# fishSim-vignette

Shane M Baylis

2024-05-16

fishSim is a package for demographic simulation and exploration of kin relationships. It includes functions to build populations and move, mate, kill, age, and switch the sex of individuals within those populations or defined subpopulations. It *doesn't* include any one-line functions that define and run an entire simulation. Instead, users will have to build their own for-loop over years, with each year containing mating, mortality, ageing, sampling, and/or movement events as needed. With a little planning, this makes fishSim *very* adaptable. Importantly, fishSim is individual-based and retains parentage information for all individuals, allowing full reconstruction of kin relationships to arbitrary depth.

In addition to population simulation features, fishSim includes a set of convenience functions allowing the user to check the scenario population growth rate, find simulation scenarios with null population growth (`PoNG()`), selectively archive subsets of the population for increased computational speed. For a given population, fishSim also includes functions to look up pairs of animals, compare their sets of ancestors, and classify them into kin categories based on their nearest shared ancestor(s), or show patterns of shared ancestors for members of a pair.

fishSim is developed in response to the demands of close-kin mark-recapture projects. 'typical' use-cases therefore involve an understanding of both demography and kin relationships. This vignette will cover two use scenarios - one relatively simple one, with:

- age-specific (but not sex-specific) fecundity,
- constant adult survival,
- a single population with no subdivisions (i.e., one 'stock'),
- no sex-switching, and
- first-year survival set by `PoNG()`, giving a flat population size through time,
- one-shot sampling of animals from the population

. . . and a one designed to show off the bells and whistles, with:

- age-specific, sex-specific fecundity with within-season mating exhaustion for females,
- age-specific, sex-specific survival,
- three subpopulations ('stocks') with markovian movement between subpopulations,
- male-to-female sex-switching,
- a growing population,
- lethal sampling, occurring through time during the simulation.

In the 'bells-and-whistles' scenario, we will also cover archiving tools. For both scenarios, we will take a brief look at relationship patterns between pairs of animals.

## The relatively-simple scenario

### Setup

In order to simulate a population, we need an object to hold that population's data, and the population will need some founding members. Our data object will have one line per individual, so let's call it `indiv`.

```
library(fishSim)
indiv <- makeFounders(stocks = c(1))
head(indiv)
#>                                  Me Sex     Dad     Mum BirthY DeathY Stock AgeLast SampY
#> 1 7c7590396d1b4c63970e15286d7acc4d   M founder founder     -1     NA     1       2    NA
#> 2 12c04f14fc9f42bb8c5efef59aeb6145   F founder founder     -1     NA     1       2    NA
#> 3 9f5aefbe3bcc4e5cb7702b4ee666d0e1   M founder founder     -2     NA     1       3    NA
#> 4 2b5220da215f43d6b4074b52f919fce4   F founder founder     -5     NA     1       6    NA
#> 5 373bb065d9ac4133afc579a75727eb82   F founder founder      0     NA     1       1    NA
#> 6 7b2e4b496c6d49df98322a6620c148a6   M founder founder     -8     NA     1       9    NA
```

Now, what's here? `indiv` has one line per individual, and each individual has an ID code (`indiv$Me`), a sex (`indiv$Sex`), a father's ID code (`indiv$Dad` - but which is 'founder' here, because founding animals can't really be said to have parents), a mother's ID code (`indiv$Mum` - ditto), a birth year `indiv$BirthY`, a death year (`indiv$DeathY` - NA for any animals that are still alive), a stock membership (`indiv$Stock`), an age in years (`indiv$AgeLast`), and a flag indicating whether the individual has been sampled (`indiv$SampY`).

The age structure, sex ratio, and stock memberships of founders can all be specified in the `makeFounders()` call. We've only changed one of the default values - instead of the default 3 stocks, we're going to have a single panmictic founder population. The other defaults give us 1000 founders, with an even sex ratio, with an age-structure that implies 70% annual survival, but with a hard age-limit of 20 years.

### Mating

Having set up our population, we can now get its members to breed, get older, switch sex and die, using the `mate()` (or `altMate()`), `birthdays()`, `sexSwitch()`, and `mort()` functions. If we had more than one stock, we could also get individuals to move between stocks with markovian movement probabilities, using `move()`. Let's look at those.

```
nrow(indiv)
#> [1] 1000
indiv <- mate(indiv, year = 1)
nrow(indiv) ## 200 newborns
#> [1] 1200
tail(indiv) ## newborns are added to the end
#>                                     Me Sex                              Dad
#> 1195 eec9760e9c854fcbbdffa6d62d660968   M a401288ffdfc44ab975ead9bcf208173
#> 1196 5d22783fef464dfa898cbf5496280d93   F a401288ffdfc44ab975ead9bcf208173
#> 1197 34e39471845e44908648715fb3f6764d   F 6b2a03d536e8450a9581beaabd6eb049
#> 1198 896c0a3ce85949f584e3dcc41571747e   F fc0deb39a1194b18a63305c5943ebb6e
#> 1199 ea89f7e08b1c4236801034ae015bf527   F 314715664c8b41a8885a2ba22b7217de
#> 1200 fb57511806db4990879c22c21b5574f4   M b15554bbfd394354a6da7c8da2a715c6
#>                                    Mum BirthY DeathY Stock AgeLast SampY
#> 1195 5059c88a7db64163b01db50bf40ebf3c      1     NA     1       0    NA
#> 1196 5059c88a7db64163b01db50bf40ebf3c      1     NA     1       0    NA
#> 1197 650846974b584e79ad64815c4b1b7e13      1     NA     1       0    NA
#> 1198 75d2d6fb006c415baebfc9f3c62b325d      1     NA     1        0     NA
#> 1199 007bddd8b6cc4fc7a45e0e383d6b5bd4      1     NA     1        0     NA
#> 1200 bb84abeae4e34df6830000442e38976e      1     NA     1        0     NA
```

Here are some newborns, generated using the default `mate()` settings. Unlike founders, these newborns all have a father's ID and a mother's ID, are all aged 0, and all have a birth year of equal to the `year` argument. There are 200 newborns, because `mate()` generates new members as a proportion of the extant population size (i.e., we specify fecundity for the population, not the individuals), and that proportion (argument `fecundity`) is 0.2 by default. Each individual was born in a clutch, and each clutch contained a Poisson-distributed number of clutch-members, the default mean ('batchSize') of 0.5. `mate()` simply kept producing newborns by mating together random male/female pairs until `nrow(indiv) * 0.2` new offspring were produced.

Alternatively, we can mate our population using `altMate()`, which specifies individual maturities by age, and a probability distribution of number of offspring per mature female. Let's generate some new founders and try that.

```
indiv <- makeFounders(stocks = c(1))
nrow(indiv) ## 1000 founders
#> [1] 1000
indiv <- altMate(indiv, firstBreed = 2, batchSize = 0.9, year = 1)
tail(indiv)
#>                                    Me Sex                               Dad
#> 3681 10766d3293314aaea69fe71c760988d9   F f8d76284b33743f9a84ae4b7e5729e14
#> 3691 862e354063694 79bab5a1e6ea8b276cd   F f8d76284b33743f9a84ae4b7e5729e14
#> 3701 ecf2f4adf266460c85afe4b230f2b879   F 29dd4945397544c18ba8db4f08fff906
#> 3711 e89261902c484fd993da93cb1011a5c8   F 29dd4945397544c18ba8db4f08fff906
#> 3721 4abb46a5e6ce45a49e778e7247ec3f1c   F 31adb63093a84d69a4a05f9a241a6f2a
#> 3731 836a2750ae1b4eeb9e7dae39074f76e8   F f5a68d29579346959f4536e5ef241ba1
#>                                   Mum BirthY DeathY Stock AgeLast SampY
#> 3681 a97fe4c4546a46688871b58be69a1dd0      1     NA     1       0    NA
#> 3691 a97fe4c4546a46688871b58be69a1dd0      1     NA     1       0    NA
#> 3701 706d9e5cf98f40cf9d122727de1489db      1     NA     1       0    NA
#> 3711 706d9e5cf98f40cf9d122727de1489db      1     NA     1       0    NA
#> 3721 ec527df2652942dbac7b737a5929a081      1     NA     1       0    NA
#> 3731 111be1d89c40433a94e296db9bd108be      1     NA     1       0    NA
```

In this call to `altMate()`, females became sexually mature at 2 years of age (`firstBreed = 2`), and each sexually-mature female produced a Poisson-distributed number of offspring with mean set by `batchSize`. It is possible to specify age-specific or sex-specific fecundities, set paternity within-clutch to be single or multiple, to exhaust fathers within each breeding season (females, in this system, always breed to exhaustion each season), and to set a sex ratio for offspring by additional arguments to `altMate()`, and these options can also be set in `mate()`.

## Mortality

Now, let's kill some of the population. Mortality probabilities can be flat, age-specific or stock-specific (and with a little extra effort, sex- or age:stock:sex-specific too), or we can randomly kill animals such that the population is reduced to a certain size, and we can set an age past which no animal will survive. For now, let's set a flat 20% mortality rate (i.e., the probability of death is equal for all animals).

```
nrow(indiv)
#> [1] 1373
indiv <- mort(indiv, year = 1, type = "flat", mortRate = 0.2)
nrow(indiv)
#> [1] 1373
head(indiv, n = 15)
#>                                  Me Sex     Dad     Mum BirthY DeathY Stock AgeLast SampY
#> 1  7e39713a70314614b7b69a11604bc5c6   M founder founder     -2     NA     1       3    NA
#> 2  854e9e80ce5c4636ae1728e56bc91b76   M founder founder     -1     NA     1       2    NA
#> 3  afebfd65acf046d392b0969c61fd7924   F founder founder     -4     NA     1       5    NA
#> 4  37a98ad9847d482187f1fb5b506e7e68   F founder founder     -2     NA     1       3    NA
#> 5  fe91cea8e7b148a4b923729bc0f43400   M founder founder     -4     NA     1       5    NA
#> 6  54c1f4f920dc4e1aaecb17d774aacc0b   F founder founder     -3     NA     1       4    NA
#> 7  fe69da15f63f4e3aac28761b71e0c264   F founder founder     -2     NA     1       3    NA
#> 8  627a0fccce84422b92d748f6da920db4   M founder founder     -8     NA     1       9    NA
#> 9  3c9d65ee38674763adbf3e04c9bd58c6   F founder founder     -1     NA     1       2    NA
#> 10 fbcc319742f34a57be507926510bdc71   F founder founder      0     NA     1       1    NA
#> 11 10c2e8812ce946809d6acbb2ddef6ab7   M founder founder     -8     NA     1       9    NA
#> 12 6d1b0cb153554047a53f4adaa398dcff   F founder founder      0     NA     1       1    NA
#> 13 f24da7cd719e43ea86022e8ed68cea61   F founder founder     -3     NA     1       4    NA
#> 14 65006030f4524045938baa5c14f02a2f   F founder founder     -4      1     1       5    NA
#> 15 68a71c85ac3649ce9f9a823ed4058467   M founder founder      0      1     1       1    NA
```

Mortality does not remove dead animals from ‘indiv’ - we will need to refer to them later - it just updates their death year to the value given in `year`. In general, though, animals with a non-NA value for death year will not move, mate, switch sex, or have birthdays.

## Birthdays

On the topic of birthdays: that’s the final thing that must be done before we can put this all together and run a full (albeit basic) demographic simulation: we need to be able to increase the age of our animals, and that is what the `birthdays()` function is for. It’s very simple: it takes all living members of the population, and increments their age by 1.

```
tail(indiv)
#>                                    Me Sex                              Dad
#> 3681 10766d3293314aaea69fe71c760988d9   F f8d76284b33743f9a84ae4b7e5729e14
#> 3691 862e35406369479bab5a1e6ea8b276cd   F f8d76284b33743f9a84ae4b7e5729e14
#> 3701 ecf2f4adf266460c85afe4b230f2b879   F 29dd4945397544c18ba8db4f08fff906
#> 3711 e89261902c484fd993da93cb1011a5c8   F 29dd4945397544c18ba8db4f08fff906
#> 3721 4abb46a5e6ce45a49e778e7247ec3f1c   F 31adb63093a84d69a4a05f9a241a6f2a
#> 3731 836a2750ae1b4eeb9e7dae39074f76e8   F f5a68d29579346959f4536e5ef241ba1
#>                                   Mum BirthY DeathY Stock AgeLast SampY
#> 3681 a97fe4c4546a46688871b58be69a1dd0      1     NA     1       0    NA
#> 3691 a97fe4c4546a46688871b58be69a1dd0      1     NA     1       0    NA
#> 3701 706d9e5cf98f40cf9d122727de1489db      1     NA     1       0    NA
#> 3711 706d9e5cf98f40cf9d122727de1489db      1     NA     1       0    NA
#> 3721 ec527df2652942dbac7b737a5929a081      1     NA     1       0    NA
#> 3731 111be1d89c40433a94e296db9bd108be      1     NA     1       0    NA
indiv <- birthdays(indiv)
tail(indiv)
#>                                    Me Sex                              Dad
#> 3681 10766d3293314aaea69fe71c760988d9   F f8d76284b33743f9a84ae4b7e5729e14
#> 3691 862e354063694 79bab5a1e6ea8b276cd   F f8d76284b33743f9a84ae4b7e5729e14
#> 3701 ecf2f4adf266460c85afe4b230f2b879   F 29dd4945397544c18ba8db4f08fff906
#> 3711 e89261902c484fd993da93cb1011a5c8   F 29dd4945397544c18ba8db4f08fff906
#> 3721 4abb46a5e6ce45a49e778e7247ec3f1c   F 31adb63093a84d69a4a05f9a241a6f2a
#> 3731 836a2750ae1b4eeb9e7dae39074f76e8   F f5a68d29579346959f4536e5ef241ba1
#>                                    Mum BirthY DeathY Stock AgeLast SampY
#> 3681 a97fe4c4546a46688871b58be69a1dd0      1     NA     1       1    NA
#> 3691 a97fe4c4546a46688871b58be69a1dd0      1     NA     1       1    NA
#> 3701 706d9e5cf98f40cf9d122727de1489db      1     NA     1       1    NA
#> 3711 706d9e5cf98f40cf9d122727de1489db      1     NA     1       1    NA
#> 3721 ec527df2652942dbac7b737a5929a081      1     NA     1       1    NA
#> 3731 111be1d89c40433a94e296db9bd108be      1     NA     1       1    NA
```

## Sampling from the population

It is often useful to keep track of which animals in the population have been captured, and if capture is lethal, to mark captured animals as dead through a separate process from the normal 'mort' functions. The function `capture()` takes a sample of individuals, marks them as 'captured' by updating their 'captured' value, and, if capture is lethal, marks them as dead by updating their death year, as in `mort()`. It is possible to make `capture()` sex-specific, age-specific, or age:sex specific, so that only one sex may be captured, only one age-class may be captured, or only one sex at one age-class may be captured, respectively.

```
## non-lethal sampling of 5 females
indiv <- capture(indiv, n = 5, year = 1, fatal = FALSE, sex = "F")
## lethal sampling of 5 animals, from either sex
indiv <- capture(indiv, n = 5, year = 1, fatal = TRUE)
## non-lethal sampling of 5 one-year-old females
indiv <- capture(indiv, n = 5, year = 1, fatal = FALSE, sex = "F", age = 1)
```

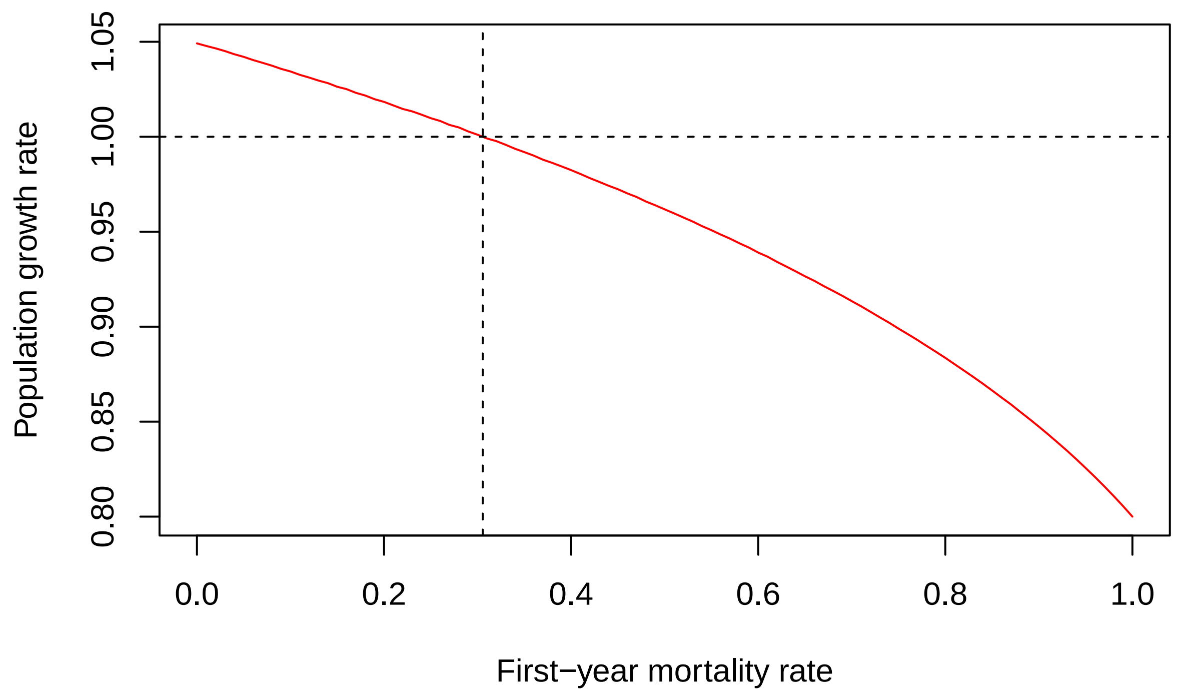
## What's my population doing? Can I make it stay the same size?

A couple more functions deserve a mention here: `check_growthrate()` and `PoNG()`. `check_growthrate()` tells you how quickly your population is growing. `PoNG()` tells you what you need first-year survival to be, in order for your population to maintain a constant size (within limits - the estimation is Leslie Matrix-based, and there are ways to fool Leslie Matrices that are out of scope for this vignette but covered briefly in the `PoNG()` and `check_growthrate()` documentation). Let's try both of those, using the mating and mortality settings we have already used.

```
check_growthrate(mateType = "flat", mortType = "flat",
                 batchSize = 0.9, firstBreed = 2, mortRate = 0.2)
#> [1] 1.018176+0i

## with the current settings, we expect our population to grow by about 1.8% annually.

PoNG(mateType = "flat", mortType = "flat", batchSize = 0.9, firstBreed = 2,
     mortRate = 0.2)
```



```
#> $root
#> [1] 0.3054975
#>
#> $f.root
#> [1] -0.0001832548
#>
#> $iter
#> [1] 8
#>
#> $init.it
#> [1] NA
#>
#> $estim.prec
#> [1] 0.0005394719

## if first-year mortality was about 0.305, rather than 0.2, our population
## would have null growth. You can also read off a range of possible growth rates
## from the plot.
```

## Turning processes into a simple simulation

To turn those processes into a full demographic simulation, all that is needed is to repeat the processes in a loop. Let's do that, but set age-specific mortality so that the long-run average population growth rate is zero using the first-year mortality rate we got from `PoNG()`, so that our population is unlikely to explode or become extinct.

```
indiv <- makeFounders(stocks = c(1))
ageMort <- c(0.305, rep(0.2, 100)) ## age-specific mortality is 0.305 for first-years,
## 0.2 for all older age-classes.

for (y in 1:60) {
    indiv <- altMate(indiv, firstBreed = 2, batchSize = 0.9, year = y) ## y for year
    indiv <- mort(indiv, year = y, type = "age", ageMort = ageMort)    ## age-specific mort
    indiv <- birthdays(indiv)
}
tail(indiv) ## a population with 60 years of births, deaths, and birthdays
#>                                       Me Sex                              Dad
#> 34071  2d7e6478f73f474f9210edda9b809770   M 980b3e5d4a034343b599934f19f54c63
#> 34081  28b977e465e14a778050ee64701a5e3c   M c903f6cb6cd3491ab02d792df83e2111
#> 34091  f86587debb8c4160aff8489bc2b03142   M c903f6cb6cd3491ab02d792df83e2111
#> 341010 d3ed2fb5ca7c474883a37043b1d23ea1   M f76106e04c16450ba17a36abc3723cbb
#> 341110 e63669fa998a4fa58ba46869fe63be4e   M f76106e04c16450ba17a36abc3723cbb
#> 341210 fe70974d1d6d42e8998b7b039c4b119b   M d710cfd29b6241b99cffe64d63b03a93
#>                                      Mum BirthY DeathY Stock AgeLast SampY
#> 34071  1f35c1fb9fc3442391f8006b2a119b06     60     60     1       0    NA
#> 34081  210112420bed4345b1a3f3acd612c3ce     60     60     1       0    NA
#> 34091  210112420bed4345b1a3f3acd612c3ce     60     60     1       0    NA
#> 341010 6bff2e18a1b543cf9f886ee942c2f998     60     NA     1       1    NA
#> 341110 6bff2e18a1b543cf9f886ee942c2f998     60     NA     1       1    NA
#> 341210 c9c3403987464fa4b598e89f162e9b19     60     NA     1       1    NA
nrow(indiv[is.na(indiv$DeathY),]) ## the currently-alive population size. Note that population
#> [1] 10400
                                ## growth only *averages* zero, and variability occurs!
```

## Looking up relationships between pairs of animals

One of the key advantages of fishSim is its ability to report on different relationships between pairs of animals in a sample. Are these two each other's siblings? Half-siblings? Is one the parent of the other? The grandparent? Some obscure half-cousin, once removed?

Internally, these relationships are stored in terms of 'shared ancestors at the n'th generation of each member', starting with the 'self' as generation 1. So a pair might have a shared `pairs$TwoThree` ancestor - that ancestor is the parent of one member, and the grandparent of the other. Or the pair might have a shared `pairs$OneTwo` ancestor - in which case, one member's *self* is the other member's parent.

If a pair shares a parent (i.e., `pairs$TwoTwo == 1`), then they also share at least two grandparents (`pairs$ThreeThree >= 2`), four great-grandparents (`pairs$FourFour >= 4`), and so on. This pattern holds generally: a half thiatic pair (`pairs$TwoThree == 1`) will also share at least two `pairs$ThreeFour >= 2` ancestors and at least four `pairs$FourFive >= 4` ancestors. It is also possible, for example, for a pair to share two grandparents, but no parents - that is the case for full cousin pairs.

```
## mark 200 individuals alive at the end of the simulation as 'captured'.
indiv <- capture(indiv, n = 200, year = 60, fatal = FALSE)

## look up each animal's ancestors and look for shared ancestors between each
## pair of sampled animals:
pairs <- findRelativesPar(indiv = indiv, sampled = TRUE, nCores = 2)
#>   Sample.Year Freq
#> 1          60  200
#> [1] "parents found at 2024-05-16 13:13:35.972276"
#> [1] "grandparents found at  2024-05-16 13:13:36.287441"
#> [1] "great-grandparents found at 2024-05-16 13:13:36.963416"
#> [1] "great-great-grandparents found at 2024-05-16 13:13:38.581548"
#> [1] "great-great-great-grandparents found at 2024-05-16 13:13:42.019572"
#> [1] "great-great-great-great-grandparents found at 2024-05-16 13:13:48.439574"
#> 1000 of 19900 comparisons2000 of 19900 comparisons3000 of 19900 comparisons4000 of 19900 comparisons

POPs <- pairs[pairs$OneTwo == 1,] ## Parent-Offspring pairs
GGPs <- pairs[pairs$OneThree == 1,] ## Grandparent-Grandoffspring pairs
HSPs <- pairs[pairs$TwoTwo == 1,] ## Half-sibling pairs
FSPs <- pairs[pairs$TwoTwo == 2,] ## Full Sibling pairs (self-comparisons
## are automatically excluded)
FCPs <- pairs[pairs$ThreeThree == 2 & pairs$TwoTwo != 1,] ## Full Cousin pairs

## look at the number of shared ancestors at each ancestral
## generation, for one of the half-sibling pairs.
lookAtPair(HSPs[1,])
#>      Self Par GP G2P G3P G4P G5P
#> Self    .   .  .   .   .   .   .
#> Par     .   1  .   .   .   .   .
#> GP      .   .  2   .   .   .   .
#> G2P     .   .  .   4   .   .   .
#> G3P     .   .  .   .   8   .   .
#> G4P     .   .  .   .   .  16   .
#> G5P     .   .  .   .   .   .  32

relatives <- namedRelatives(pairs) ## shows the number of pairs of each relationship type
relatives
#>   POPs GGPs dGGPs G4Ps dG4Ps G6Ps dG6Ps HSPs FSPs HTPs FTPs HCPs FCPs GHCPs GFCPs  ORCs
#> 1    4    1     0    0     0    2     0    4    2   15    1   17    1    20     0 13114
```

## The ‘bells and whistles’ scenario

In this second scenario, we will simulate a metapopulation with:

- age-specific, sex-specific fecundity with within-season mating exhaustion for females,
- stock-specific, age-specific survival,
- three subpopulations (‘stocks’) with markovian movement between subpopulations,
- male-to-female sex-switching,
- a growing population.
- archiving for speed (?)

We set up the population basically as before, with a couple of slight tweaks. First, we should make sure the founding population has multiple stocks (we’ll give it three). Second, we will set up an archive matrix. The archive matrix is intended to hold the records of dead animals in large, long-running simulations: dead animals do not take part in any further mating, movement, sex-switching, aging, or mortality events, but if they stay in `indiv`, `indiv` can become huge and unweildy, slowing down all of those processes. There is of course a trade-off, in that writing dead animals to the archive takes system time, so the optimum may be to only archive dead animals once every few ‘years’.

### Setting up the data objects

```
## set up founders with three stocks: two that each contain 30% of the population,
## and one that contains the remaining 40%.

indiv <- makeFounders(pop = 1000, stocks = c(0.3, 0.3, 0.4))

## set up archive - just a matrix with zero rows and eight columns

archive <- make_archive()
```

### Parameterising movement, survival, and maturity structures

Because we will have inter-stock movement in this sim, we will also need to set up a matrix giving the probability that an individual will move into another stock, given its current stock membership. Because survival will be age- and stock-dependent, we will need a matrix of survival rates with as many columns as stocks, and as many rows as (at least) the age of the oldest plausible population-member. Because we will have age-specific, sex-specific fecundity, we will need a male age-specific maturity curve and a female age-specific maturity curve. Let’s set those up now.

```
## Markovian movement matrix
stocks <- c(0.3, 0.3, 0.4)
admix.m <- matrix(NA, nrow = length(stocks), ncol = length(stocks))
for(i in 1:nrow(admix.m)) {
    admix.m[i,] <- stocks*stocks[i]
}
## admix.m shows movement proportional to starting population sizes.
admix.m
#>      [,1] [,2] [,3]
#> [1,] 0.09 0.09 0.12
#> [2,] 0.09 0.09 0.12
#> [3,] 0.12 0.12 0.16
## let's tweak those numbers so that animals tend to stay where
## they are, and not move around so much.
admix.m <- matrix(c(0.23, 0.03, 0.04, 0.03, 0.23, 0.04, 0.04, 0.04, 0.32),
                  nrow = length(stocks), ncol = length(stocks), byrow = FALSE)
admix.m
#>      [,1] [,2] [,3]
#> [1,] 0.23 0.03 0.04
#> [2,] 0.03 0.23 0.04
#> [3,] 0.04 0.04 0.32

## Age- and stock-dependent survival
ageStockMort <- matrix(c(0.47, 0.37, 0.27, rep(0.23, 97),
                         0.45, 0.35, 0.25, 0.20, rep(0.22, 96),
            0.45, 0.3, 0.3, 0.19, rep(0.2, 96)),
                       ncol = length(stocks), nrow = 100)
head(ageStockMort)
#>      [,1] [,2] [,3]
#> [1,] 0.47 0.45 0.45
#> [2,] 0.37 0.35 0.30
#> [3,] 0.27 0.25 0.30
#> [4,] 0.23 0.20 0.19
#> [5,] 0.23 0.22 0.20
#> [6,] 0.23 0.22 0.20

## Sex-specific maturity curves

maleCurve <- c(0,0,0,0.1,0.5,0.8,0.85,0.9,0.95,rep(1, 91))
femaleCurve <- c(0,0,0.5,0.9,0.95,rep(1,95))
## maleCurve and femaleCurve should both be long enough that no individuals
## will outlive the curve.
head(maleCurve)
#> [1] 0.0 0.0 0.0 0.1 0.5 0.8
head(femaleCurve)
#> [1] 0.00 0.00 0.50 0.90 0.95 1.00
```

## Checking population growth rates with the new parameters

Now, what's going to happen with these subpopulations, if we run with those parameters?

```
check_growthrate(mateType = "ageSex", mortType = "ageStock", batchSize = 1.6,
                 femaleCurve = femaleCurve,
                 ageStockMort = ageStockMort)
#> [1] 0.9932222+0i 1.0147919+0i 1.0270640+0i
## Not bad. Two of the three populations are increasing. The third will probably be kept viable
## by immigration from the other two. Note that I fiddled 'batchSize' (which is the mean number
## of offspring per mature female per breeding attempt) a bit.
```

## Running the simulation

Now we just run the population simulator, as we did for the simple population, but with a couple of extra things to specify. Let's give it 100 years.

```
for (k in 1:100) {
    ## very rarely, switch some males to females
    indiv <- sexSwitch(indiv = indiv, direction = "MF", prob = 1e-04)
    ## move animals according to the markovian matrix we set up before
    indiv <- move(indiv = indiv, moveMat = admix.m)
    ## mate animals using the age-specific, sex-specific curves we set up before
    indiv <- altMate(indiv = indiv, batchSize = 1.6, type = "ageSex", maleCurve = maleCurve,
                     femaleCurve = femaleCurve, year = k)
    ## kill animals on the basis of their age and stock, as set up before
    indiv <- mort(indiv = indiv, year = k, type = "ageStock", ageStockMort = ageStockMort)
    if(k %% 10 == 0) {
        archive <- archive_dead(indiv = indiv, archive = archive)
        indiv <- remove_dead(indiv = indiv)
        cat( sprintf( '\rIteration %i complete', k))
    ## occasionally sample from the population and do some clean-up. And report on progress.
    }
    if(k %in% c(94:100)) {
    indiv <- capture(indiv, n = 40, fatal = TRUE, year = k)
    }
    indiv <- birthdays(indiv = indiv)
}
#> Iteration 10 completeIteration 20 completeIteration 30 completeIteration 40 completeIteration 50 com
archive <- rbind(archive, indiv)
indiv <- archive  ## merge 'indiv' and 'archive', since they were only separated for speed.
```

## Looking up relationships between pairs of animals

Now that the simulation is finished, examining kin relationships is the nearly same as the simple scenario. The only difference comes from the fact that we we integrated multiple years of sample-capture into the simulation, so we don't need to call `capture()` as a separate operation.

```
## look up each animal's ancestors and look for shared ancestors between each
## pair of sampled animals:
pairs <- findRelativesPar(indiv = indiv, sampled = TRUE, nCores = 2)
#>   Sample.Year Freq
#> 1          94   40
#> 2          95   40
#> 3          96   40
#> 4          97   40
#> 5          98   40
#> 6          99   40
#> 7         100   40
#> [1] "parents found at 2024-05-16 13:14:26.7354"
#> [1] "grandparents found at  2024-05-16 13:14:27.07036"
#> [1] "great-grandparents found at 2024-05-16 13:14:27.728694"
#> [1] "great-great-grandparents found at 2024-05-16 13:14:29.05722"
#> [1] "great-great-great-grandparents found at 2024-05-16 13:14:31.801815"
#> [1] "great-great-great-great-grandparents found at 2024-05-16 13:14:37.055712"
#> 1000 of 39060 comparisons2000 of 39060 comparisons3000 of 39060 comparisons4000 of 39060 comparisons

POPs <- pairs[pairs$OneTwo == 1,] ## Parent-Offspring pairs
GGPs <- pairs[pairs$OneThree == 1,] ## Grandparent-Grandoffspring pairs
```

```
HSPs <- pairs[pairs$TwoTwo == 1,] ## Half-sibling pairs
FSPs <- pairs[pairs$TwoTwo == 2,] ## Full Sibling pairs (self-comparisons
## are automatically excluded)
FCPs <- pairs[pairs$ThreeThree == 2 & pairs$TwoTwo != 1,] ## Full Cousin pairs

## look at the number of shared ancestors at each ancestral
## generation, for one of the half-sibling pairs.
lookAtPair(HSPs[1,])
#>      Self Par GP G2P G3P G4P G5P
#> Self    .   .  .   .   .   .   .
#> Par     .   1  .   .   .   .   .
#> GP      .   .  2   .   .   .   .
#> G2P     .   .  .   4   .   .   .
#> G3P     .   .  .   .   8   1   .
#> G4P     .   .  .   .   1  16   3
#> G5P     .   .  .   .   .   3  32

relatives <- namedRelatives(pairs) ## shows the number of pairs of each relationship type
relatives
#>   POPs GGPs dGGPs G4Ps dG4Ps G6Ps dG6Ps HSPs FSPs HTPs FTPs HCPs FCPs GHCPs GFCPs  ORCs
#> 1    9   11     0    5     0    2     0   22    3   62    4  104    1    46     1 33185
```

# Addenda

## A note on mark-dependent survival or movement

It is possible to simulate mark-dependent survival or movement by working on subsets of the 'indiv' matrix within the main simulation loop, and applying different mortality or movement operations to each subset. Note that if `mate()`, `altMate()`, `birthdays()`, etc., are run on a subset of the population, the operation will only affect members of that subset. In the case of subsets for marked and unmarked individuals, this would imply that marked animals only mate with other marked animals, and *vice versa*. It is important to ensure that a combined `indiv` object is available for any operations that affect the whole population, or to repeat each opertation for each subset.

```r
indiv <- makeFounders(stocks = c(1))
ageMort <- c(0.305, rep(0.2, 100)) ## age-specific mortality is 0.305 for first-years,
## 0.2 for all older age-classes.

for (y in 1:60) {
    indiv <- altMate(indiv, firstBreed = 2, batchSize = 0.9, year = y)
    ## 'mate' happens first, affecting the whole population

    indiv <- capture(indiv, n = 5, year = y, fatal = FALSE)
    indiv_unmarked <- indiv[is.na(indiv$SampY),] ## all the unmarked animals
    indiv_marked <- indiv[!is.na(indiv$SampY),]  ## all the marked animals

    indiv_unmarked <- mort(indiv_unmarked, year = y, type = "age", ageMort = ageMort)
        ## normal mort for unmarked animals
    indiv_marked <- mort(indiv_marked, year = y, type = "age", ageMort = ageMort*2)
        ## double mort for marked animals
    indiv <- rbind(indiv_unmarked, indiv_marked)
        ## stick them back together and proceed as normal
    rm(indiv_marked, indiv_unmarked) ## clean up

    indiv <- birthdays(indiv)
}
```

## Arbitrary variables to affect the simulation

Capture status isn't the only variable that can influence survival or movement. It is possible to add arbitrary additional variables to an `indiv` object in order to keep track of anything you might need to know about an individual, and it is possible to make simulation steps depend on those variables. You might want a 'hasBred' column, for instance, keeping track of which animals have produced offspring. Or, alongside a custom 'myMate' function (see 'custom functions', below), you might want to keep track of each animal's pair-bonded mate, so that animals can form pair bonds that persist across years. Here's an example of a species with two morphs, where the spiny morph has a slightly different movement-pattern to the smooth morph:

```r
stocks <- c(0.3, 0.3, 0.4)
indiv <- makeFounders( stocks = stocks)
indiv[,10] <- sample(c("spiny", "smooth"), nrow(indiv), replace = TRUE)
## an extra column to hold stuff. Extras
## _must_ be in column numbers 10 or greater
colnames(indiv)[10] <- "Morph"
```

```r
## a movement matrix for spiny individuals
admix.spiny <- matrix(NA, nrow = length(stocks), ncol = length(stocks))
for(i in 1:nrow(admix.spiny)) {
    admix.spiny[i,] <- stocks*stocks[i]
}
## admix.spiny shows movement proportional to starting population sizes.
admix.spiny
#>      [,1] [,2] [,3]
#> [1,] 0.09 0.09 0.12
#> [2,] 0.09 0.09 0.12
#> [3,] 0.12 0.12 0.16

## a movement matrix for smooth individuals - let's make them more sedentary,
## with 90% chance of staying where they are and 5% chance of moving to each
## other stock
admix.smooth <- matrix(c(0.05), nrow = length(stocks), ncol = length(stocks), byrow = FALSE)
diag(admix.smooth) <- 0.9
admix.smooth <- admix.smooth / sum(admix.smooth) ## standardise so the matrix sums to 1
admix.smooth
#>            [,1]       [,2]       [,3]
#> [1,] 0.30000000 0.01666667 0.01666667
#> [2,] 0.01666667 0.30000000 0.01666667
#> [3,] 0.01666667 0.01666667 0.30000000

for (y in 1:20) {
    indiv <- altMate(indiv, firstBreed = 2, batchSize = 0.9, year = y)
    ## 'mate' happens first, affecting the whole population

    ## newborns need a 'smooth' or 'spiny' status, because altMate won't give one
    indiv$Morph[ is.na( indiv$Morph)] <- sample(c("spiny", "smooth"),
                                          nrow(indiv[ is.na( indiv$Morph),]), replace = TRUE)

    indiv <- mort(indiv, year = y, type = "flat", mortRate = 0.2)
    ## then 'mort', also affecting the whole population

    indiv_smooth <- indiv[indiv$Morph == "smooth",] ## all the smooth animals
    indiv_spiny <- indiv[indiv$Morph == "spiny",]  ## all the spiny animals

    indiv_smooth <- move(indiv_smooth, moveMat = admix.smooth)
        ## one movement pattern for smooth animals
    indiv_spiny <- move(indiv_spiny, moveMat = admix.spiny)
        ## another movement pattern for spiny animals
    indiv <- rbind(indiv_smooth, indiv_spiny)
        ## stick them back together and proceed as normal
    rm(indiv_smooth, indiv_spiny) ## clean up

    indiv <- birthdays(indiv)
}
```

## A note on custom functions in fishSim

The objects used in `fishSim` are deliberately kept simple, to encourage hacking wherever hacking is useful. To that end, most functions in `fishSim` both input and output a `data.frame` that we have called `indiv` throughout this vignette.

If your species of interest has a life-history not readily captured by the existing `fishSim` functions, you may wish to write a custom function for just that step in the simulation. For instance, you may have a species in which each female mates with a handful of males in the year preceding each breeding season, and then uses stored sperm from these mates to fertilise a clutch of eggs just before giving birth. Let's call this the 'sperm storage' scenario. The sperm storage scenario can't be coded using the built-in `mate` or `altMate` functions - in those functions, each mating event either has single paternity (i.e., all offspring produced by each female have the same father) or completely random paternity (i.e., the father of each offspring is randomly selected from among all mature males in the same stock), or, using `mate`, within-clutch single paternity but potentially multiple clutches (each with a different father). In the situations handled by `mate` and `altMate`, the expected number of offspring for a female is linearly related to her number of mates, which doesn't fit with our 'sperm storage' scenario.

To get around this problem, we will need to define a custom `mate`-like function, that inputs an `indiv`-like `data.frame`, generates new offspring the way the scenario demands, and outputs an `indiv`-like `data.frame` to be used by the other functions in `fishSim`. Such a function could look like this:

```r
# A custom myMate function
#
# parameter 'indiv' is a standard indiv frame, as from makeFounders()
#
# parameter 'batchSize' sets the number of offspring per breeding female
# such that the number of offspring ~Poisson(lambda = batchSize)
#
# parameter 'breedProb' is the probability that a mature female will
# breed this in this mating event
#
# parameter 'maturityAge' is the knife-edge age at maturity, for both
# males and females
#
# parameter 'year' is the year. It is used to set the $BirthY for new
# recruits
#
# myMate uses a zero-truncated Poisson distribution to determine the
# number of mates for each female that breeds. Parameter 'meanMates'
# sets the T parameter for that distribution

myMate <- function(indiv, batchSize, breedProb, maturityAge, year, meanMates) {

    library(ids)

    ## select mature females and males at current time-step
    matureFemales <- indiv[indiv$Sex == "F" & indiv$AgeLast >= maturityAge & is.na(indiv$DeathY) ,]
    matureMales <- indiv[indiv$Sex == "M" & indiv$AgeLast >= maturityAge & is.na(indiv$DeathY),]
    ## select subset of mature females that will breed (given breedProb)
    iBreed_F <- matureFemales[sample(1:nrow(matureFemales), size = nrow(matureFemales)*breedProb),]
    iBreed_M <- matureMales

    ## if no mature females or males available, don't add new individuals to the population
    if(nrow(iBreed_F) == 0 | nrow(iBreed_M) == 0) { return(indiv) }
    clutchSize <- rpois(nrow(iBreed_F), batchSize) # how many offsprings per breeding females?
    nMates <- rTruncPoisson(nrow(iBreed_F), T = meanMates) # how many mates per breeding females?
    mates <- lapply(nMates, function(i) sample(iBreed_M$Me, i, replace = FALSE)) # draw mates for each 
    # and then assign fathers to each new recruit based on mates identified previously
    # (note each item in the list corresponds to a unique female)
    dads <- lapply(1:length(mates), function(i) sample(mates[[i]], clutchSize[[i]], replace = TRUE))

    # now set-up table of recruits assembling all variables
    recruits <- data.frame(Me=uuid(sum(clutchSize), drop_hyphens = TRUE), # unique ID
                           Sex=sample(c("M","F"), sum(clutchSize), prob = c(0.5,0.5), replace = TRUE),
                           Mum=rep(iBreed_F$Me, clutchSize), # add mum vector that match litter sizes a
                           Dad=unlist(dads), # unlist dads (length should match new recruits number)
                           BirthY=year, # rest of meta-data is simple for recruits. birth year
                           DeathY=NA, # not dead yet
                           Stock=1, # single stock for now
                           AgeLast=0, # recruits start at age 0
                           SampY=NA) # not sampled yet

    outs <- rbind(indiv, recruits)
    return(outs)
}
```

With myMate defined, we can go on to write a simulation using the other fishSim tools, as in this stripped-down example:

```
indiv <- makeFounders(stocks = c(1))
ageMort <- c(0.305, rep(0.2, 100)) ## age-specific mortality as in the
## first, 'simple' example
for (y in 1:10) {
    indiv <- myMate(indiv = indiv, batchSize = 3, breedProb = 1,
                    maturityAge = 4, year = y, meanMates = 3)
    indiv <- mort(indiv, year = y, type = "age", ageMort = ageMort)
    indiv <- birthdays(indiv)
    if(y %in% c(6:10)) {
        indiv <- capture(indiv, n = 40, fatal = FALSE, year = y)
    }
}
pairs <- findRelativesPar(indiv = indiv, sampled = TRUE, nCores = 2)
#>   Sample.Year Freq
#> 1           6   39
#> 2           7   38
#> 3           8   40
#> 4           9   40
#> 5          10   40
#> [1] "parents found at 2024-05-16 13:14:50.608424"
#> [1] "grandparents found at  2024-05-16 13:14:50.671169"
#> [1] "great-grandparents found at 2024-05-16 13:14:50.763042"
#> [1] "great-great-grandparents found at 2024-05-16 13:14:50.873166"
#> [1] "great-great-great-grandparents found at 2024-05-16 13:14:50.987104"
#> [1] "great-great-great-great-grandparents found at 2024-05-16 13:14:51.12034"
#> 1000 of 19306 comparisons2000 of 19306 comparisons3000 of 19306 comparisons4000 of 19306 comparisons
namedRelatives(pairs)
#>   POPs GGPs dGGPs G4Ps dG4Ps G6Ps dG6Ps HSPs FSPs HTPs FTPs HCPs FCPs GHCPs GFCPs  ORCs
#> 1    17     5       0     2       0     0       0    53    69    94  1637     52  2718      11   2215  12433
```