KernelPop

Allan E. Strand

Department of Biology
College of Charleston
Charleston, SC 29424
stranda@cofc.edu

January 21, 2007

1 Introduction

`KernelPop`is a software environment for simulating population genetics in a explicitly spatial manner. It is a discrete-time and individual-based. This software provides flexibility at several levels of organization. From general to narrow these include: Landscapes, habitats, populations, individuals, and loci. `KernelPop`implements this structure for a single species using a single list-based data structure in R. The vast majority of functions either help create or modify this structure or to extract information from it.

Using the set of tools provided with KernelPop along with R, it is possible to define and execute almost any demographic scenario that can be implemented in discrete-time. The actual simulations are carried out in C++ to increase speed.

2 Landscape object

Figure 1 illustrates the high-level organization of a landscape object. The following subsections will document the different sub-components.

Each will:

- describe the sub-object
- describe the function(s) used to create it

The first step in creating a landscape object is to create a skeleton landscape

```
> land <- landscape.new.empty()
> names(land)
[1] "intparam"    "switchparam" "floatparam"  "demography"  "loci"
[6] "expression"  "individuals"
```

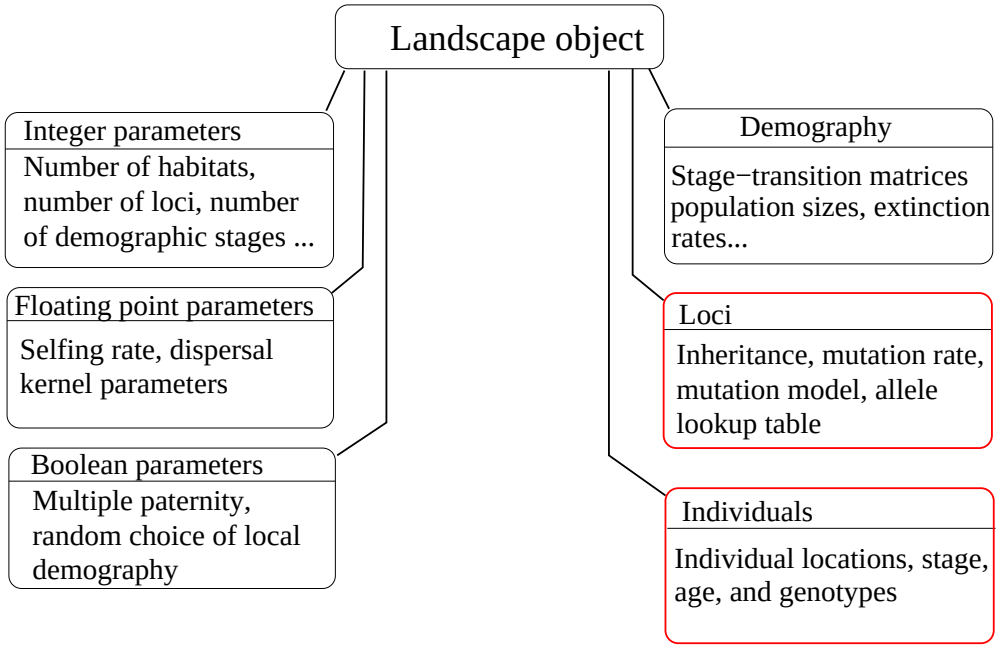


Figure 1: Schematic of the high-level organization of the landscape object

2.1 intparam

The `intparam` element of the landscape object describes integer values, they include:

habitats (h) This parameter provides the number of rectangular habitats within a landscape. These habitats may or may not be populated.

stages (s) The number of demographic stages present in a single habitat.

locusnum The number of genetic loci to be simulated (not changeable directly)

numepochs It is possible to have multiple epochs during the course of a simulation. This can be implemented either in R by the investigator (recommended) or in C++ by specifying multiple epochs sub-objects. numepochs specifies the number of these to be used in C++. (not changeable directly)

currentgen (cg) Current generation (year) of a simulation (best to leave unchanged)

currentepoch (ce) Current epoch (leave unchanged)

totalgens (totgen) Total number of generations that can be simulated

numdemos The number of different within-habitat demographies. These can vary across a landscape. (not changeable directly)

maxlandsize (maxland) Total number of individuals that can be simulated

nphen (np) not used yet.

The `args()` provides the default values for this function[1].

```
> args(landscape.new.intparam)

function (rland, h = 1, s = 1, cg = 0, ce = 0, totgen = 1000,
    maxland = 2e+05, np = 0)
NULL

> land <- landscape.new.intparam(land, h = 4, s = 6)
```

Note that this function takes a landscape object (in this case, skeleton) as one of its parameters. It returns the modified landscape. This is typical for the landscape creation and modification functions

2.2 floatparams

These are parameters describe selfing rate (s) and then a set of dispersal characteristics that will apply to all stages in the simulation

selfing (s) Selfing rate in a mixed-mating model

seedmu Scale of first seed dispersal distribution

seedshape Shape of first seed dispersal distribution

pollenmu Scale of first pollen dispersal distribution

pollenshape Shape of first pollen dispersal distribution

seedmu2 Scale of second seed dispersal distribution

seedshape2 Shape of second seed dispersal distribution

seedmix mixture parameter between the two seed distributions (1=all dist 1, 0=all dist 2)

aspect Factor that reduces the y-dimension of dispersal. A value of 1 is equal x and y dispersal. This does not change the dispersal distances, however, just their direction

pollenmu2 Scale of second pollen dispersal distribution

pollenshape2 Shape of second pollen dispersal distribution

[1] All code in this document should be available in annotated form in the file kernelPop-intro.R distributed with this pdf. If you source it into R it will execute the code and produce similar results. If you set '`par(ask=T)`' before '`sourcing`', the graphs (and subsequent steps!) will wait for you to hit return

pollenmix mixture parameter between the two pollen distributions (1=all dist 1, 0=all dist 2)

```
> args(landscape.new.floatparam)

function (rland, s = 0, seedscale = c(10, 10), seedshape = c(10,
    10), seedmix = 1, pollenscale = c(2, 10), pollenshape = c(2,
    6), pollenmix = 1, asp = 1, mindens = 1e-25)
NULL

> land <- landscape.new.floatparam(land)
```

2.3 switchparam

These are boolean parameters that make choices about a landscape

randepoch If 0, choose different epochs in the order they are specified. If 1 choose epochs at random (again multiple epochs are kind of obsolete, but are built into the basic C++ engine) (don't change)

randdemo If there are multiple local demographies for different habitats, choose among them at random for each habitat if 1; if 0, assign them in the same order as the habitats are defined.

multp If 1 each zygote potentially has a different father (slower). If 0 all offspring for a mother in a particular year are full-sibs

```
> args(landscape.new.switchparam)

function (rland, re = 0, rd = 0, mp = 1)
NULL

> land <- landscape.new.switchparam(land)
```

2.4 demography

This sub-object basically defines all of the vital rates that determine survival and reproduction. It also contains functions that define metapopulation characteristics like extinction rate per habitat, carrying capacity per habitat and dispersal parameters.

It is divided into two subcomponents, each of which is further divided.

2.4.1 localdem

This is a description of the sub-matrices that define demography in each habitat. It is a list of any length between 1 and the number of habitats. Depending on the value of `randdemo` in the switch subcomponent, each habitat is either: 1) randomly assigned an element from this list with probabilities that can be defined in the epochs sub object (sec. 2.4.2) or 2) assigned demographies by cycling through this list.

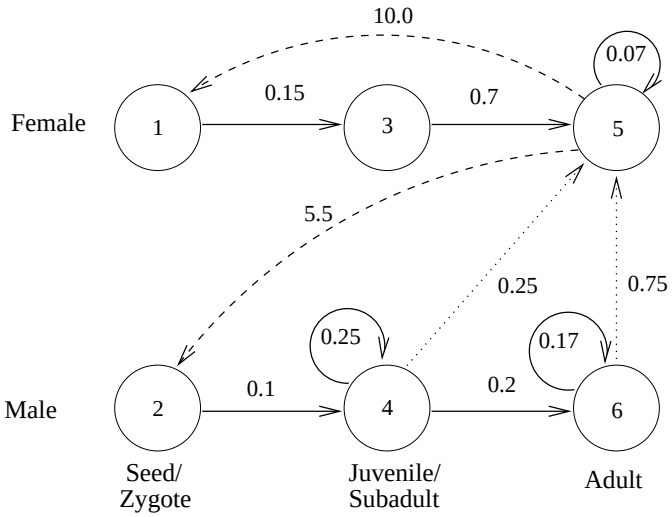
Each element is composed of three matrices that describe a stage based demography. Figure

```
> S <- matrix(c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.18, 0, 0,
+     0, 0, 0, 0, 0.14, 0, 0.26, 0, 0, 0, 0, 0.7, 0, 0.09, 0, 0,
+     0, 0, 0.2, 0, 0.18), byrow = T, nrow = 6)
> R <- matrix(c(0, 0, 0, 0, 14, 0, 0, 0, 0, 0, 8.5, 0, 0, 0, 0,
+     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
+     0, 0), byrow = T, nrow = 6)
> M <- matrix(c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
+     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.25, 0, 0.75, 0, 0, 0,
+     0, 0, 0), byrow = T, nrow = 6)
> args(landscape.new.local.demo)

function (rland, S, R, M)
NULL

> land <- landscape.new.local.demo(land, S, R, M)
> print("add a new local demography with different Reproduction")

[1] "add a new local demography with different Reproduction"

> R2 <- matrix(c(0, 0, 0, 0, 8, 0, 0, 0, 0, 0, 5.5, 0, 0, 0, 0,
+     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
+     0, 0), byrow = T, nrow = 6)
> land <- landscape.new.local.demo(land, S, R2, M)
```

The previous code block installed two local demographies into the localdem list.

2.4.2 epochs

This is a list of elements each of which define landscape-level characteristics. Currently my recommendation is to declare only 1 and modify it in R to change simulations through time.



R

To \ From	1	2	3	4	5	6
1					10	
2					5.5	
3						
4						
5						
6						

S

To \ From	1	2	3	4	5	6
1						
2						
3	0.15					
4		0.1		0.25		
5			0.7		0.07	
6				0.2		0.17

M

To \ From	1	2	3	4	5	6
1						
2						
3						
4						
5				0.25		0.75
6						

Figure 2: Stage-based demography within habitats. This life-cycle has six stages, essentially three male and three female stages. The rates of male and female survival are different (for example, intermediate sized females (stage 3) have high probabilities of growing into adults in the next season). Females produce a mean of 10 and 5.5 male and female juveniles per generation, respectively. These are means of Poisson distributions. Both stage 4 and stage 5 can produce pollen though stage 6 produces three times the pollen on average.

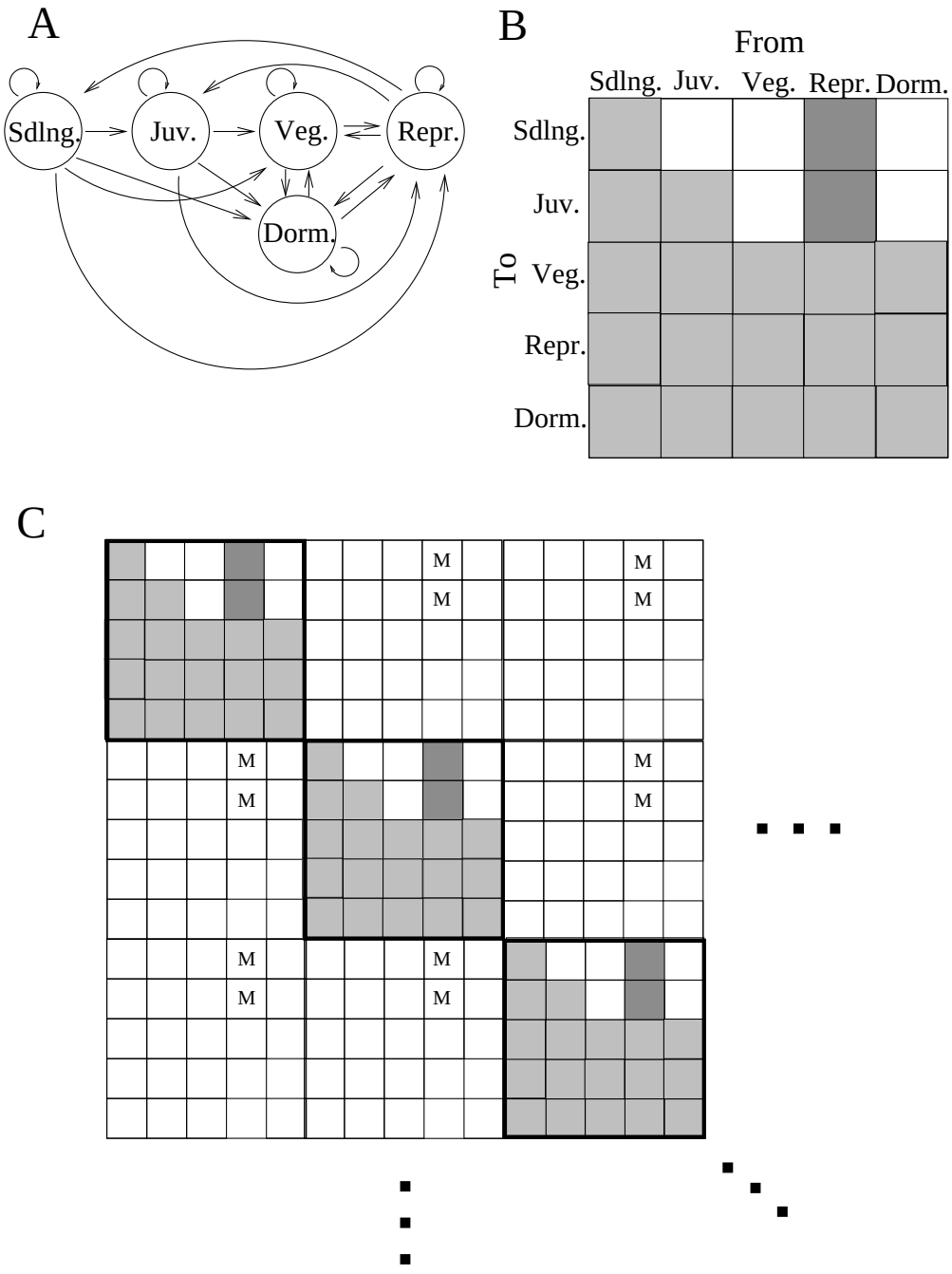


Figure 3: Schematic of landscape matrix metaphor

Landscape demographic characteristics The matrices defined in section 2.4.1 comprise sub-matrices of 3 larger matrices that are (number of habitats)*(number of stages) in dimension This is similar to a multi regional model. Figure may help illustrate this concept, though it has the reproduction and survival matrices summed into a single matrix.

These matrices can implement dispersal among populations independent of the dispersal kernels defined in the next paragraphs. I wanted to maintain this ability because it provides a convenientt way to model management efforts like transplants, introductions, and stocking efforts.

For most uses of `KernelPop`, these matrices can be set to zero in each cell. They still need to be defined, however. The matrix `zeromat` will be used for each of the three landscape matrices

```
> zeromat <- matrix(0, nrow = 4 * 6, ncol = 4 * 6)
```

Vectors describing habitats within a landscape Vectors with length equal to the number of habitats are defined in this sub-element to characterize:

extinction the yearly rate of extinction of each habitat

carrying capacity the largest number of individuals supported in each habitat

```
> extnct <- c(0, 0.1, 0, 0.1)
> k <- c(1000, 600, 600, 1000)
```

Vectors with length equal to the length of the local demography describe the probability of picking a local demography from the localdem list. Always defined, but only used if `randdemo` (section 2.3) is set appropriately.

```
> ldem <- c(0.5, 0.5)
```

Seed dispersal kernel matrix This is a matrix that has (number of habitats)*(number of stages) rows and six columns. If not specified, a working matrix is constructed from the floatparam elements

The rows correspond to stages in the landscape. The columns correspond to (in order)

1. the seed dispersal kernel. This can be 1,2, or 3 (the default). 1 and 2 are really just special cases of the mixed pdf (kernel 3)
2. the scale parameter for kernel component 1
3. the shape parameter for kernel component 1
4. the scale parameter for kernel component 2
5. the shape parameter for kernel component 2

6. the mixing parameter. Ranges from 0 to 1. If 1 dispersal is determined solely by kernel 1. If 0, dispersal is determined solely by kernel 2. Intermediate values represent a mixture.

```
> sk <- matrix(0, nrow = 4 * 6, ncol = 6)
> sk[, 1] <- rep(3, 4 * 6)
> sk[, 2] <- rep(10, 4 * 6)
> sk[, 3] <- rep(1.1, 4 * 6)
> sk[, 4] <- rep(100, 4 * 6)
> sk[, 5] <- rep(50, 4 * 6)
> sk[, 6] <- rep(0.5, 4 * 6)
> sk
```

```
      [,1] [,2] [,3] [,4] [,5] [,6]
 [1,]    3   10  1.1  100   50  0.5
 [2,]    3   10  1.1  100   50  0.5
 [3,]    3   10  1.1  100   50  0.5
 [4,]    3   10  1.1  100   50  0.5
 [5,]    3   10  1.1  100   50  0.5
 [6,]    3   10  1.1  100   50  0.5
 [7,]    3   10  1.1  100   50  0.5
 [8,]    3   10  1.1  100   50  0.5
 [9,]    3   10  1.1  100   50  0.5
[10,]    3   10  1.1  100   50  0.5
[11,]    3   10  1.1  100   50  0.5
[12,]    3   10  1.1  100   50  0.5
[13,]    3   10  1.1  100   50  0.5
[14,]    3   10  1.1  100   50  0.5
[15,]    3   10  1.1  100   50  0.5
[16,]    3   10  1.1  100   50  0.5
[17,]    3   10  1.1  100   50  0.5
[18,]    3   10  1.1  100   50  0.5
[19,]    3   10  1.1  100   50  0.5
[20,]    3   10  1.1  100   50  0.5
[21,]    3   10  1.1  100   50  0.5
[22,]    3   10  1.1  100   50  0.5
[23,]    3   10  1.1  100   50  0.5
[24,]    3   10  1.1  100   50  0.5
```

Pollen dispersal kernel matrix This is a matrix that has (number of habitats)*(number of stages) rows and six columns. If not specified, a working matrix is constructed from the floatparam elements

The rows correspond to stages in the landscape. The columns correspond to (in order)

1. the pollen dispersal kernel. This can be 1,2, or 3 (the default). 1 and 2 are really just special cases of the mixed pdf (kernel 3)
2. the scale parameter for kernel component 1
3. the shape parameter for kernel component 1
4. the scale parameter for kernel component 2
5. the shape parameter for kernel component 2
6. the mixing parameter. Ranges from 0 to 1. If 1 dispersal is determined solely by kernel 1. If 0, dispersal is determined solely by kernel 2. Intermediate values represent a mixture.

```
> pk <- matrix(0, nrow = 4 * 6, ncol = 6)
> pk[, 1] <- rep(3, 4 * 6)
> pk[, 2] <- rep(5, 4 * 6)
> pk[, 3] <- rep(2, 4 * 6)
> pk[, 4] <- rep(100, 4 * 6)
> pk[, 5] <- rep(50, 4 * 6)
> pk[, 6] <- rep(1, 4 * 6)
> pk
```

```
      [,1] [,2] [,3] [,4] [,5] [,6]
 [1,]    3    5    2  100   50    1
 [2,]    3    5    2  100   50    1
 [3,]    3    5    2  100   50    1
 [4,]    3    5    2  100   50    1
 [5,]    3    5    2  100   50    1
 [6,]    3    5    2  100   50    1
 [7,]    3    5    2  100   50    1
 [8,]    3    5    2  100   50    1
 [9,]    3    5    2  100   50    1
[10,]    3    5    2  100   50    1
[11,]    3    5    2  100   50    1
[12,]    3    5    2  100   50    1
[13,]    3    5    2  100   50    1
[14,]    3    5    2  100   50    1
[15,]    3    5    2  100   50    1
[16,]    3    5    2  100   50    1
[17,]    3    5    2  100   50    1
[18,]    3    5    2  100   50    1
[19,]    3    5    2  100   50    1
[20,]    3    5    2  100   50    1
```

```
[21,]    3    5    2  100   50    1
[22,]    3    5    2  100   50    1
[23,]    3    5    2  100   50    1
[24,]    3    5    2  100   50    1
```

This pollen kernel specifies a Weibull with shape parameter 2 and scale parameter 5. The mixing parameter is set to 1 so the other distribution component values are actually unimportant.

Habitat locations The habitat locations are encoded as a set of 4 vectors giving the left and right x coordinates and bottom and top y coordinates for each habitat. Here the two habitats are defined as 0,600,0,600 and 800,1400,800,1400

```
> lx <- c(0, 0, 800, 800)
> rx <- c(600, 600, 1400, 1400)
> bty <- c(0, 800, 0, 800)
> ty <- c(600, 1400, 600, 1400)
```

Put it all in an epoch The different epoch elements are added to the landscape:

```
> args(landscape.new.epoch)

function (rland, S = NULL, R = NULL, M = NULL, epochprob = 1,
    startgen = 0, extinct = NULL, carry = NULL, localprob = NULL,
    pollen.kernels = NULL, seed.kernels = NULL, leftx = NULL,
    rightx = NULL, boty = NULL, topy = NULL, maxland = c(0, 0,
        10000, 10000))
NULL

> land <- landscape.new.epoch(land, S = zeromat, R = zeromat, M = zeromat,
+     extinct = extnct, carry = k, localprob = ldem, pollen.kernels = pk,
+     seed.kernels = sk, leftx = lx, rightx = rx, boty = bty, topy = ty)
```

2.5 Loci

This section describes the genetic loci These loci are unlinked, though those with maternal inheritance are effectively linked as there is no segregation among them.

This element is a list of loci. Each locus has some characteristics and then a list of all the alleles at the locus

2.5.1 Locus characteristics

The locus characteristics include

type The type of locus, infinite allele, stepwise mutation, or sequence.

ploidy haploid or diploid

trans transmission mode (biparental versus maternal)

rate locus-wide per meiosis mutation rate

Alleles Each locus has a list of alleles with the following elements. The elements aren't typically modified directly, just in the definition of a locus and the course of a simulation

aindex the allele index, this is used to create lookup tables between individuals genotypes and the allele states

birth year of allele arising from mutation

prop the proportion of the allele at that locus across the entire landscape (never really used though)

state the actual allele state (microsatellite repeat number, infinite allele designation, actual sequence)

Each locus is added in turn by a call to `lanscape.new.locus` Here I add three loci:

1. haploid maternally inherited infinite allele model (5 alleles)
2. diploid biparentally inherited stepwise mutation model (3 alleles)
3. diploid biparentally inherited finite sequence model (3 alleles, 125 bases)

Each will be initialized with 3 alleles. Sequences are generated at random with base frequencies equal to 0.25 per residue.

```
> args(landscape.new.locus)
```

```
function (rland, type = 0, ploidy = 1, mutationrate = 0, transmission = 1,
    numalleles = 2, allelesize = 50, frequencies = NULL)
NULL
```

```
> land <- landscape.new.locus(land, type = 0, ploidy = 1, transmission = 1,
+     numalleles = 5)
> land <- landscape.new.locus(land, type = 1, ploidy = 2, transmission = 0,
+     numalleles = 3)
> land <- landscape.new.locus(land, type = 2, ploidy = 2, transmission = 0,
+     numalleles = 3, allelesize = 125)
> length(land$loci)
```

```
[1] 3
```

2.6 Individuals

Along with the loci defined in the section below (both colored red in figure 1) the individuals section changes through the course of a simulation. Each landscape object describes a landscape state at a point in time. The individuals that are alive are represented in this section by a matrix. This matrix has as many rows as individuals in the landscape. The number of columns includes (currently 9) demographic columns followed by genetics columns that are determined by the locus object. This works out as 1 column for every haploid locus and 2 columns for the diploid loci. `landscape.ploidy(land)` returns the ploidy for each locus in order they were appended to the landscape. Because the number of demographic columns could change the function `landscape.democol()` returns the highest number of the deomgraphic columns (currently 9)

```
> landscape.ploidy(land)

[1] 1 2 2

> landscape.democol()

[1] 9
```

The function `landscape.new.individuals(land)` automatically populates a landscape with no individuals. It allows you to specify the population sizes in each demographic stage in the landscape. Therefore the vector length is (number habitats)*(number of local stages) in length. One thing to keep in mind is that if you used the standard function `landscape.simulate(land,x)` to simulate, it will first apply the rules encoded in the S matrices before the R matrices. This means if you should probably define at least some offspring/juvenile individuals initially to mature into reproductive individuals.

Also be careful, it is easy to define huge numbers of reproductive individuals that produce huge numbers of offspring in the next generation. This can overwhelm some computers depending on RAM and chip speed. Simulating a total of 10,000-20,000 individuals are tolerable, but not for large numbers of reps on a mac g4 laptop running at 1.4Ghz with 1Gb RAM. Reproduction is the slow step, so life-cycles that have high over survival and low reproduction run faster than low survival- high reproduction life-cycles.

```
> vlen <- land$intparam$habitats * land$intparam$stages
> vec <- rep(100, vlen)
> land <- landscape.new.individuals(land, vec)
```

3 Operations on landscapes

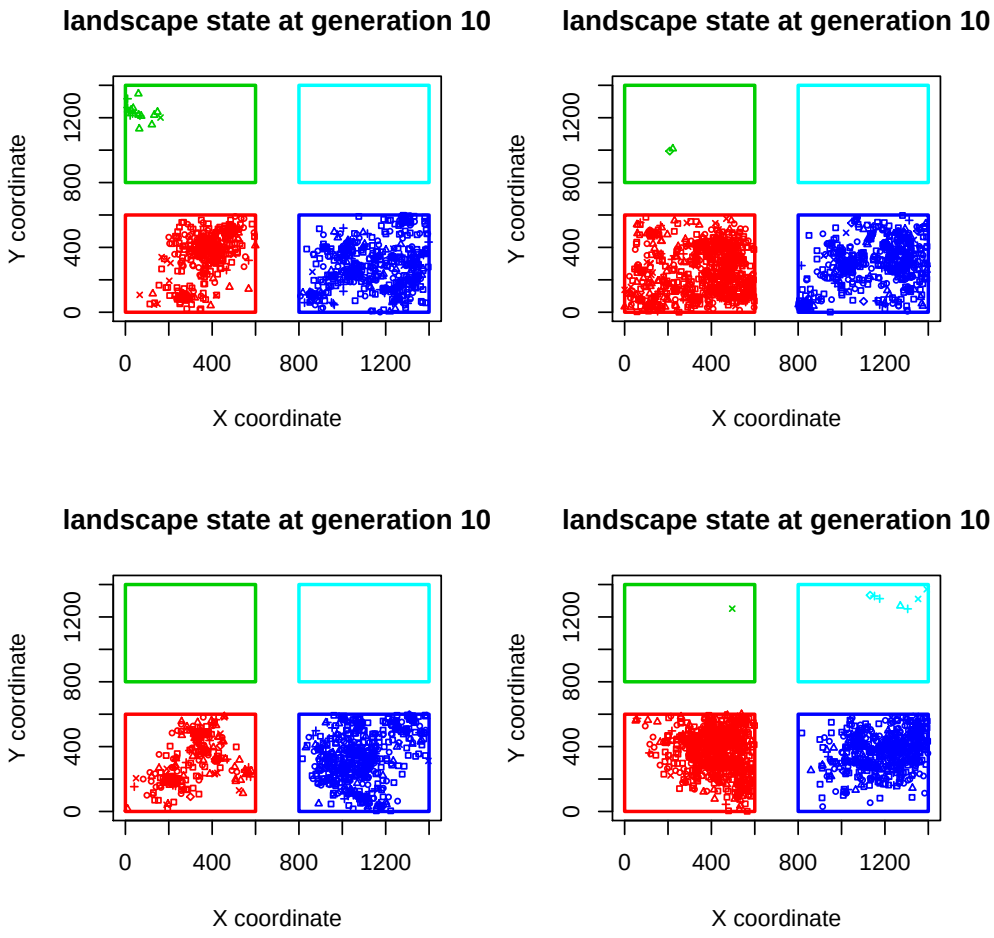
3.1 Simulation

To simulate ecology and genetics use `landscape.simulate()`

```
> l1 <- landscape.simulate(land, 10)
> l2 <- landscape.simulate(land, 10)
> l3 <- landscape.simulate(land, 10)
> l4 <- landscape.simulate(land, 10)
```

These commands created four replicate simulations of 10 years each, starting with the same initial conditions defined above.

```
> par(mfrow = c(2, 2))
> landscape.plot.locations(l1)
> landscape.plot.locations(l2)
> landscape.plot.locations(l3)
> landscape.plot.locations(l4)
> par(mfrow = c(1, 1))
```



3.2 I/O

landscapes can be written and read from disk as R binary files using `save` and `load`. This allows the state of a simulation to be saved at any point, and just as importantly, a landscape can be read into a fresh install of kernelPop and the simulation should proceed with the same parameter values as the previous generations (probably a different random seed, though).

```
> save(file = "l1.rda", l1)
> rm(l1)
> load(file = "l1.rda", l1)
```

3.3 Altering landscapes through time

Because a landscape is a complete state, it is possible to change them through the course of the simulation. This rasises the possiblity that landscapes can be altered during a simulation run. At the moment this is going to require altering the landscape object directly. This is no big deal, but it does take a pretty good understanding of the landscape structure. Some things are not a good idea to change. Most of the `intparam`s should remain untouched. The `floatparams` can change (though if you are doing this, most of the changes will probably be at the level of the pollen kernels and seed kernels). The `switch` parameters can be changed, as can the demographic rates in `localdem` and `epoch`.

Do not change the `loci` object directly. That should really happen using the simulation routines.

The `individuals` object can be changed in some ways. The easiest is to kill individuals at random. It is also fairly easy to add selection on a particular allele at a locus, though doing this every generation starts to use up some CPU cycles converting R objects to C++ objects.

This will increase long distance dispersal in this entire landscape at generation 10.

```
> names(l1$demography$epochs[[1]])

 [1] "RndChooseProb" "StartGen"      "Extinct"       "Carry"
 [5] "Localprob"     "S"             "R"             "M"
 [9] "leftx"         "rightx"        "topy"          "boty"
[13] "pollenkern"    "seedkern"

> sk <- matrix(0, nrow = 4 * 6, ncol = 6)
> sk[, 1] <- rep(3, 4 * 6)
> sk[, 2] <- rep(10, 4 * 6)
> sk[, 3] <- rep(1.1, 4 * 6)
> sk[, 4] <- rep(400, 4 * 6)
> sk[, 5] <- rep(100, 4 * 6)
> sk[, 6] <- rep(0.5, 4 * 6)
> sk
```

```
      [,1] [,2] [,3] [,4] [,5] [,6]
 [1,]    3   10  1.1  400  100  0.5
 [2,]    3   10  1.1  400  100  0.5
 [3,]    3   10  1.1  400  100  0.5
 [4,]    3   10  1.1  400  100  0.5
 [5,]    3   10  1.1  400  100  0.5
 [6,]    3   10  1.1  400  100  0.5
 [7,]    3   10  1.1  400  100  0.5
 [8,]    3   10  1.1  400  100  0.5
 [9,]    3   10  1.1  400  100  0.5
[10,]    3   10  1.1  400  100  0.5
[11,]    3   10  1.1  400  100  0.5
[12,]    3   10  1.1  400  100  0.5
[13,]    3   10  1.1  400  100  0.5
[14,]    3   10  1.1  400  100  0.5
[15,]    3   10  1.1  400  100  0.5
[16,]    3   10  1.1  400  100  0.5
[17,]    3   10  1.1  400  100  0.5
[18,]    3   10  1.1  400  100  0.5
[19,]    3   10  1.1  400  100  0.5
[20,]    3   10  1.1  400  100  0.5
[21,]    3   10  1.1  400  100  0.5
[22,]    3   10  1.1  400  100  0.5
[23,]    3   10  1.1  400  100  0.5
[24,]    3   10  1.1  400  100  0.5
```

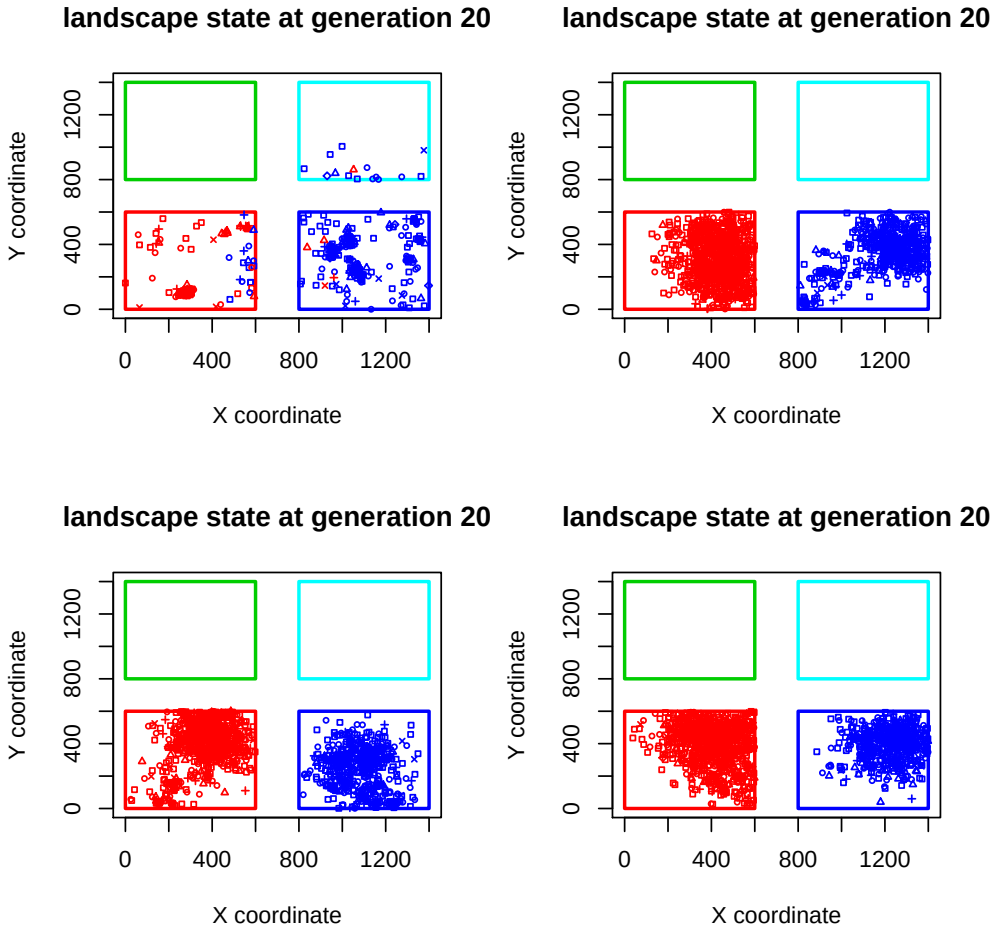
```
> l1$demography$epochs[[1]]$seedkern <- sk
```

Now we can simulate another 10 years, then plot it again. Note that the same object is used as parameter and result for the current state before and after simulation.

```
> l1 <- landscape.simulate(l1, 10)
> l2 <- landscape.simulate(l2, 10)
> l3 <- landscape.simulate(l3, 10)
> l4 <- landscape.simulate(l4, 10)
```

In this plot the changed landscape is on the top left panel.

```
> par(mfrow = c(2, 2))
> landscape.plot.locations(l1)
> landscape.plot.locations(l2)
> landscape.plot.locations(l3)
> landscape.plot.locations(l4)
> par(mfrow = c(1, 1))
```



You can see usually see more long-distance seed dispersal events in the top left.

4 Extracting information from landscapes

4.1 Distances

The coordinates of every individual *and* their parents makes it straightforward to exmine the actual dispersal distributions for zygotes and male gametes.

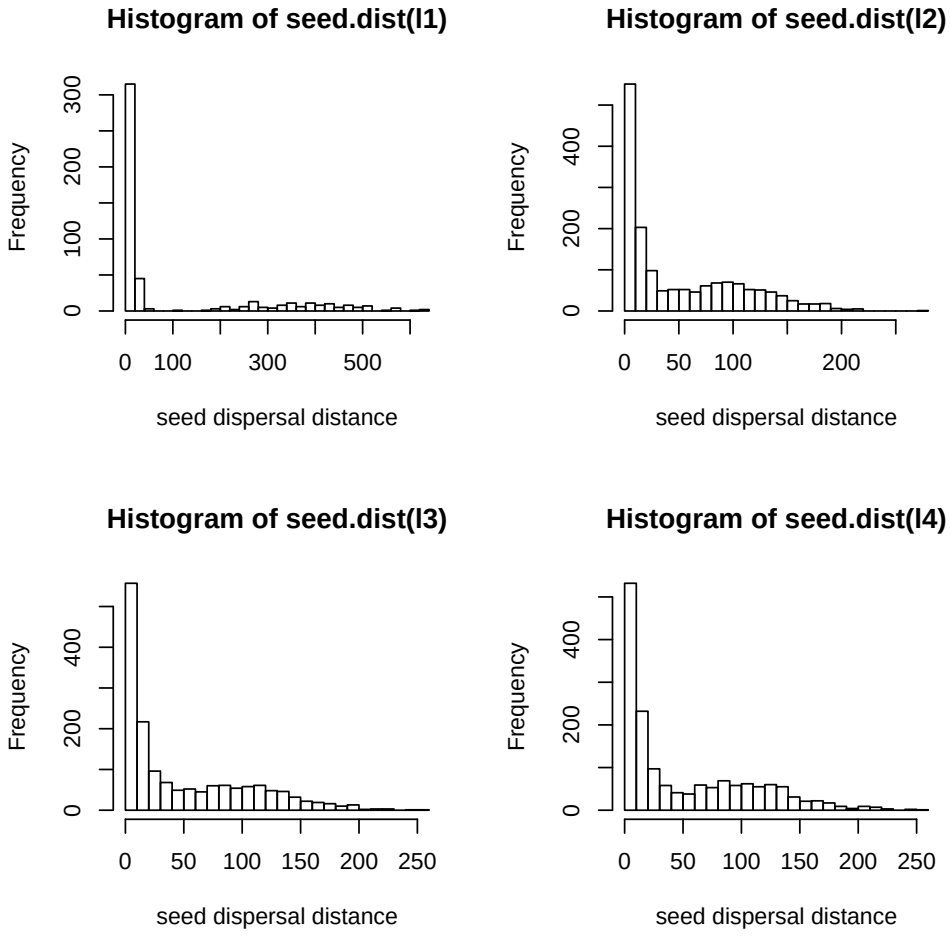
Here are the actual dispersal distances in generation 20.

The take advantage of really simple functions distributed with R

This is the seed-kernel. This includes every dispersal event that gave rise to an individual in the landscape used for the parameter.

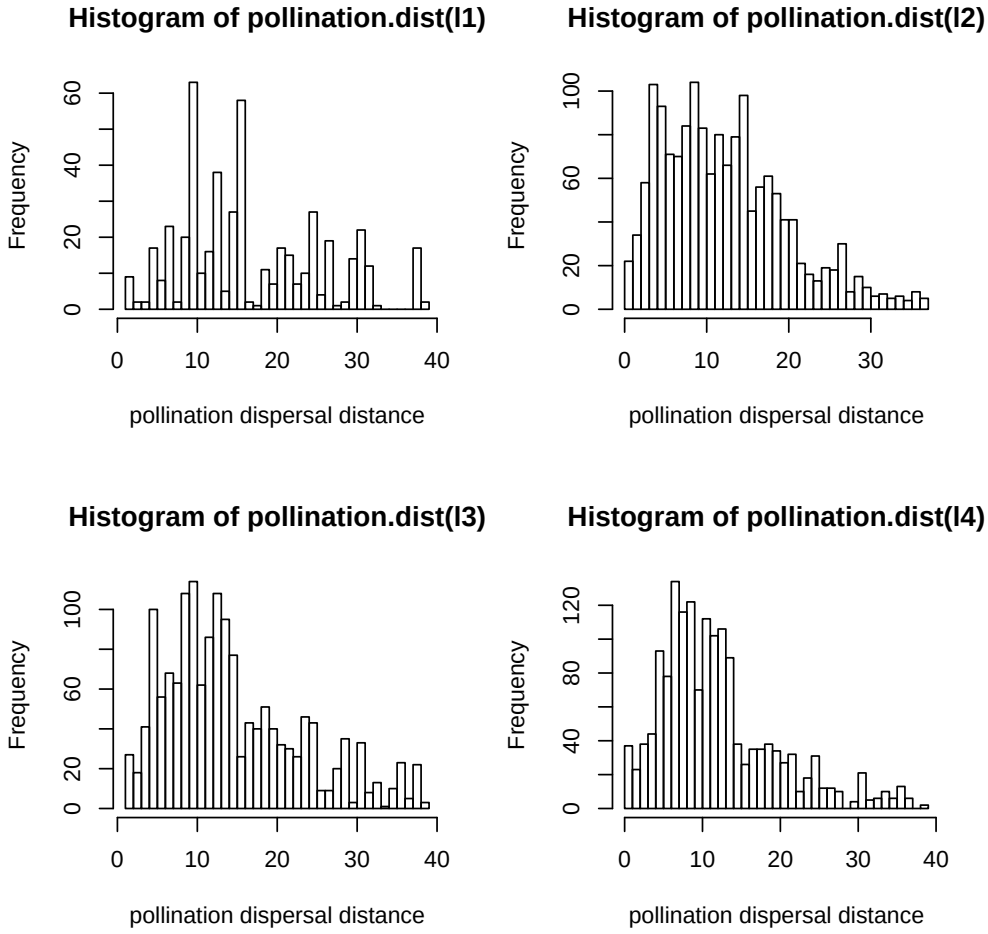
```
> source("../test/distance-functions.R")
> par(mfrow = c(2, 2))
> hist(seed.dist(l1), breaks = 30, xlab = "seed dispersal distance")
```

```
> hist(seed.dist(l2), breaks = 30, xlab = "seed dispersal distance")
> hist(seed.dist(l3), breaks = 30, xlab = "seed dispersal distance")
> hist(seed.dist(l4), breaks = 30, xlab = "seed dispersal distance")
> par(mfrow = c(1, 1))
```



Here is the pollination kernel. Pollination distance distributions are really dependent on the spatial structure of plants.

```
> par(mfrow = c(2, 2))
> hist(pollination.dist(l1), breaks = 30, xlab = "pollination dispersal distance")
> hist(pollination.dist(l2), breaks = 30, xlab = "pollination dispersal distance")
> hist(pollination.dist(l3), breaks = 30, xlab = "pollination dispersal distance")
> hist(pollination.dist(l4), breaks = 30, xlab = "pollination dispersal distance")
> par(mfrow = c(1, 1))
```



4.2 Populations

The function `landscape.populations(land)` returns a vector of the population assignments of each individual in the landscape. This can be useful in selecting individuals from specific populations for further processing as well as measuring population sizes. For example the population sizes in the landscape l1 just simulated are:

```
> table(landscape.populations(l1))

  1   3   4
155 321  15
```

4.3 Genetic information

So far the construction of landscapes and their simulation have been described. Genetic information can be extracted from the landscape as well. The allele indices and states for

each genotype at each locus can be accessed by `landscape.locus` and `landscape.states`, respectively. There are also some summary statistics. These include F_{ST}, Φ_{ST}, allele frequencies, and measures of $\theta = 4N_e\mu$.

An important function is `landscape.sample` this simulates random sampling of populations to subsequently analyse. It speeds up analyses and allows you to examing the impact of sampling on summary statistics.

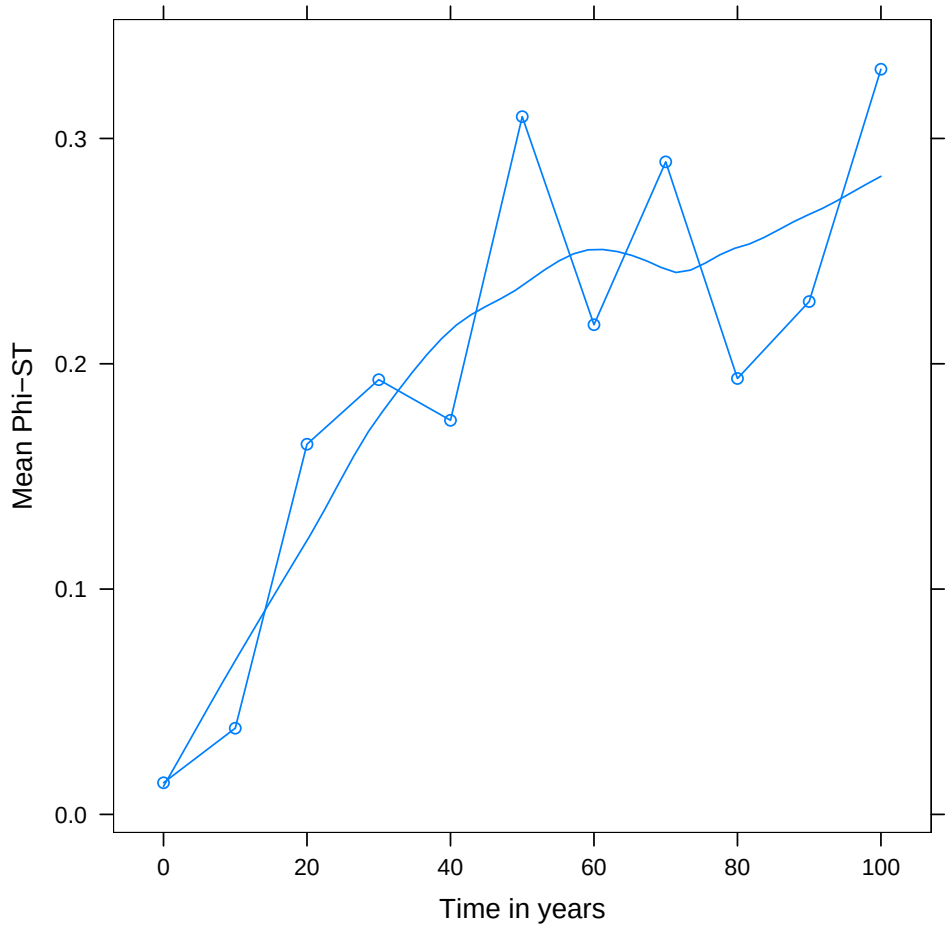
Here is some code to simulate the landscape `land` defined above for 100 generations, saving the state at 10 generation intervals in a list called `landlist` At generation 50, the mean dispersal distance is increased (though the mixture parameter is weighted more highly towards local dispersal)

```
> gland <- land
> landlist <- vector("list", 11)
> landlist[[1]] <- gland
> for (i in 2:11) {
+     print(table(landscape.populations(gland)))
+     gland <- landscape.simulate(gland, 10)
+     landlist[[i]] <- gland
+     if (i == 6) {
+         sk <- gland$demography$epochs[[1]]$seedkern
+         sk[, 4] <- rep(500, dim(sk)[1])
+         sk[, 6] <- rep(0.8, dim(sk)[1])
+         gland$demography$epochs[[1]]$seedkern <- sk
+     }
+ }

  1   2   3   4
600 600 600 600

  1   3
189 596

  1   3
365 598

  1   3
503 598

  1   3
997 597

  1   3
997 598
```

```
  1   2   3   4
997  77 597  76

  1   2   3   4
997  78 598 183

  1   2   3   4
997  69 596 147

  1   2   3   4
997 124 597 161
```

This code chunk takes the list created above, "collects" 24 individuals from each of the extant populations (`landscape.amova` calls `landscape.sample` internally) at each time point and calculates mean Φ_{ST}. It creates an object with two columns: the generation of the simulation and the mean Φ_{ST}. These are then plotted.

```
> plot.ob <- do.call(rbind, lapply(landlist, function(l) {
+     c(l$intparam$currentgen, mean(landscape.amova(l, ns = 24)))
+ }))
> print(xyplot(plot.ob[, 2] ~ plot.ob[, 1], type = c("b", "smooth"),
+     xlab = "Time in years", ylab = "Mean Phi-ST"))
```



4.4 Writing files out for other programs to use

The simple analyses in R may fall short of those implemented in other languages There several functions to write files in formats that support other software. Right now, these functions only write out genotypic data suitable for frequency-based analyses. There is room for improvement for outputting sequences and microsat states.

GenePop There is a function that outputs data into GenePop format which can be used by a host of other software. This is called `landscape.genepop.output`. It only exports the diploid loci. Right now, `landscape.genepop.output` does output allele states, including microsat states, but it does not produce sequences for the sequence locus type. Instead it produces the allele indices as alleles in the output file. Because allele indices can equal zero, and this is the missing data designation in genepop, 1 is added to all allele indices in the output file

```
> landscape.genepop.output(gland)
```

Arlequin `landscape.write.foreign` can output diploid Arlequin files of genotypes. The allele states are not used.

```
> l <- landscape.write.foreign(gland, fn = "diploid-arlequin.arb",
+     fmt = "arlequin")
```

migrate `landscape.write.foreign` can also output files in a format that `migrate` should be able to read. This function outputs diploid data in the form of genotypes for the genotypic version of migrate analyses. No sequences are output yet.

```
> l <- landscape.write.foreign(gland, fn = "migrate.infile", fmt = "migrate")
```

Biosys `landsacpe.write.foreign` should also be able to output files in biosys format

```
> l <- landscape.write.foreign(gland, fn = "biosys.txt", fmt = "biosys")
```

fdist `landscape.write.fdist` produces an infile suitable for use in the program FDIST. This is based solely on allele frequencies

```
> landscape.write.fdist(gland)
```

The code snippets above should leave files in the 'inst/doc' directory of your `KernelPop`installation.