Writing forward-time simulations, an in-depth course

6th June 2007

outline

simuPOP modules

```
>>> from simuOpt import setOptions
>>> setOptions(alleleType='long', optimized=False)
>>> from simuPOP import *
simuPOP : Copyright (c) 2004-2006 Bo Peng
Version 9.9.9 (Revision 9999, May 21 2007) for Python 2.3.4
[GCC 3.4.6 20060404 (Red Hat 3.4.6-3)]
Random Number Generator is set to mt19937 with random seed 0x70103f076358dc0
This is the standard long allele version with 65536 maximum allelic states.
For more information, please visit http://simupop.sourceforge.net,
or email simupop-list@lists.sourceforge.net (subscription required).
>>>
```

- Allele type: short, long, binary
- Standard and Optimized
- MPI (parallel) version, not ready

Common properties of all individuals

All individuals have the same genotypic structure, which refers to

- Ploidy (diploid, haploid, triploid, ...)
- Number of chromosomes
- Number of loci on each chromosome
- Name and position of loci
- Name of information fields

And less importantly

- Allele names
- Existence of sex chromosome

Chromosome structure

```
>>> pop = population(size=10, loci=[2, 4, 5])
>>> print pop.numLoci()
(2, 4, 5)
>>> # index starts at zero!
>>> print pop.numLoci(1)
4
>>> print pop.ploidy()
2
>>> print pop.ploidyName()
diploid
>>> print pop.chromBegin(1)
2
>>> print pop.locusPos(3)
2.0
>>> print pop.locusName(4)
loc2-3
>>>
```

Loci position and names

```
>>> pop = population(size=10, loci=[2, 4], maxAllele=3,
...     lociPos=[[1.5, 2.5], [1, 2, 5, 10]],
...     lociNames=['loc%x' % x for x in range(6)],
...     alleleNames=['A', 'T', 'C', 'G'])
>>> print pop.locusPos(3)
2.0
>>> print pop.locusName(4)
loc4
>>> print pop.alleleName(1)
T
>>>
```

Outline

Have a look at the population (Dump)

```
>>> Dump(pop)
Ploidy:                  2
Number of chrom:         2
Number of loci:          2 4
Maximum allele state:    3
Loci positions:
                1.5 2.5
                1 2 5 10
Loci names:
                loc0 loc1
                loc2 loc3 loc4 loc5
population size:         10
Number of subPop:        1
Subpop sizes:            10
Number of ancestral populations:         0
individual info:
sub population 0:
   0: MU AA AAAA | AA AAAA
   1: MU AA AAAA | AA AAAA
   2: MU AA AAAA | AA AAAA
   3: MU AA AAAA | AA AAAA
   4: MU AA AAAA | AA AAAA
   5: MU AA AAAA | AA AAAA
   6: MU AA AAAA | AA AAAA
   7: MU AA AAAA | AA AAAA
   8: MU AA AAAA | AA AAAA
   9: MU AA AAAA | AA AAAA
End of individual info
```

Have a look at the population (Dump)

```
>>> InitByFreq(pop, alleleFreq=[0.3, 0.7])
>>> Dump(pop)
Ploidy:                 2
Number of chrom:        2
Number of loci:         2 4
Maximum allele state:   3
Loci positions:
                1.5 2.5
                1 2 5 10
Loci names:
                loc0 loc1
                loc2 loc3 loc4 loc5
population size:        10
Number of subPop:       1
Subpop sizes:           10
Number of ancestral populations:        0
individual info:
sub population 0:
   0: MU TT ATTT | TT AATT
   1: FU TT TTTT | TA TATT
   2: FU TT TTAA | AT AAAT
   3: FU TT TTTA | TT TTTA
   4: MU TA TTTT | TT AATT
   5: MU TT AAAT | TA TTAT
   6: MU TT TTTT | TA TATT
   7: FU TA TATT | TT ATAT
   8: FU TT TTTT | TT TTAT
   9: FU TA TTTT | TT TTTT
```

Outline

Create a population with subpopulations

```
>>> pop = population(subPop=[2, 5, 6], loci=[2])
>>> print pop.popSize()
13
>>> print pop.subPopSizes()
(2, 5, 6)
>>> print pop.subPopSize(1)
5
>>> Dump(pop)
Ploidy:                  2
Number of chrom:         1
Number of loci:          2
Maximum allele state:    65535
Loci positions:
                1 2
Loci names:
                loc1-1 loc1-2
population size:         13
Number of subPop:        3
Subpop sizes:            2 5 6
Number of ancestral populations:        0
individual info:
sub population 0:
   0: MU    0  0 |    0  0
   1: MU    0  0 |    0  0
sub population 1:
   2: MU    0  0 |    0  0
   3: MU    0  0 |    0  0
   4: MU    0  0 |    0  0
```

Mating is within subpopulation only

```
>>> pop = population(subPop=[5, 6], loci=[2])
>>> simu = simulator(pop, randomMating())
>>> simu.evolve(
...     preOps = [
...         initByFreq(alleleFreq=[0.2, 0.8], subPop=[0]),
...         initByFreq([0, 0, 0, 0.5, 0.5], subPop=[1])
...         ],
...     ops = [
...         dumper(alleleOnly=True, indRange=[[0, 3], [5, 7]]),
...         recombinator(rate=0.1) ],
...     end = 1
... )
```

Mating is within subpopulation only – continue

```
individual info:
sub population 0:
   0: MU   1  1 |   1  1
   1: FU   1  1 |   1  1
   2: FU   1  1 |   1  0
sub population 1:
   5: MU   4  3 |   4  4
   6: MU   3  4 |   4  4
End of individual info.


No ancenstral population recorded.
individual info:
sub population 0:
   0: MU   1  1 |   1  1
   1: FU   1  0 |   1  1
   2: FU   1  1 |   1  1
sub population 1:
   5: MU   4  4 |   3  4
   6: MU   3  4 |   4  3
End of individual info.


No ancenstral population recorded.
True
```

Subpopulation manipulations

Outline

Information fields

Pieces of information that can be attached to each individual, e.g.

- `fitness:` fitness of each individual, calculated by selectors
- `father_idx, mother_idx`: index of parents in the parental generation
- `old_index:` index of an individual in the population where it is sampled

Or, self-defined

- birthday
- geographic location
- ...

Information fields – an example

Outline

The most flexible operators that can perform any operation, but are less efficient.

The idea: user provide a function with specified input and output, simuPOP calls this function during evolution.

Python operator

```
func(pop [, param])
```

Python Individual operator

`func(ind [, genotype] [, param]), return True/False or an array`

- `ind`: individual
- `genotype`: if parameter loci is given, genotype at these loci are passed to the function
- `param`: if parameter param is given, param passed from simuPOP
- `return`: if parameter infoFields is given, assign return values to these information fields

An example of `pyIndOperator`

Specialized Python operators: pyPenetrance

pySelector

simulator

mating scheme

evolve!

Exercise time!

simuLDDecay.py

Outline

Illustration

Create a population

Genotypic structure

Individuals

Population strcuture

Information fields

Variables

Outline

Stages

Stages, an example

Output

Table-like output

Outline

Mating schemes

Simulator

Evolve?!

Outline

Outline

Manipulating of HapMap data

Outline