infty$alpha(x))
hist(alphas_infty) ## distribution of the genotypic values

## as a method of trait
alphas_quant <- sapply(RILS, function(x) x$alpha(quant))
hist(alphas_quant) ## distribution of the genotypic values

#########################################################################
## user extensions for meiosis recombination

## define your own meiosis process
## "fixed" meiosis i.e., recombinations will occur at fixed points in the chromosome
FixedSrc <- '
// [[Rcpp::depends(isqg)]]
# include <isqg.h> // loading headers of the package

// fixed point recombination at position the middle of the chromosome
// Map is an alias for std::vector<double>
// Chromosome is a class with several slots/methods
// see inst/include/isqg for details
Map fixed(Chromosome * group) {

  double middle(group->get_length() / 2.) ;

  if (static_cast<bool>(R::rbinom(1.0, 0.5))) {
    return Map() ;
  } else {
    return Map(1, middle) ;
  }

```

```
131: }
132:
133: // wrap the function as external pointer
134: // MPtr is a smart pointer
135: // FPtrM is a function pointer
136: // see inst/include/isqg/ for details
137: // [[Rcpp::export]]
138: MPtr fixedp() { return MPtr(new FPtrM(& fixed), true) ; }
139: '
140:
141: ## compile the extension
142: sourceCpp(code = FixedSrc, rebuild = TRUE)
143:
144: ## define a specie w/ custom meiosis
145: spp_custom <- set_specie(ToyMap, meiosis = fixedp())
146:
147: ## generate some gamete prototypes under your own meiosis
148: spp_custom$gamete(n = 20)
149:
150: ##############################################################################
151: ## Comparing meiosis models
152:
153: ## fixed dimensions for the simulation
154: N <- 100 # n markers
155: n <- 1E4 # n gametes
156: L <- 2 # chromosome length in Morgans
157: C <- 2 # number of chromosomes
158:
159: ## test map with two chromosomes of length 2M with 100 markers evenly spaced
160: map <- data.frame(snp = paste0('s', 1:(N * 2)),
161:                   arrange(expand.grid(chr = 1:C, pos = seq(0, L, length.out = N)),
162:                           chr, pos),
163:                   stringsAsFactors = FALSE)
164:
165: standard <- set_specie(map) # start isqg genome
166: gamS <- standard$gamete(n)  # sampling n gametes
167:
168: ## get gamete as numeric vector
169: breaks <- function(gam) {
170:     vec <- as.numeric(strsplit(gam, '')[[1]])
171:     return(vec)
172: }
173:
174: ## numeric gametes
175: cxS <- laply(gamS, breaks)
176:
177: ## fast way to get frequencies of recombination
178: src <- '
179: // [[Rcpp::plugins(cpp11)]]
180:
181: # include <Rcpp.h>
182: # include <vector>
183: # include <algorithm>
184:
185: // [[Rcpp::export]]
186: std::vector<int> freq2by2(std::vector<int> gamete) {
187:
188:   int dim(gamete.size()) ;
189:   std::vector<int> out(std::pow(dim, 2), -1) ;
190:
191:   for (auto row = 0; row < dim; row++)
192:     for (auto col = 0; col < dim; col++)
193:       out.at(row * dim + col) = gamete.at(row) != gamete.at(col) ;
194:
195:   return out ;
```

```
196:
197: }
198: '
199:
200: ## compiling auxiliary function
201: sourceCpp(code = src, rebuild = TRUE)
202:
203: ## loci by loci recombination frequency
204: frvS <- apply(apply(cxS, 1, freq2by2), 1, mean)
205:
206: ## digest
207: frdS <- mutate(merge(merge(cbind(expand.grid(snpi = map$snp, snpj = map$snp,
208:                                               stringsAsFactors = FALSE),
209:                                  fr = frvS),
210:                            map, by.x = 'snpi', by.y = 'snp'),
211:                      map, by.x = 'snpj', by.y = 'snp'),
212:                ## add a shift for markers in different chromosomes
213:                dist = ifelse(chr.x == chr.y, abs(pos.x - pos.y), L + .25))
214:
215: ## check against Haldane mapping function
216: ## points represent the observed frequency of recombination as function of distance
217: ## the red line is the expected frequency of recombination under Haldane function
218: ggplot(data = frdS, mapping = aes(x = dist, y = fr)) +
219:     geom_point(alpha = 1/10) +
220:     stat_function(mapping = aes(x = dist),
221:                   fun = function(x) (1 - exp(-2 * x)) / 2, # Haldane
222:                   colour = 'red')
223:
224: ## define independent segregation -- 2^k masks are equally probable
225: ## all markers segregates independently from each other
226: Independent <- '
227: // [[Rcpp::depends(isqg)]]
228:
229: # include <isqg.h> // loading headers of the package
230: # include <vector>
231: # include <algorithm>
232:
233: // NOTE:
234: // independent markers
235: Map indep(Chromosome * group) {
236:
237:   Map map(group->get_map()) ;
238:
239:   for (auto it = 0; it < map.size(); it++)
240:     if (static_cast<bool>(R::rbinom(1., .5))) map.at(it) = 2. + 1. ;
241:
242:   map.erase(std::remove(map.begin(), map.end(), 2. + 1.), map.end());
243:
244:   return map ;
245:
246: }
247:
248: // wrap the function as external pointer
249: // [[Rcpp::export]]
250: MPtr indepp() { return MPtr(new FPtrM(& indep), true) ; }
251: '
252:
253: ## compile the extension for independent loci
254: sourceCpp(code = Independent, rebuild = TRUE)
255:
256: custom <- set_specie(map, indepp()) # start isqg genome
257: gamC <- custom$gamete(n)  # sampling n gametes
258:
259: ## numeric gametes
260: cxC <- laply(gamC, breaks)
```

```
261:
262: ## loci by loci recombination frequency
263: frvC <- apply(apply(cxC, 1, freq2by2), 1, mean)
264:
265: ## digest
266: frdC <- mutate(merge(merge(cbind(expand.grid(snpi = map$snp, snpj = map$snp,
267:                                               stringsAsFactors = FALSE),
268:                                    fr = frvC),
269:                               map, by.x = 'snpi', by.y = 'snp'),
270:                         map, by.x = 'snpj', by.y = 'snp'),
271:                ## add a shift for markers in different chromosomes
272:                dist = ifelse(chr.x == chr.y, abs(pos.x - pos.y), L + .25))
273:
274: ## all two by two frequency of recombination are around 0.5 (independent)
275: ## points represents observed frequency of recombination as a funtion of distance in M
276: ggplot(data = frdC, mapping = aes(x = dist, y = fr)) +
277:     geom_point(alpha = 1/10)
278:
279: ## recombination hotspots -- shifting beta random draws
280: ## in this model, crossing over will preferentially occur at hotspots
281: ## it is carried out by means of random draws from a beta distribution
282: Hotspots <- '
283: // [[Rcpp::depends(isqg)]]
284:
285: # include <isqg.h> // loading headers of the package
286: # include <vector>
287: # include <algorithm>
288:
289: // shifted beta distribution
290: // [[Rcpp::export]]
291: Rcpp::NumericVector shiftbeta(int n, double x) {
292:
293:   return x * Rcpp::rbeta(n, 10.0, 2.5) ;
294:
295: }
296:
297: // NOTE:
298: // independent markers
299: Map hotspot(Chromosome * group) {
300:
301:   double length(group->get_length()) ;
302:
303:   int event(static_cast<int>(R::rpois(length))) ;
304:
305:   if ( event == 0 ) {
306:
307:     return Map() ;
308:
309:   } else {
310:
311:     Map chiasmata(Rcpp::as<Map>(shiftbeta(event, length))) ;
312:
313:     std::sort(chiasmata.begin(), chiasmata.end()) ;
314:
315:     return chiasmata ;
316:
317:   }
318:
319: }
320:
321: // wrap the function as external pointer
322: // [[Rcpp::export]]
323: MPtr hotspotp() { return MPtr(new FPtrM(& hotspot), true) ; }
324: '
325:
```

```
## compile the extension for recombination hotspots
sourceCpp(code = Hotspots, rebuild = TRUE)

hotspot <- set_specie(map, hotspotp()) # start isqg genome
gamH <- hotspot$gamete(n)  # sampling n gametes

## numeric gametes
cxH <- laply(gamH, breaks)

## get where crossing happen
cxX <- apply(cxH, 1, function(gam) which(as.logical(diff(gam))))

## digest
Counts <- data.frame(snp = map$snp,
                     chr = map$chr,
                     pos = map$pos,
                     count = hist(unlist(cxX),
                                  breaks = seq(.5, 200.5, 1),
                                  plot = FALSE)$counts,
                     stringsAsFactors = FALSE)

## distribution of crossing over across both chromosomes -- hotspots
## it is clear to see the hotspots of recombination
ggplot(data = subset(Counts, snp != 's100'), # dropping independent assortment!
       mapping = aes(x = pos, y = count)) +
    geom_bar(stat = 'identity') +
    facet_wrap(facets = ~ chr)

#############################################################################
## polymorphisms rather than diploid

## two bits together will have four alleles
droplets <- data.frame(snp = paste0('s', 2),   # two loci
                       chr = 1, pos = c(1, 1), # at the same position
                       stringsAsFactors = FALSE)

## initiate the droplet "specie"
XY <- set_specie(droplets)
XY$gamete(100) # there is no recombination between loci

## first individual has two alleles 11 and 12
i1 <- c('1 1', '1 2')
names(i1) <- droplets$snp
I1 <- import(XY, i1)

## second individual has the other two alleles 22 and 21
i2 <- c('2 2', '2 1')
names(i2) <- droplets$snp
I2 <- import(XY, i2)

## crossing then will have the four alleles segregating with equal proportion
CD <- t(genotype(cross(n = 1000, I1, I2), phase = TRUE))

## checking proportions
table(c(sapply(c(1, 3), # getting DNA tape **bear in mind the space**
               function(t) apply(CD, 1,
                                 function(g) paste(sapply(g, substring,
                                                          first = t,
                                                          last = t),
                                                   collapse = '')))))

## three bits together will have eight alleles
triplets <- data.frame(snp = paste0('s', 3),   # two loci
                       chr = 1, pos = c(1, 1, 1), # at the same position
                       stringsAsFactors = FALSE)
```

```
391:
392: ## initiate the triplet "specie"
393: XYZ <- set_specie(triplets)
394: XYZ$gamete(100) # there is no recombination between loci
395:
396: ## first individual has two alleles 111 and 112
397: i4 <- c('1 1', '1 1', '1 2')
398: names(i4) <- triplets$snp
399: I4 <- import(XYZ, i4)
400:
401: ## second individual has two alleles 121 122
402: i5 <- c('1 1', '2 2', '1 2')
403: names(i5) <- triplets$snp
404: I5 <- import(XYZ, i5)
405:
406: ## third individual has two alleles 211 212
407: i6 <- c('2 2', '1 1', '1 2')
408: names(i6) <- triplets$snp
409: I6 <- import(XYZ, i6)
410:
411: ## forth individual has two alleles 221 222
412: i7 <- c('2 2', '2 2', '1 2')
413: names(i7) <- triplets$snp
414: I7 <- import(XYZ, i7)
415:
416: ## putting _in silico_ genotypes within a list
417: pop <- list(I4, I5, I6, I7)
418:
419: ## function to simulate random crosses between all four individuals
420: random <- function(w) {
421:     pair <- sample(1:4, 2, replace = FALSE)
422:     x <- pop[[pair[1]]]
423:     y <- pop[[pair[2]]]
424:     z <- cross(n = 1, x, y)
425:     return(z)
426: }
427:
428: ## crossing then will have the four alleles segregating with equal proportions
429: CT <- t(genotype(sapply(1:1000, random), phase = TRUE))
430:
431: ## checking proportions
432: table(c(sapply(c(1, 3), # getting DNA tape **bear in mind the space**
433:                function(t) apply(CT, 1,
434:                                  function(g) paste(sapply(g, substring,
435:                                                           first = t,
436:                                                           last = t),
437:                                                    collapse = '')))))
438:
439: ## \EOF
440: ##############################################################################
```