Forward-time simulations using simuPOP, a tutorial

Bo Peng, Ph.D.

Department of Epidemiology
U.T. M.D. Anderson Cancer Center
Houston, TX

June 15th, 2007
simuPOP workshop
School of Public Health, Department of Biostatistics
University of Alabama Birmingham

outline

Outline

1 What is simuPOP

- Forward- and backward-time simulation
- Features of simuPOP
- Applications
- Availability

simuPOP is ...

A forward-time population genetics simulation environment

simuPOP is ...

A forward-time population genetics simulation environment

- A population genetics simulation program

simuPOP is ...

A forward-time population genetics simulation environment

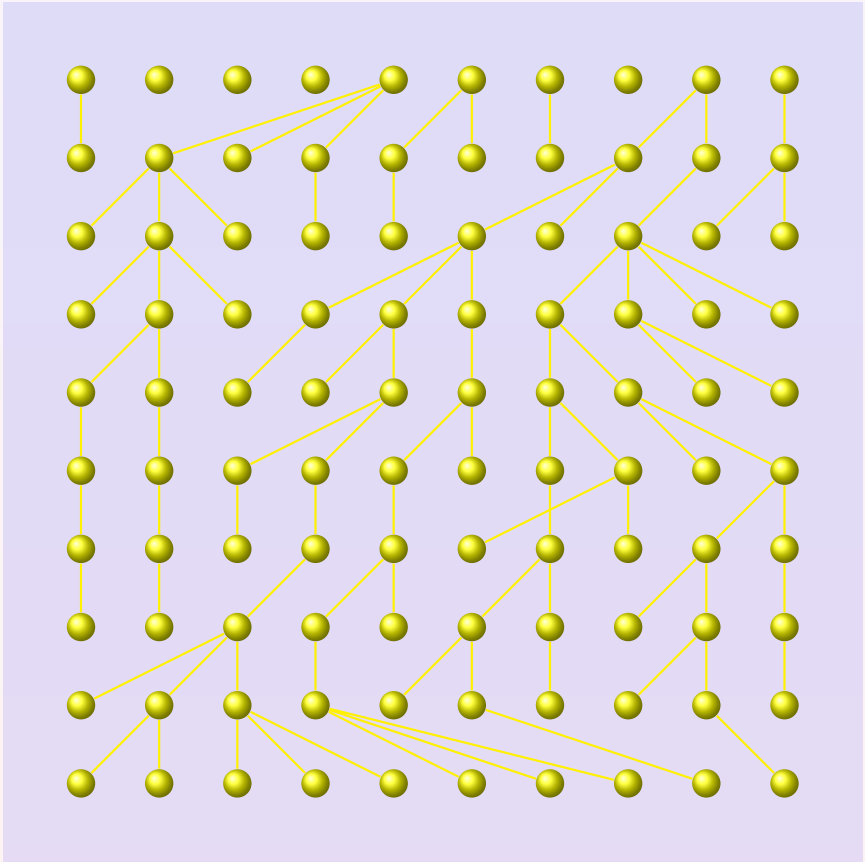
- A population genetics simulation program
- Not coalescent-based

simuPOP is ...

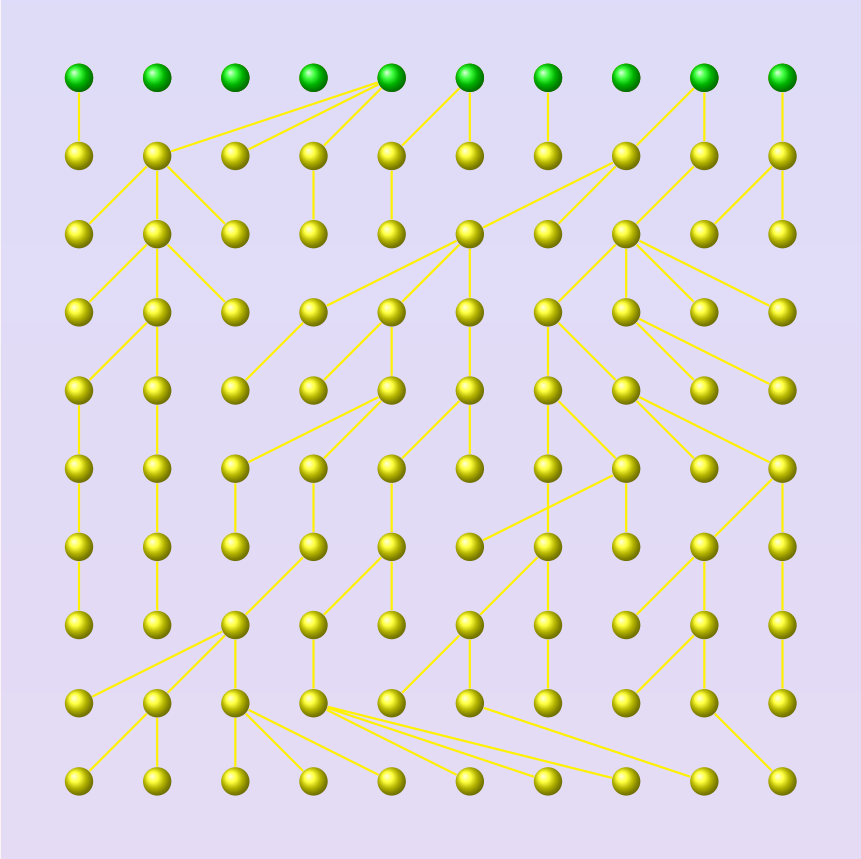
A forward-time population genetics simulation environment

- A population genetics simulation program
- Not coalescent-based
- Based on an object-oriented scripting language (Python)

Forward-time simulation

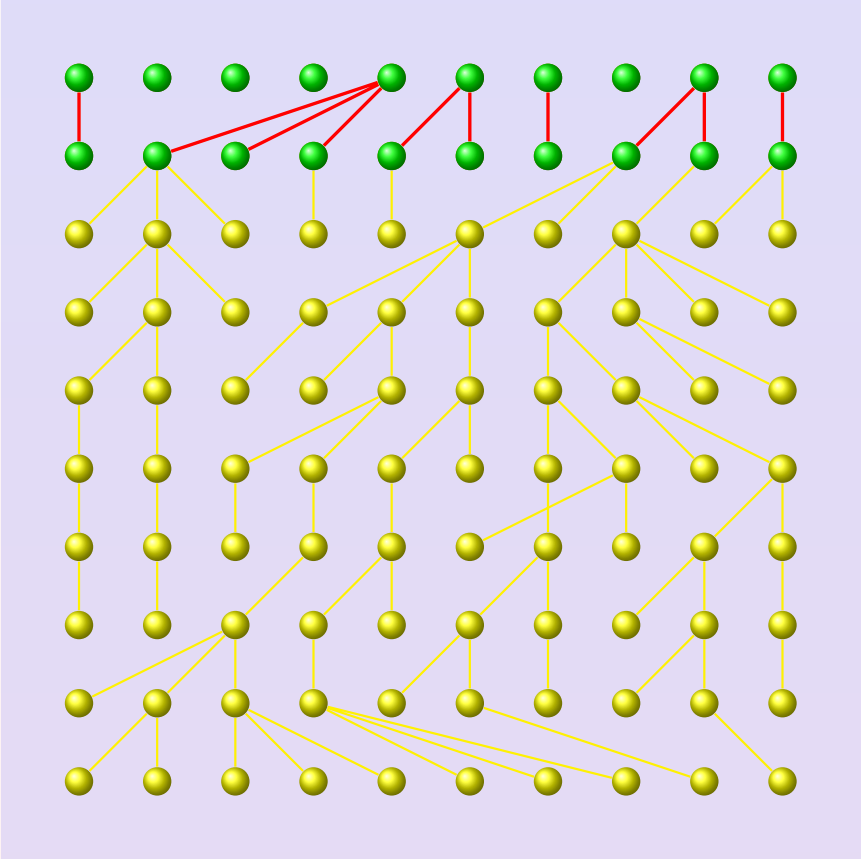


Forward-time simulation



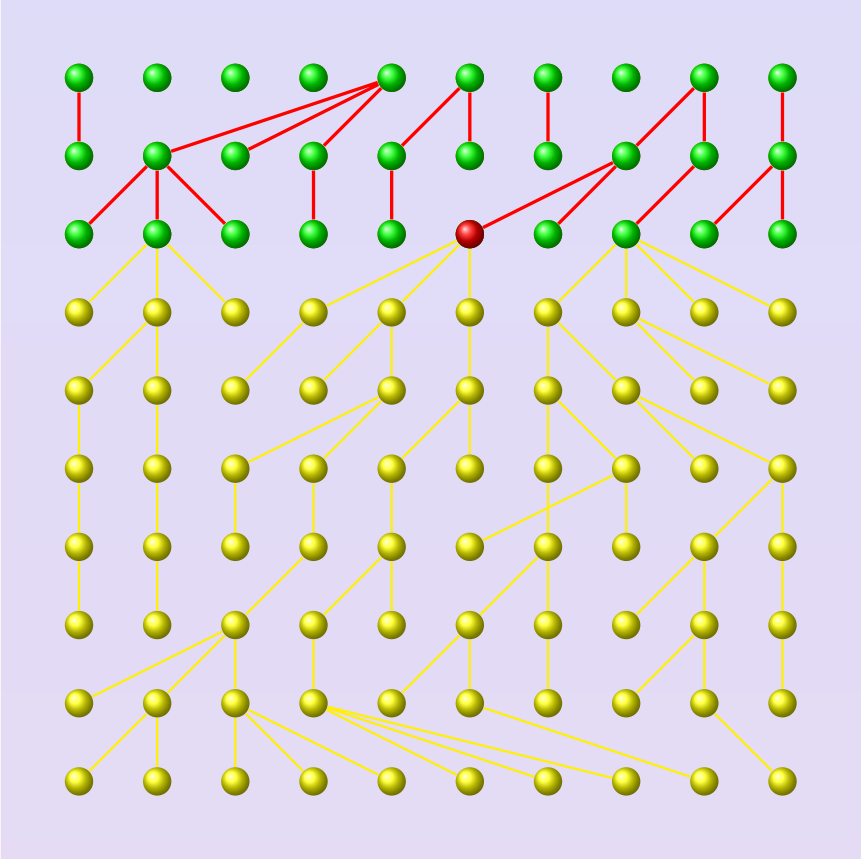
- Start from an initial population

Forward-time simulation



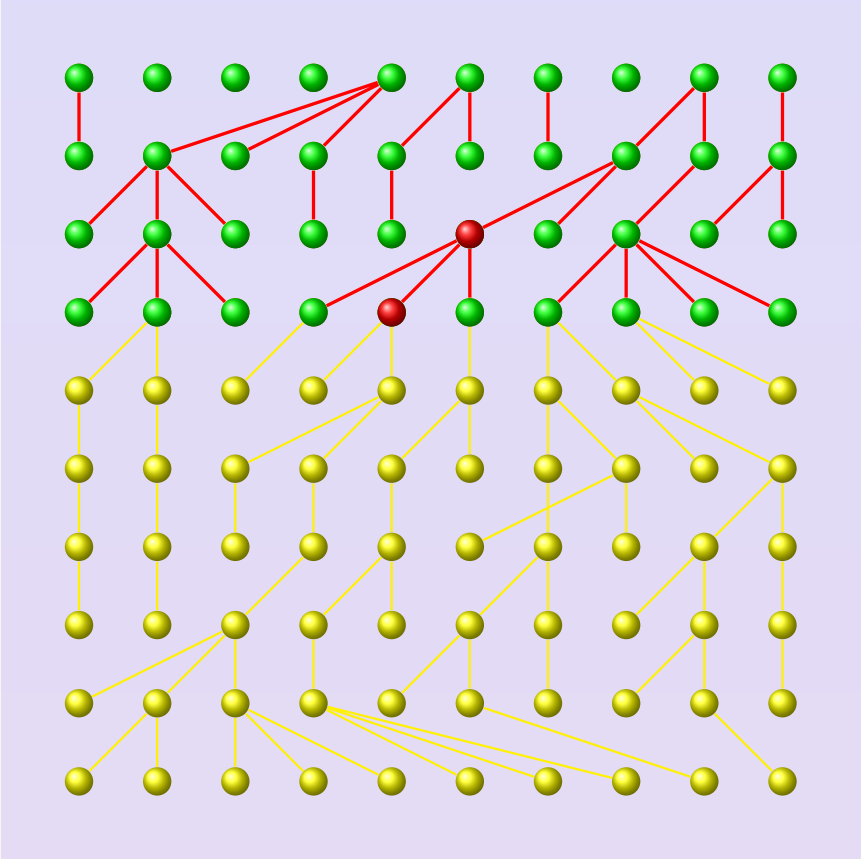
- Start from an initial population
- Evolve forward in time, generation by generation, subject to certain number of genetic and/or demographic effects

Forward-time simulation



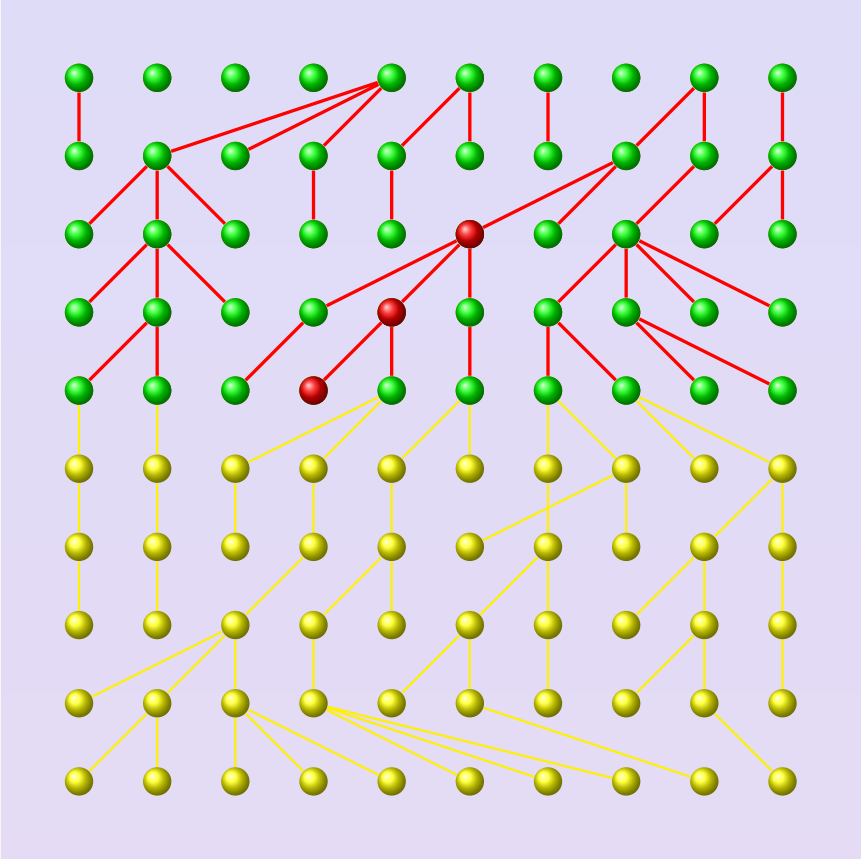
- Start from an initial population
- Evolve forward in time, generation by generation, subject to certain number of genetic and/or demographic effects

Forward-time simulation



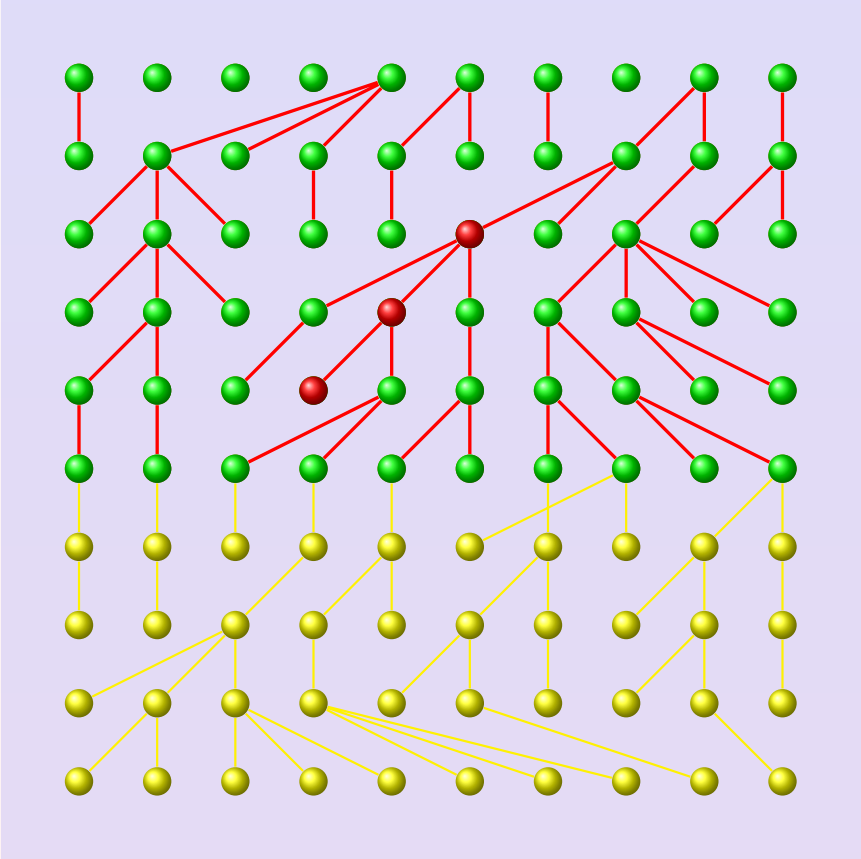
- Start from an initial population
- Evolve forward in time, generation by generation, subject to certain number of genetic and/or demographic effects

Forward-time simulation



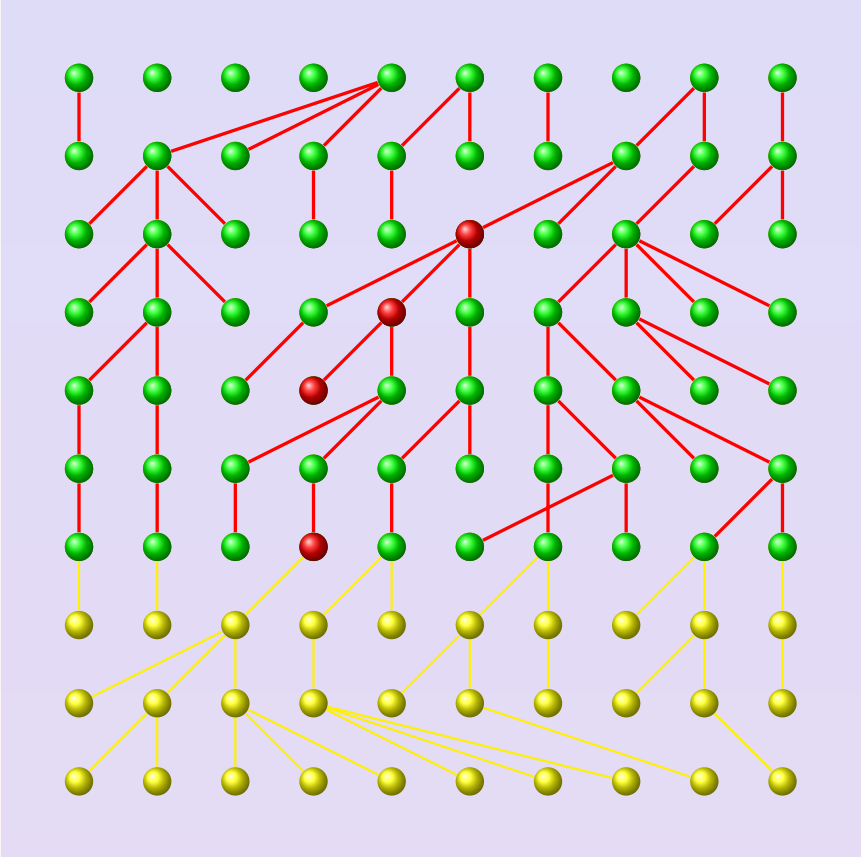
- Start from an initial population
- Evolve forward in time, generation by generation, subject to certain number of genetic and/or demographic effects

Forward-time simulation



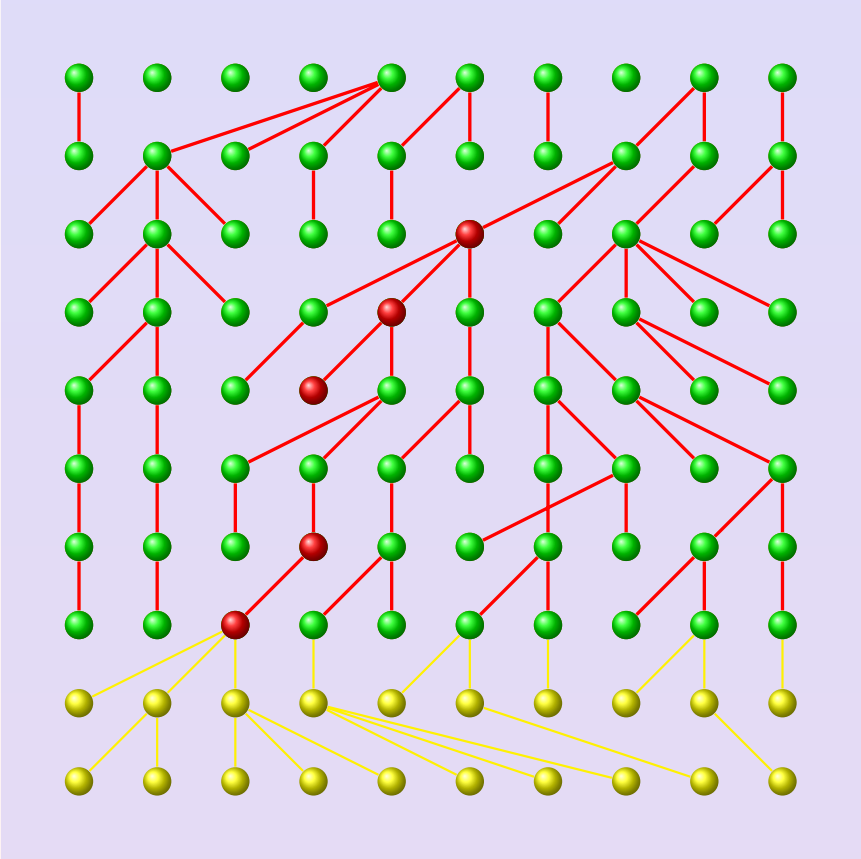
- Start from an initial population
- Evolve forward in time, generation by generation, subject to certain number of genetic and/or demographic effects

Forward-time simulation



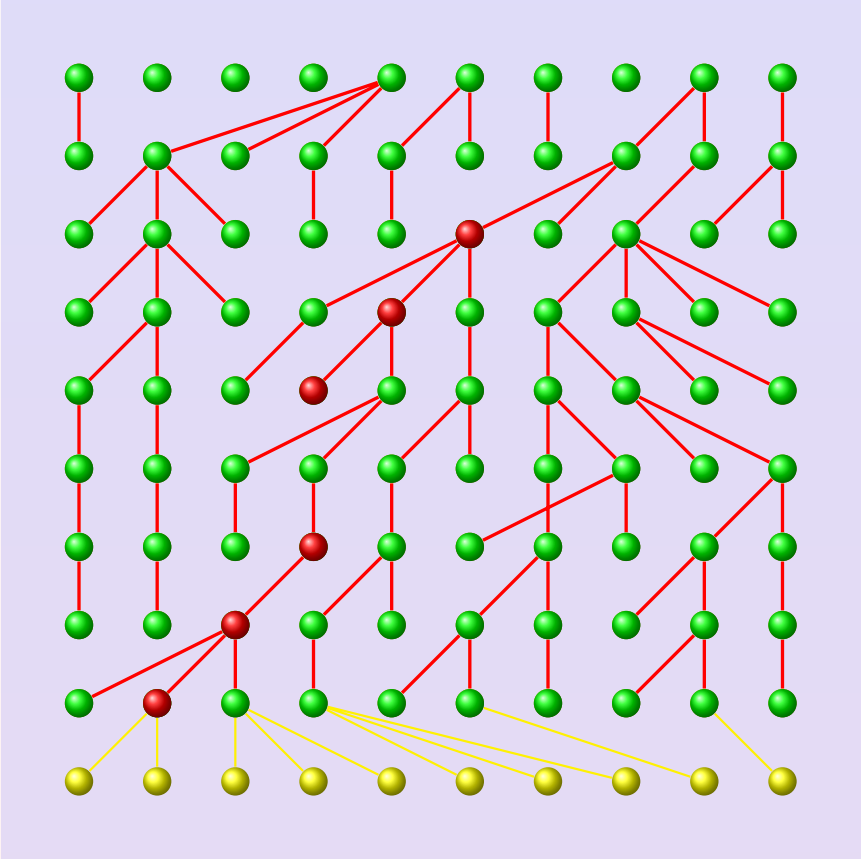
- Start from an initial population
- Evolve forward in time, generation by generation, subject to certain number of genetic and/or demographic effects

Forward-time simulation



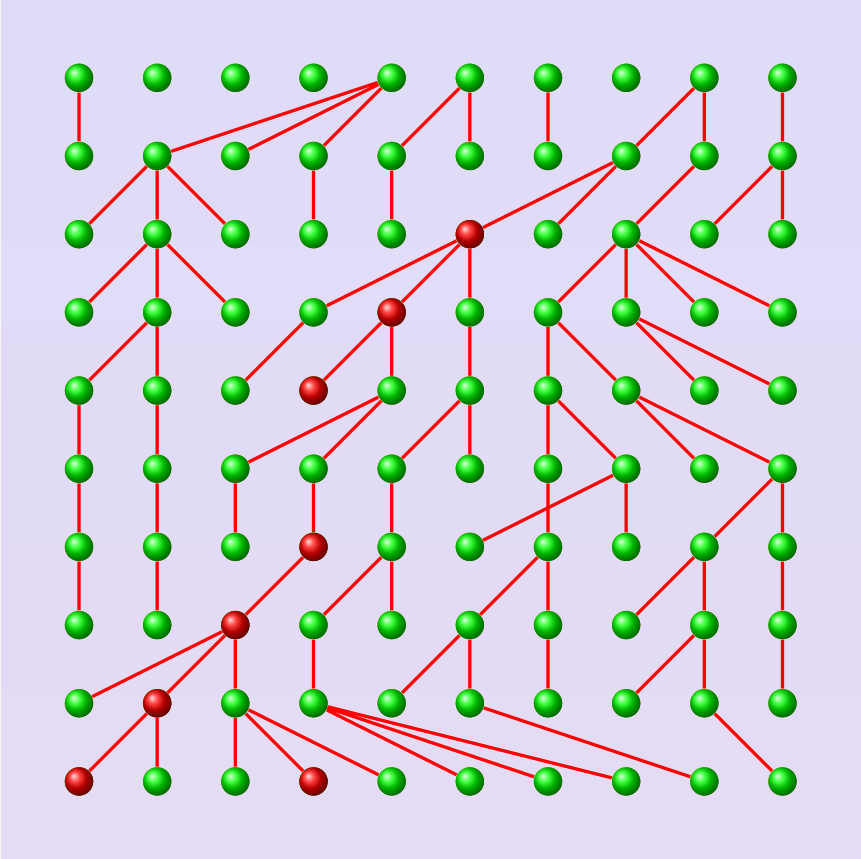
- Start from an initial population
- Evolve forward in time, generation by generation, subject to certain number of genetic and/or demographic effects

Forward-time simulation



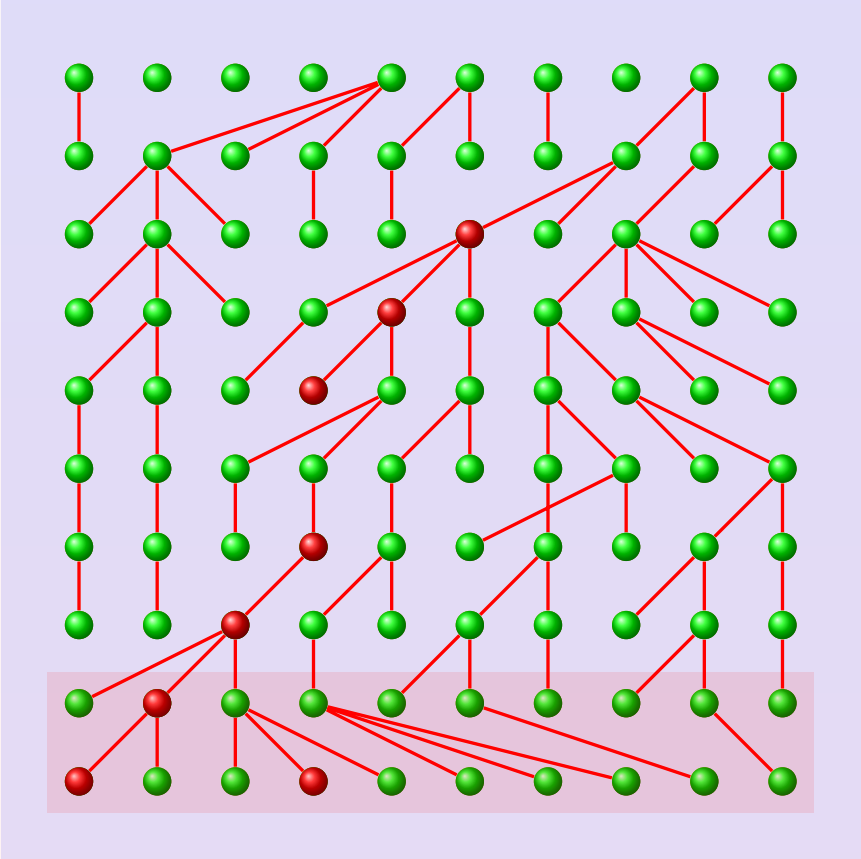
- Start from an initial population
- Evolve forward in time, generation by generation, subject to certain number of genetic and/or demographic effects

Forward-time simulation



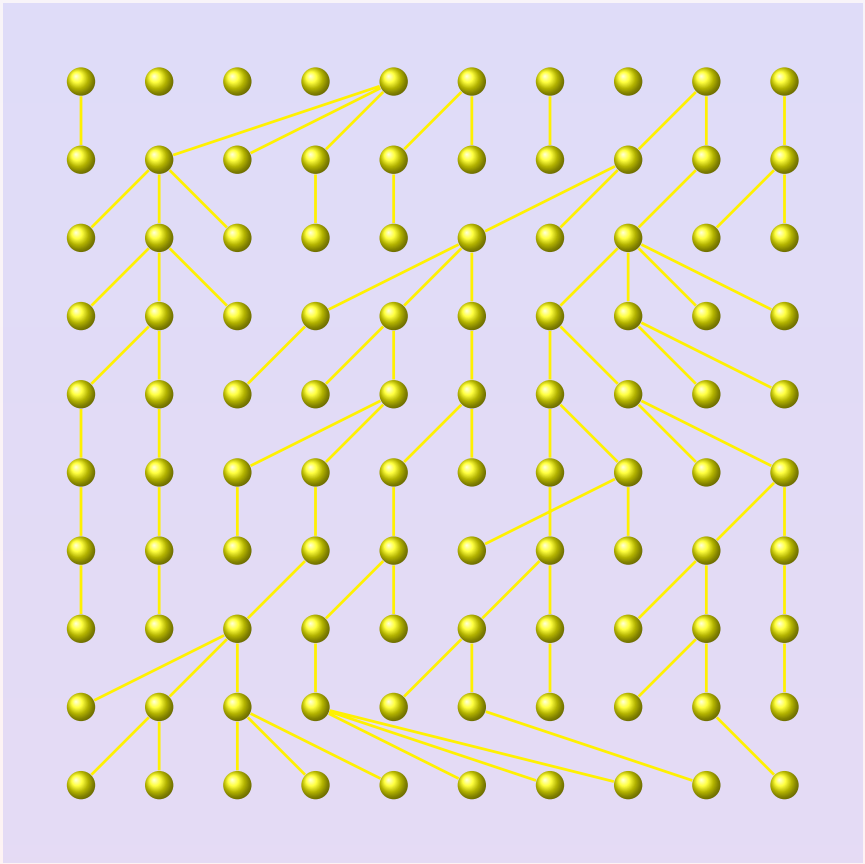
- Start from an initial population
- Evolve forward in time, generation by generation, subject to certain number of genetic and/or demographic effects

Forward-time simulation

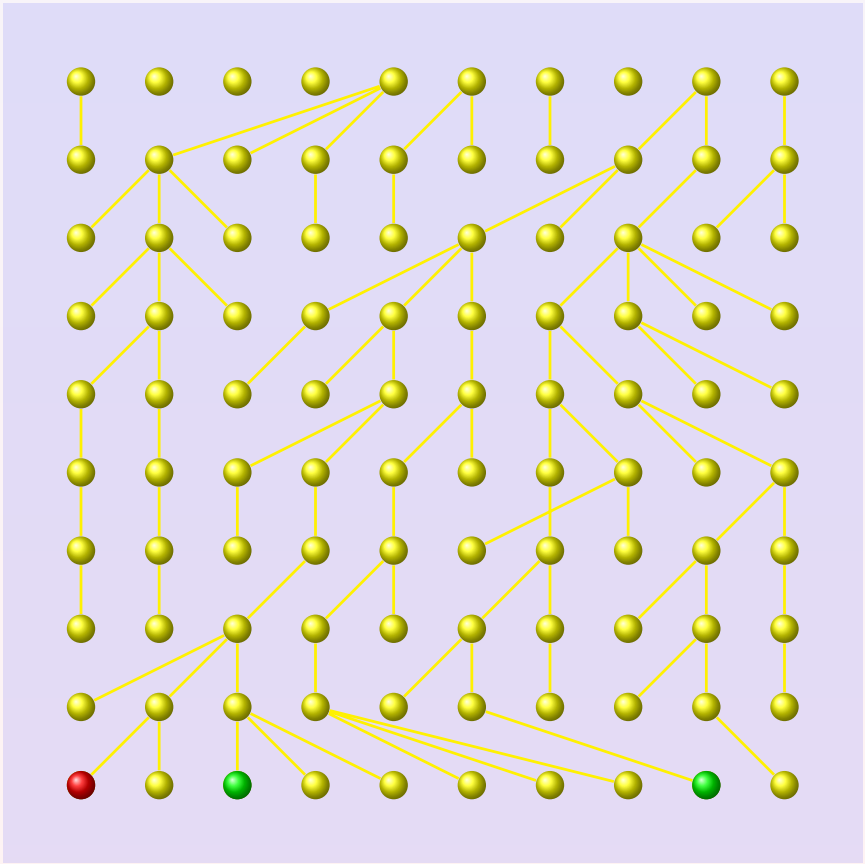


- Start from an initial population
- Evolve forward in time, generation by generation, subject to certain number of genetic and/or demographic effects
- Samples are collected from the last several generations

Backward-time simulation

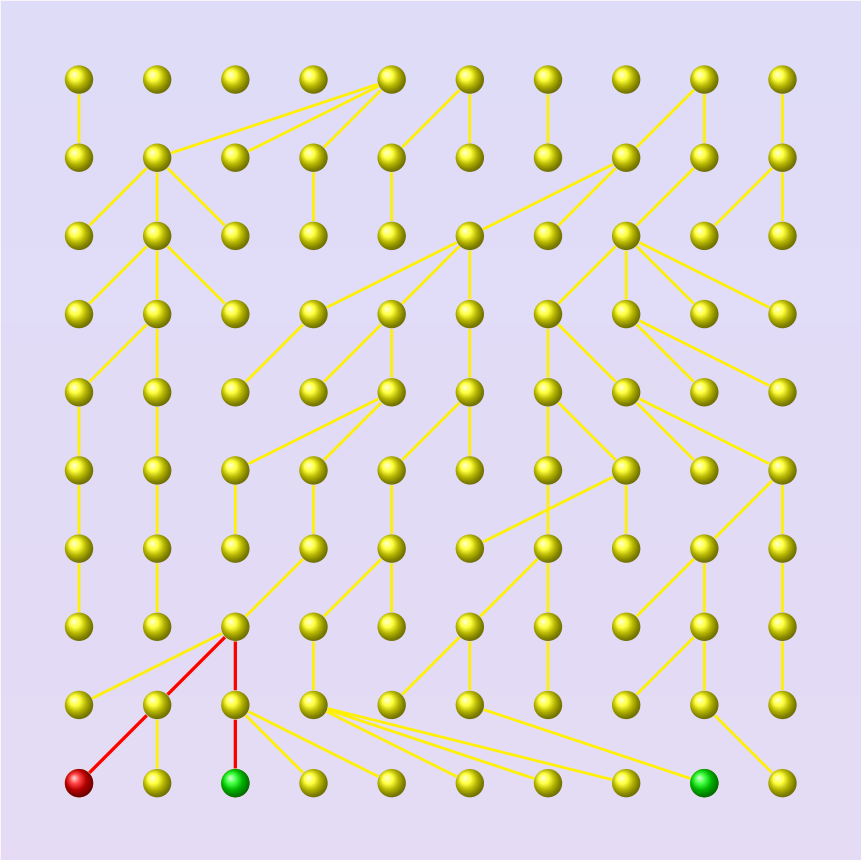


Backward-time simulation



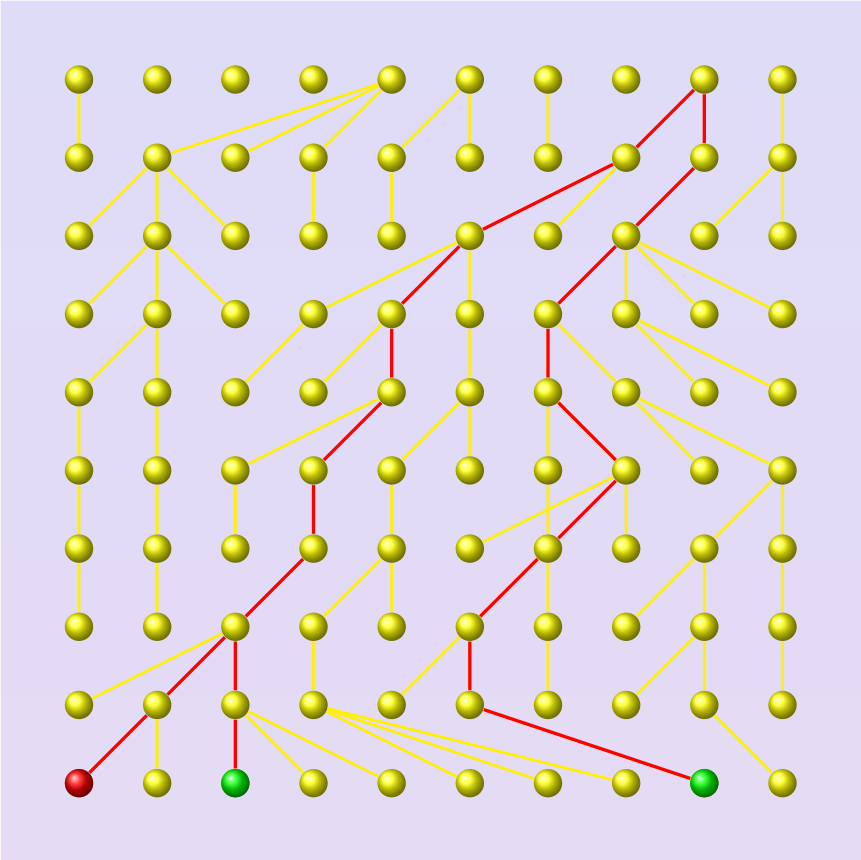
- Start from a sample with unknown genotype

Backward-time simulation



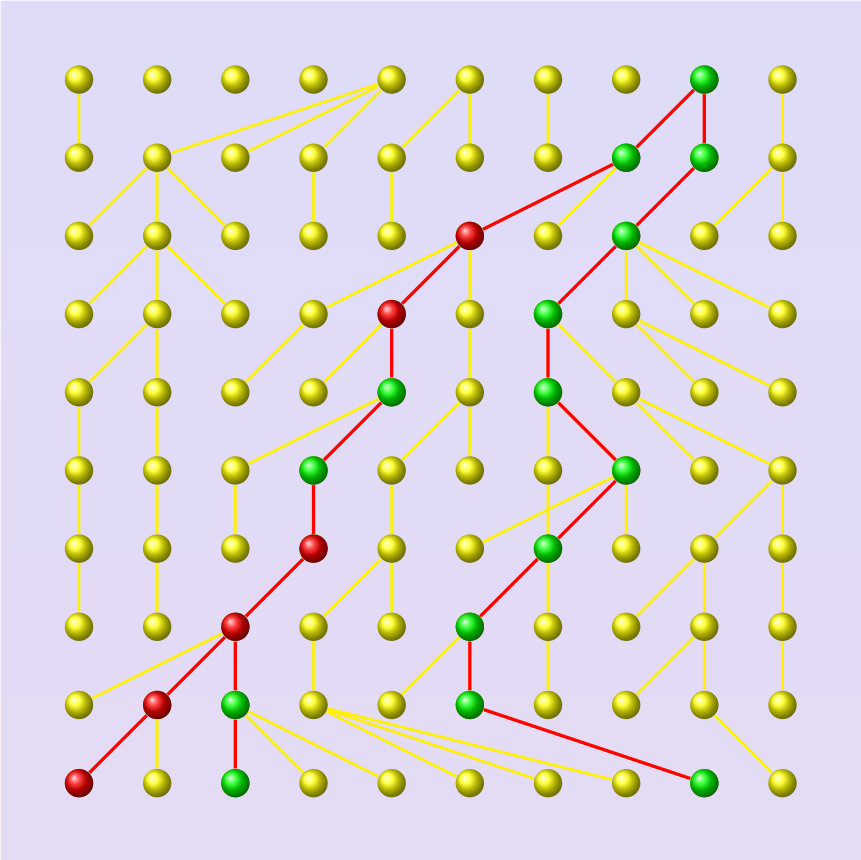
- Start from a sample with unknown genotype
- Coalesce individuals until the most recent common ancestor of all individuals is found

Backward-time simulation



- Start from a sample with unknown genotype
- Coalesce individuals until the most recent common ancestor of all individuals is found

Backward-time simulation



- Start from a sample with unknown genotype
- Coalesce individuals until the most recent common ancestor of all individuals is found
- Starting from the MRCA, proceed forward in time and fill the genotype of each individual

Forward vs. backward-time simulations

Backward-time

- Sample based, efficient

Forward-time

- Population based, inefficient

Forward vs. backward-time simulations

Backward-time

- Sample based, efficient
- Limited selection, recombination models and mating schemes

Forward-time

- Population based, inefficient
- Can simulate almost arbitrary evolutionary scenarios

Forward vs. backward-time simulations

Backward-time

- Sample based, efficient
- Limited selection, recombination models and mating schemes
- Can not study population properties, or properties of ancestral generations

Forward-time

- Population based, inefficient
- Can simulate almost arbitrary evolutionary scenarios
- Can study population properties and ancestral generations

Forward vs. backward-time simulations

Backward-time

- Sample based, efficient
- Limited selection, recombination models and mating schemes
- Can not study population properties, or properties of ancestral generations
- Used mostly for sample generation

Forward-time

- Population based, inefficient
- Can simulate almost arbitrary evolutionary scenarios
- Can study population properties and ancestral generations
- Not limited to sample generation

Forward-time simulation programs

For specific applications

- Easy to write simple simulations
- Difficult to write complicated simulations
- A few programs are available (e.g. EasyPOP, FPG, Nemo), easy to use if they happen to fit your need

Forward-time simulation programs

For specific applications

- Easy to write simple simulations
- Difficult to write complicated simulations
- A few programs are available (e.g. EasyPOP, FPG, Nemo), easy to use if they happen to fit your need

For general purposes

- Difficult to write
- Easy to set up complicated simulations
- simuPOP fits in this category

What simuPOP does

simuPOP provides

- a large number of functions to manipulate populations
  copy, split, merge, manipulate individual genotypes, determine affection status, save to and load from various formats, generate sample, ...

What simuPOP does

simuPOP provides

- a large number of functions to manipulate populations copy, split, merge, manipulate individual genotypes, determine affection status, save to and load from various formats, generate sample, ...
- and a mechanism to evolve populations forward in time subject to arbitrary demographic and genetic forces such as population size changes, mutation, migration, recombination, selection, ...

Structure of individuals

Assume ploidy = 2, maxAllele = 1

0	1	2		3	4			5	6
0	1	1		1	0			0	1
0	0	1		1	1			0	1

0			1	2	3	4	5
0			1	0	0	0	1
1			0	1	1	0	0

Male

- Affected

fitness	father_id	···

Structure of individuals

Assume ploidy = 2, maxAllele = 1

0	1	2		3	4			5	6
0	1	1		1	0			0	1
0	0	1		1	1			0	1

0			1	2	3	4	5
0			1	0	0	0	1
1			0	1	1	0	0

Chromosome 0

Male

- Affected

fitness	father_id	···

Structure of individuals

Assume ploidy = 2, maxAllele = 1

0	1	2		3	4			5	6
0	1	1		1	0			0	1
0	0	1		1	1			0	1

Chromosome 0

0	1	2	3	4	5
0	1	0	0	0	1
1	0	1	1	0	0

Chromosome 1

Male

- Affected

fitness	father_id	···

Structure of individuals

Assume ploidy = 2, maxAllele = 1

0	1	2		3	4			5	6
0	1	1		1	0			0	1
0	0	1		1	1			0	1

Chromosome 0

0			1	2	3	4	5
0			1	0	0	0	1
1			0	1	1	0	0

Chromosome 1

Male

Sex

- Affected

fitness	father_id	···

Structure of individuals

Assume ploidy = 2, maxAllele = 1

0	1	2		3	4			5	6
0	1	1		1	0			0	1
0	0	1		1	1			0	1

Chromosome 0

0			1	2	3	4	5
0			1	0	0	0	1
1			0	1	1	0	0

Chromosome 1

Male

Sex

- Affected

Affection status

fitness	father_id	···

Structure of individuals

Assume ploidy = 2, maxAllele = 1

0	1	2		3	4			5	6
0	1	1		1	0			0	1
0	0	1		1	1			0	1

Chromosome 0

0			1	2	3	4	5
0			1	0	0	0	1
1			0	1	1	0	0

Chromosome 1

Male

Sex

- Affected

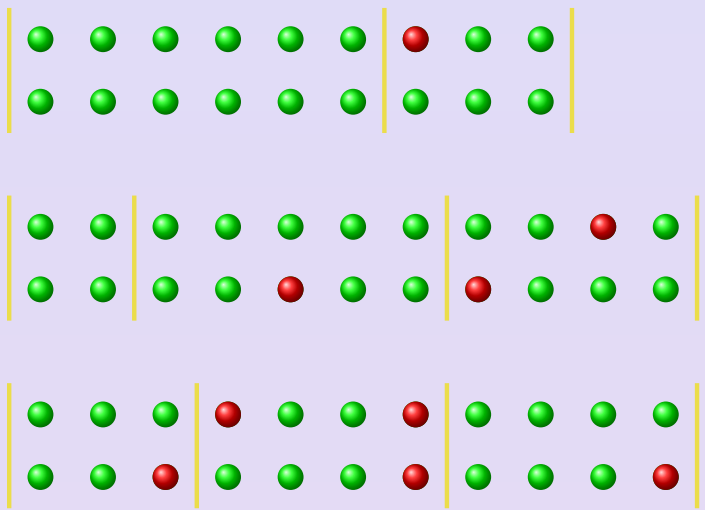
Affection status

fitness	father_id	···

Information fields

Structure of populations

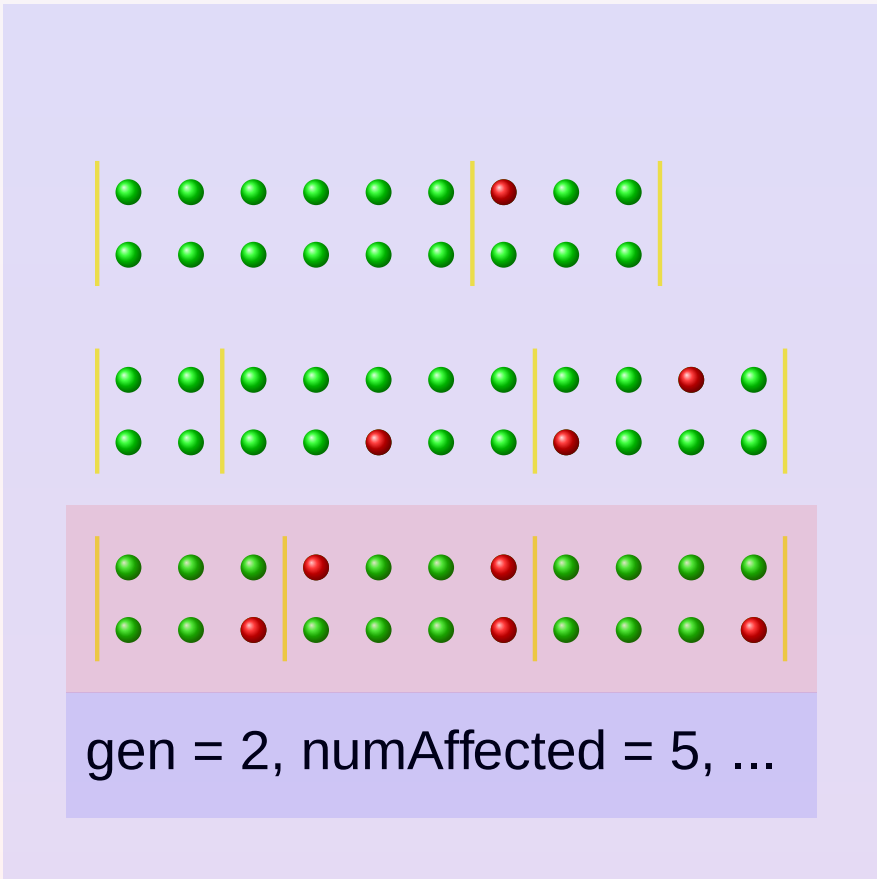
- Unaffected
- Affected



gen = 2, numAffected = 5, ...

Structure of populations

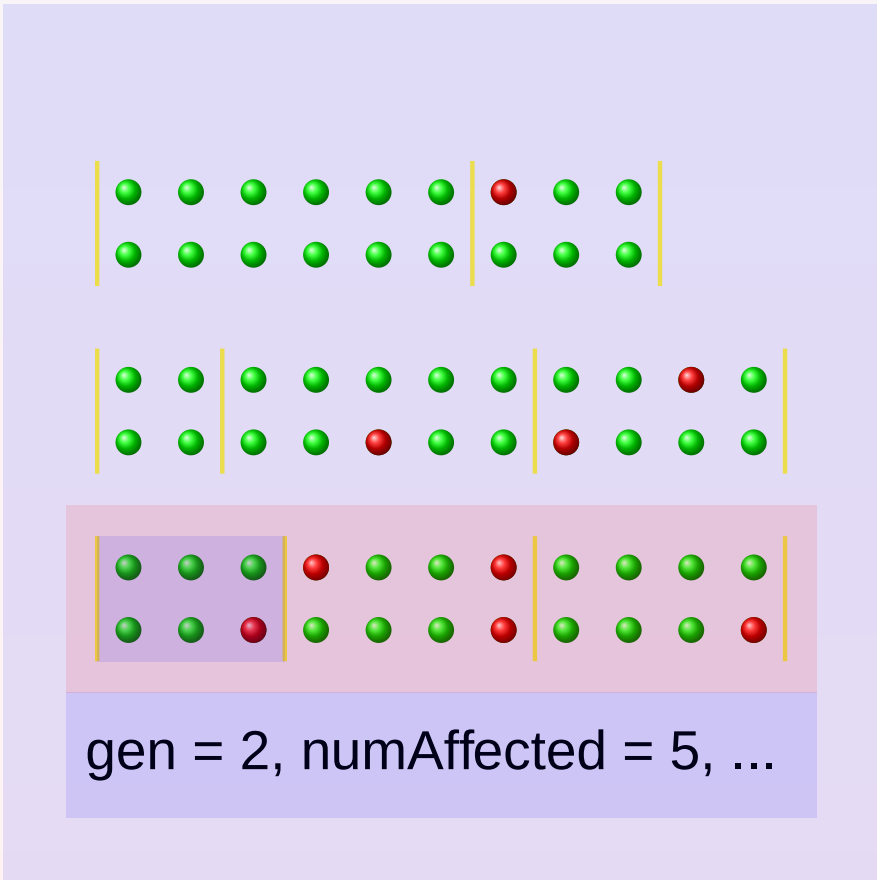
- Unaffected
- Affected



Current generation

Structure of populations

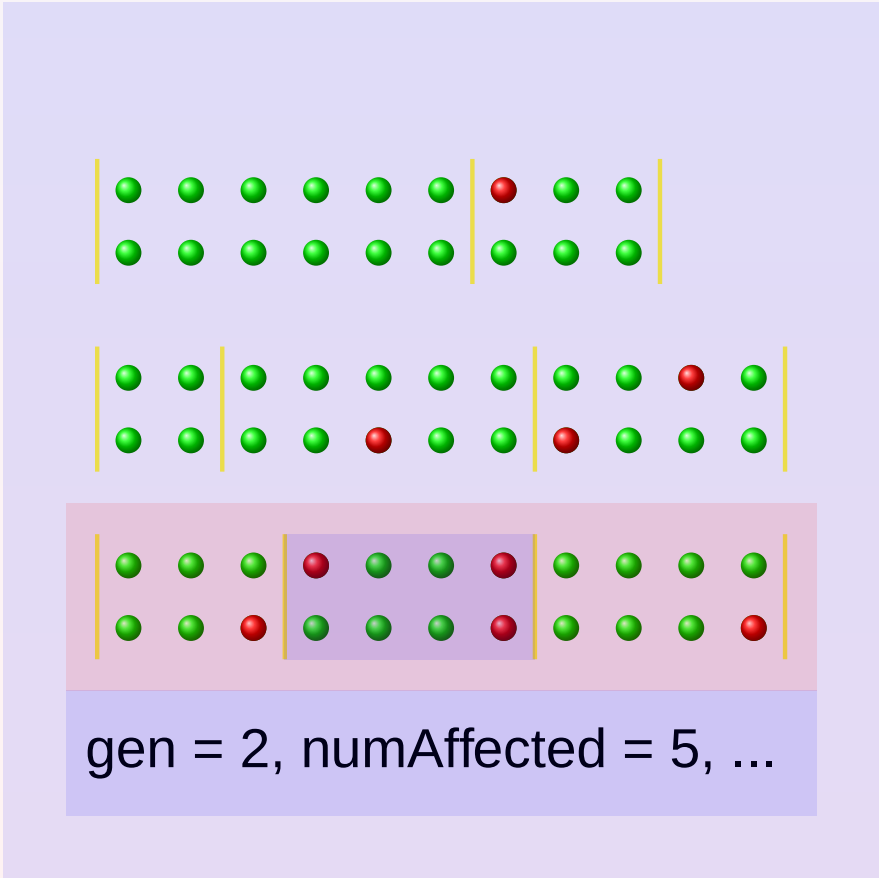
- Unaffected
- Affected



Current generation

Structure of populations

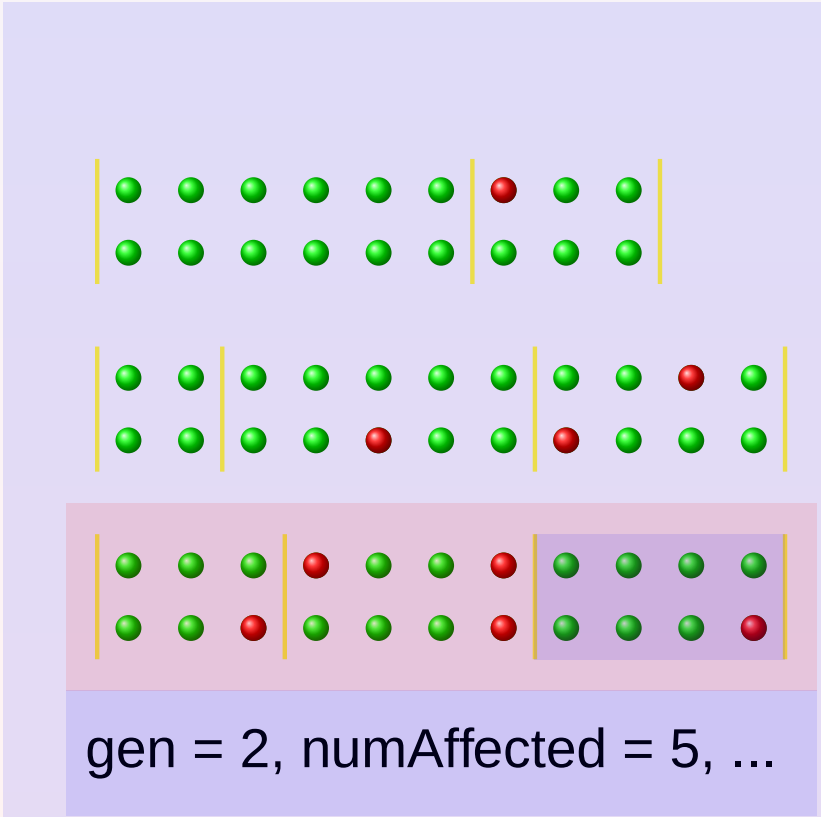
- Unaffected
- Affected



Current generation

Structure of populations

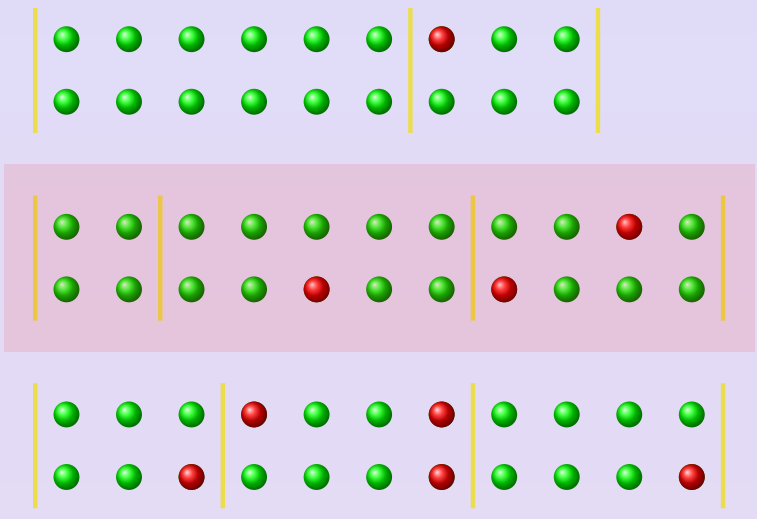
- Unaffected
- Affected



Current generation

Structure of populations

- Unaffected
- Affected

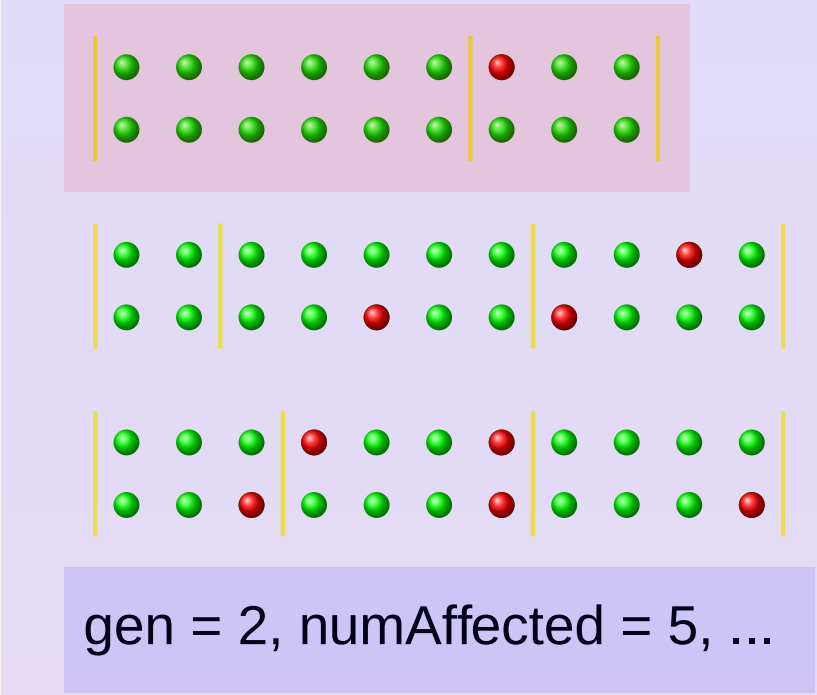


Ancestral generation 1

Current generation

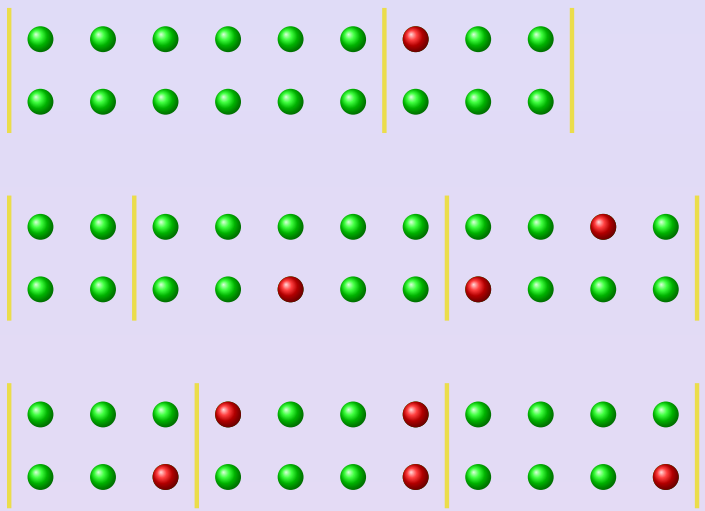
Structure of populations

- Unaffected
- Affected



Structure of populations

- Unaffected
- Affected



Ancestral generation 2

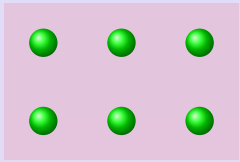
Ancestral generation 1

Current generation

gen = 2, numAffected = 5, ...

Population variables

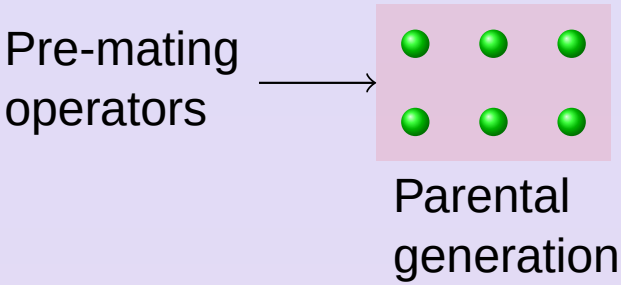
The evolutionary process



Parental generation

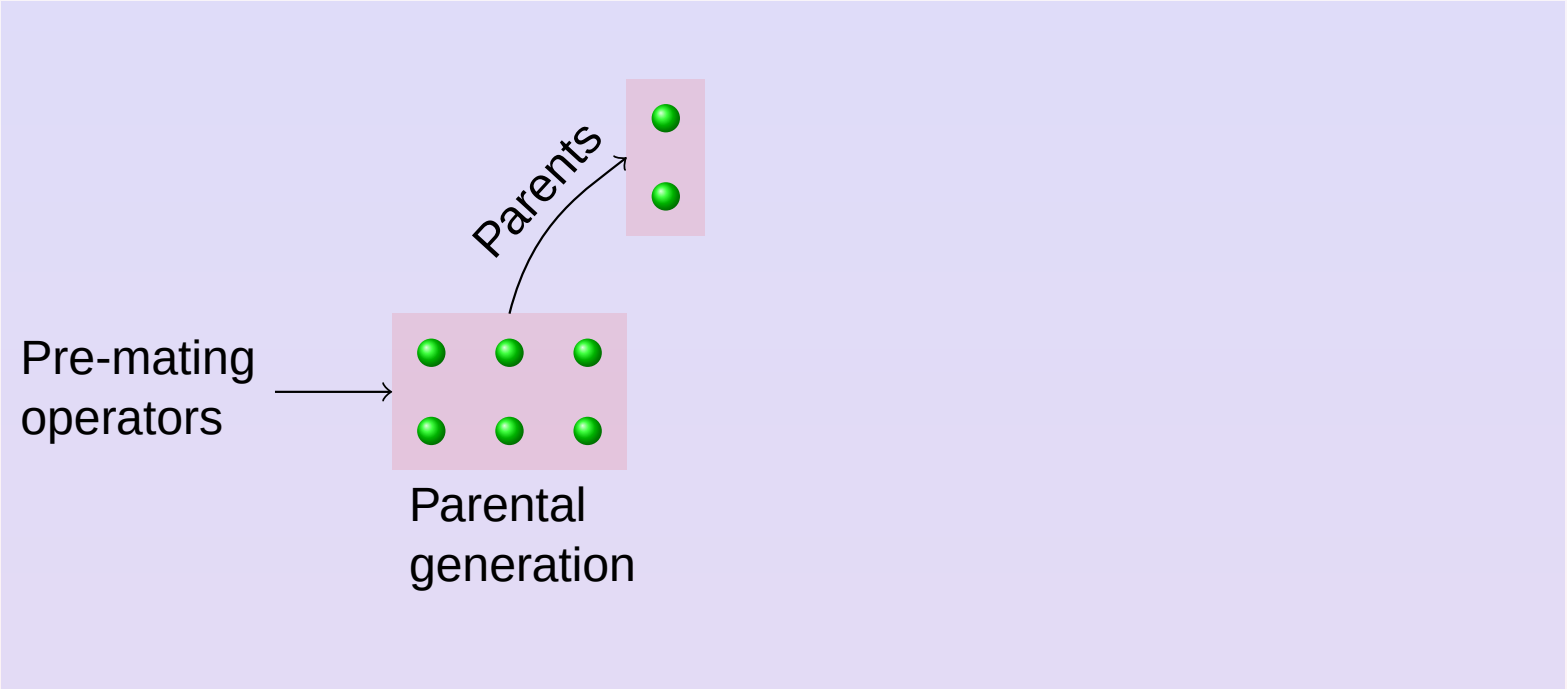
Involved simuPOP objects: population and individual, operator, mating scheme, simulator

The evolutionary process



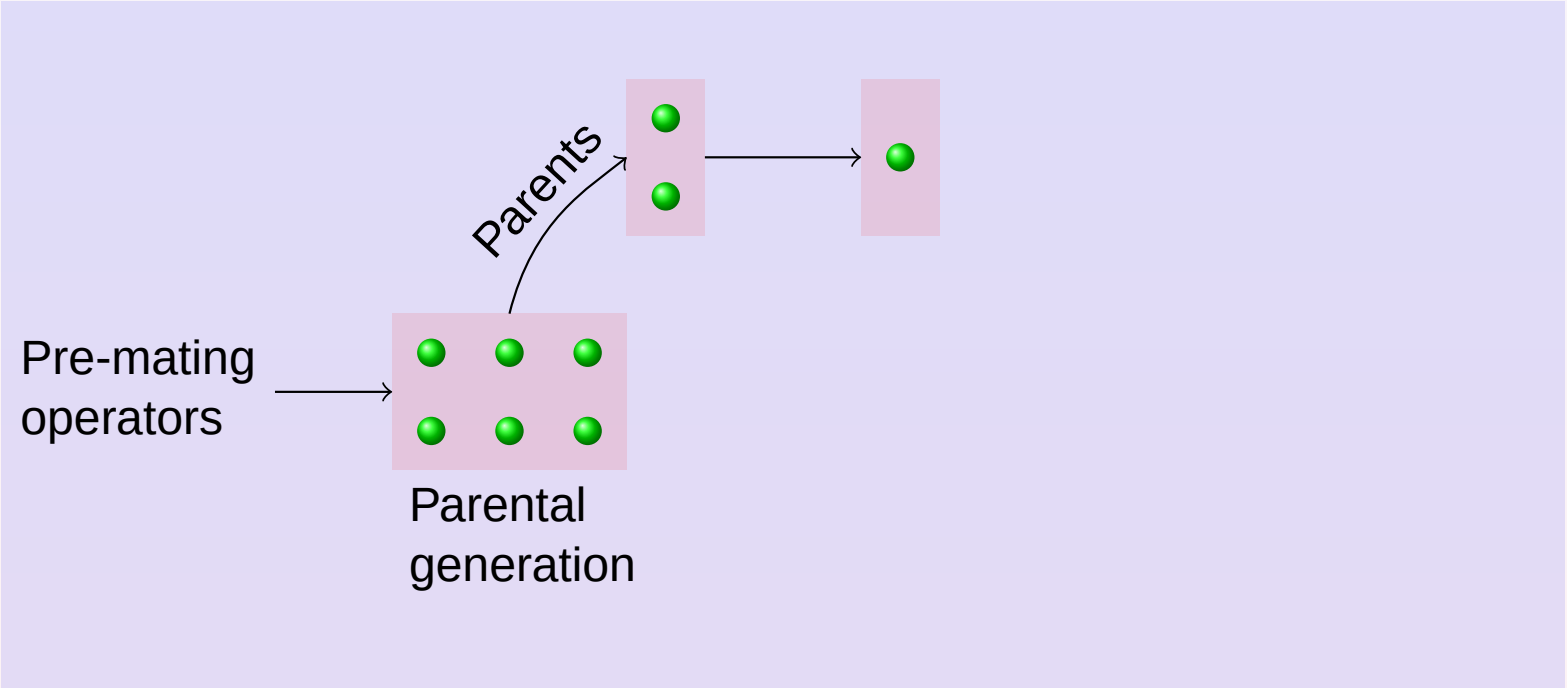
Involved simuPOP objects: population and individual, operator, mating scheme, simulator

The evolutionary process



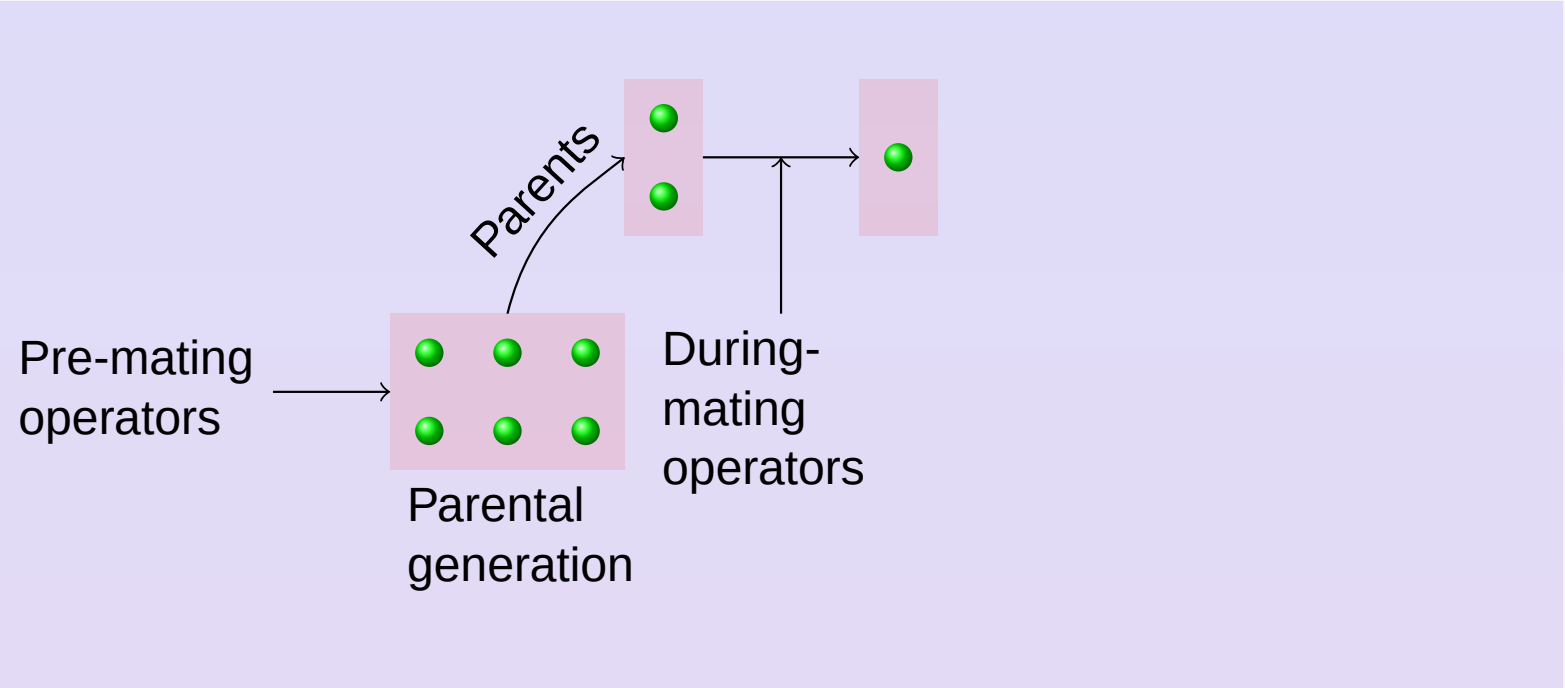
Involved simuPOP objects: population and individual, operator, mating scheme, simulator

The evolutionary process



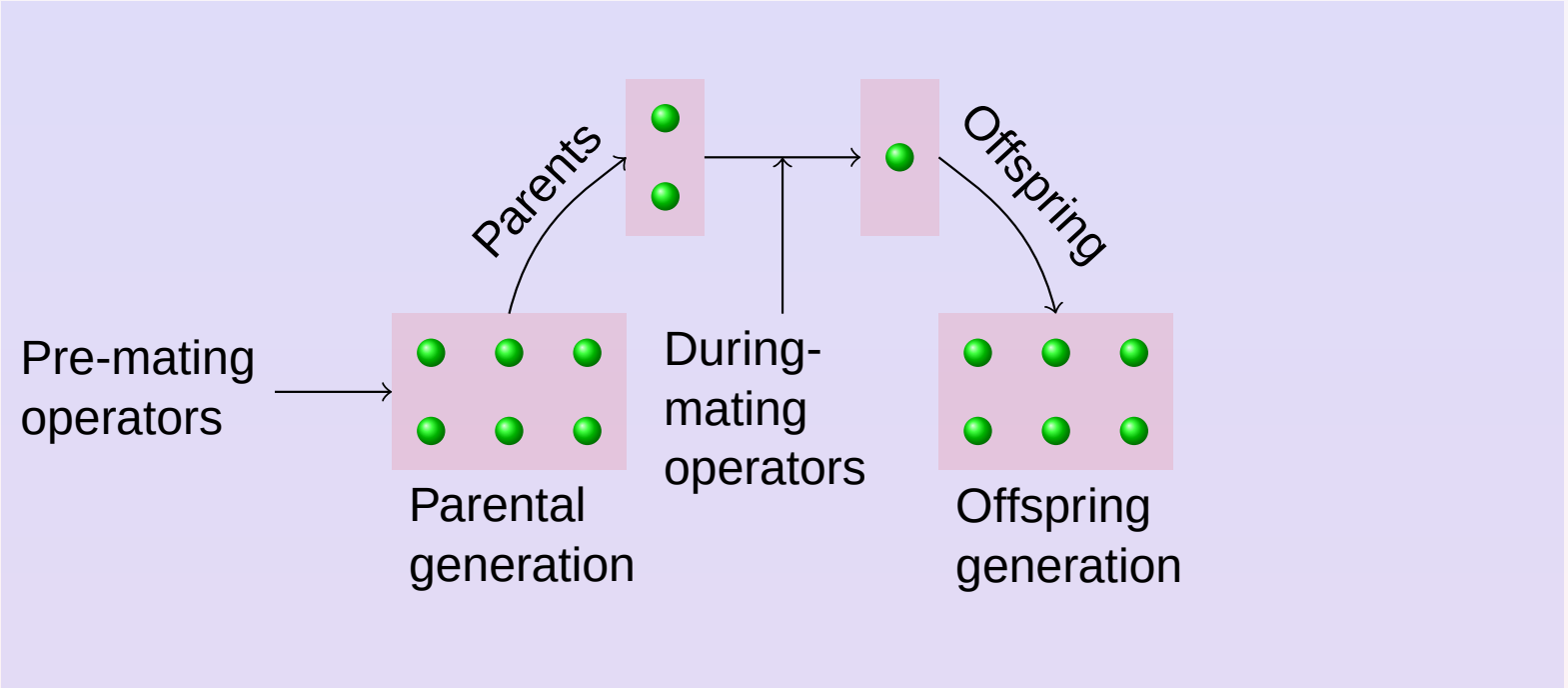
Involved simuPOP objects: population and individual, operator, mating scheme, simulator

The evolutionary process



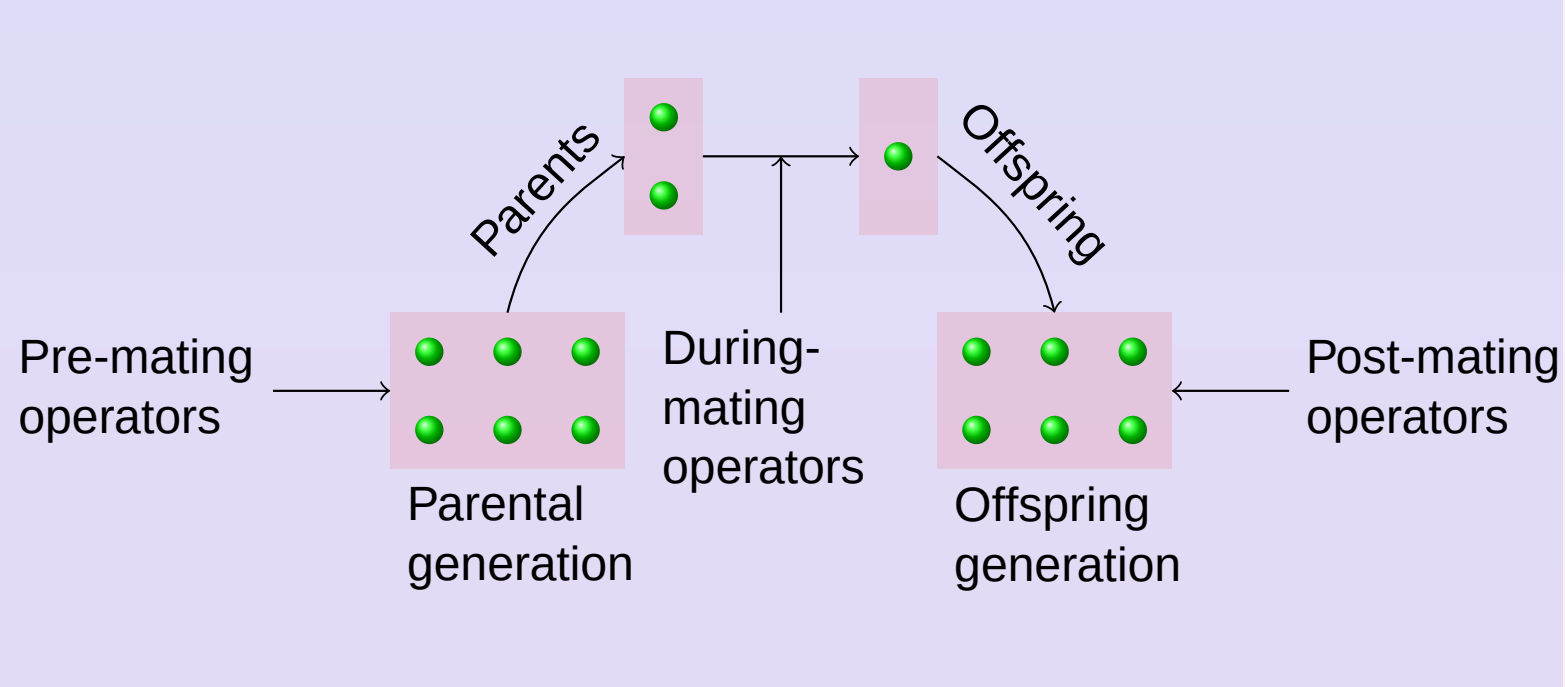
Involved simuPOP objects: population and individual, operator, mating scheme, simulator

The evolutionary process



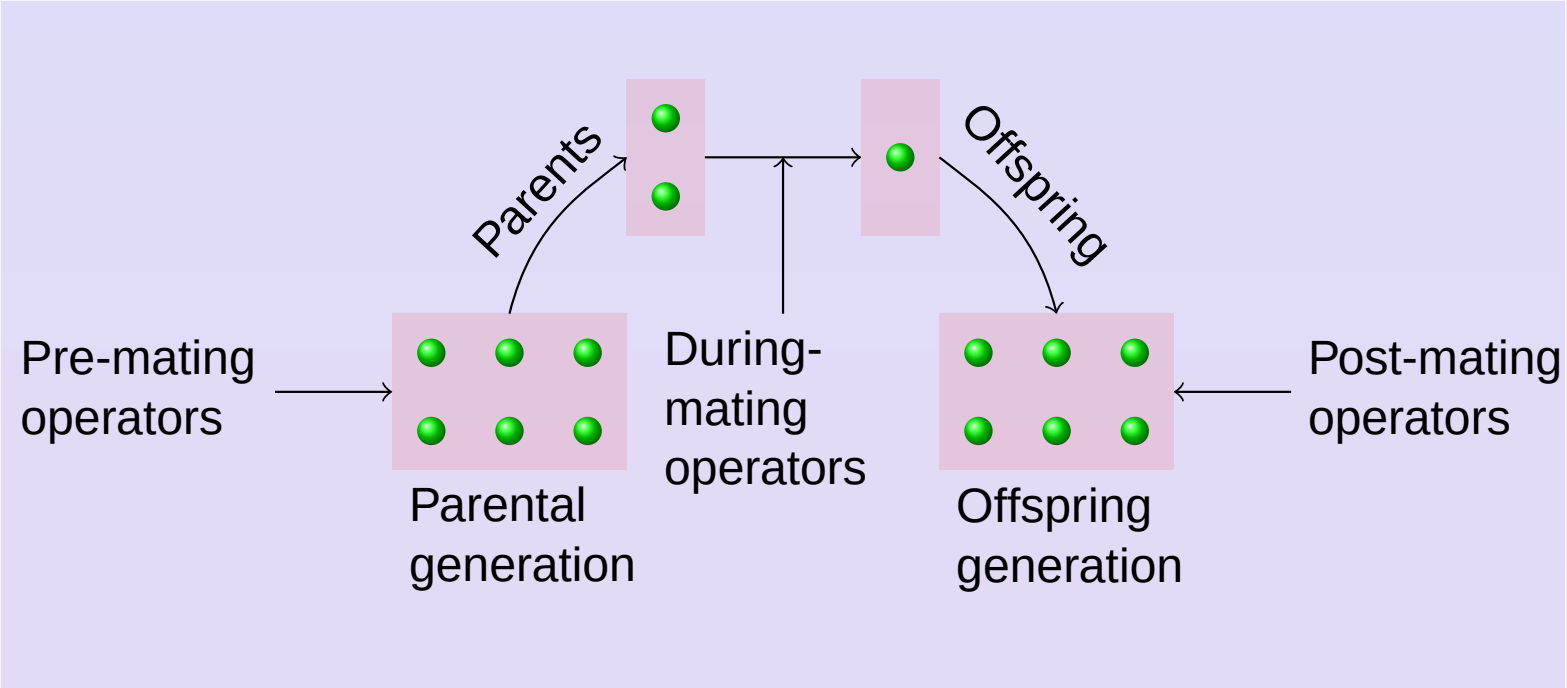
Involved simuPOP objects: population and individual, operator, mating scheme, simulator

The evolutionary process



Involved simuPOP objects: population and individual, operator, mating scheme, simulator

The evolutionary process



Involved simuPOP objects: population and individual, operator, mating scheme, simulator

What distinguishes simuPOP from others

scripting simuPOP is provided as a set of Python modules, and is therefore backed by a full-blown object-oriented programming language.

flexibility simuPOP does not impose any limit on the size of genome, population, demographic model, etc. Using a large number of standard and hybrid (Python-assisted) operators, users can simulate almost arbitrarily complex evolutionary processes.

integration Owing to the 'glue language' nature of Python, it is easy to integrate simuPOP with other languages and programs.

I like it, but, oohm, why Python??

- The core of simuPOP is written in C++ for efficiency
- Python is the glue language, a wrapper of the core
- Python is used to write simuPOP extensions (user interface etc)
- The core sometimes calls Python (Python operators) for maximum flexibility

Do I have to write a script?

simuPOP can be used in two ways:

- You should learn how to write simuPOP scripts if you
  - need a particular type of simulation for you own research, and
  - know exactly what you want to do
- You can use existing simuPOP scripts without knowing simuPOP if
  - you need to use an existing simulation scenario to simulate samples or populations
  - this scenario is implemented in simuPOP

This is fun, but is it useful?

simuPOP can simulate the change of the genetic composition of a population in a complicated evolutionary process. It can be used to

- Demonstrate population genetics phenomena

This is fun, but is it useful?

simuPOP can simulate the change of the genetic composition of a population in a complicated evolutionary process. It can be used to

- Demonstrate population genetics phenomena
- Study the impact of genetic and demographic forces on the evolution of a population

This is fun, but is it useful?

simuPOP can simulate the change of the genetic composition of a population in a complicated evolutionary process. It can be used to

- Demonstrate population genetics phenomena
- Study the impact of genetic and demographic forces on the evolution of a population
- Study the evolution of (complex) genetic diseases

This is fun, but is it useful?

simuPOP can simulate the change of the genetic composition of a population in a complicated evolutionary process. It can be used to

- Demonstrate population genetics phenomena
- Study the impact of genetic and demographic forces on the evolution of a population
- Study the evolution of (complex) genetic diseases
- Simulate samples to validate gene-mapping methods

This is fun, but is it useful?

simuPOP can simulate the change of the genetic composition of a population in a complicated evolutionary process. It can be used to

- Demonstrate population genetics phenomena
- Study the impact of genetic and demographic forces on the evolution of a population
- Study the evolution of (complex) genetic diseases
- Simulate samples to validate gene-mapping methods
- Study ascertainment methods in simulated populations

This is fun, but is it useful?

simuPOP can simulate the change of the genetic composition of a population in a complicated evolutionary process. It can be used to

- Demonstrate population genetics phenomena
- Study the impact of genetic and demographic forces on the evolution of a population
- Study the evolution of (complex) genetic diseases
- Simulate samples to validate gene-mapping methods
- Study ascertainment methods in simulated populations
- ...

Simulations of complex human diseases

Backward-time

- Haploid only

Forward-time

- No limit on ploidy

Simulations of complex human diseases

Backward-time

- Haploid only
- Additive selection and penetrance models

Forward-time

- No limit on ploidy
- Arbitrary selection and penetrance models

Simulations of complex human diseases

Backward-time

- Haploid only
- Additive selection and penetrance models
- One disease susceptibility locus

Forward-time

- No limit on ploidy
- Arbitrary selection and penetrance models
- Multiple DSL with interaction

Simulations of complex human diseases

Backward-time

- Haploid only
- Additive selection and penetrance models
- One disease susceptibility locus
- Generate independent samples of fixed format

Forward-time

- No limit on ploidy
- Arbitrary selection and penetrance models
- Multiple DSL with interaction
- Generate multi-generation populations

Availability

- simuPOP website: `http://simupop.sourceforge.net`
- Mailing list: `simupop-list@lists.sourceforge.net`
- License: GPL 2.0
- Platforms: all OS on which Python is available
- Monthly release, currently at 0.7.10
- Documentation: *simuPOP User's Guide* and *simuPOP Reference Manual*

Outline

2 An example

- An example
- Visualization with R

A simple example


```
>>> from simuPOP import *
>>> simu = simulator(
...     population(size=1000, ploidy=2, loci=[2]),
...     randomMating(),
...     rep = 3)
>>> simu.evolve(
...     preOps = [initByValue([1,2,2,1])],
...     ops = [
...         recombinator(rate=0.1),
...         stat(LD=[0,1]),
...         pyEval(r"'%3d    ' % gen", rep=0, step=10),
...         pyEval(r"'%f    ' % LD[0][1]", step=10),
...         pyEval(r"'\n'", rep=REP_LAST, step=10)
...     ],
...     end=100
... )
```

Loading simuPOP module


```
>>> from simuPOP import *
>>> simu = simulator(
...     population(size=1000, ploidy=2, loci=[2]),
...     randomMating(),
...     rep = 3)
```

Import the default simuPOP module

population


```
>>> from simuPOP import *
>>> simu = simulator(
...     population(size=1000, ploidy=2, loci=[2]),
...     randomMating(),
...     rep = 3)
```

Create a `population` of 1000 `diploid` individuals, each having two `loci` on the first chromosome

simulator and mating scheme


```
>>> from simuPOP import *
>>> simu = simulator(
...     population(size=1000, ploidy=2, loci=[2]),
...     randomMating(),
...     rep = 3)
```

Create a `simulator` that has one replicate of this population, and a random mating scheme

Operators!


```
>>> from simuPOP import *
>>> simu = simulator(
...     population(size=1000, ploidy=2, loci=[2]),
...     randomMating(),
...     rep = 3)
>>> simu.evolve(
...     preOps = [initByValue([1,2,2,1])],
...     ops = [
...         recombinator(rate=0.1),
...         stat(LD=[0,1]),
...         pyEval(r"'%3d    ' % gen", rep=0, step=10),
...         pyEval(r"'%f    ' % LD[0][1]", step=10),
...         pyEval(r"'\n'", rep=REP_LAST, step=10)
...     ],
...     end=100
... )
```

`initByValue` is applied before evolution

Operators!


```
>>> from simuPOP import *
>>> simu = simulator(
...     population(size=1000, ploidy=2, loci=[2]),
...     randomMating(),
...     rep = 3)
>>> simu.evolve(
...     preOps = [initByValue([1,2,2,1])],
...     ops = [
...         recombinator(rate=0.1),
...         stat(LD=[0,1]),
...         pyEval(r"'%3d    ' % gen", rep=0, step=10),
...         pyEval(r"'%f    ' % LD[0][1]", step=10),
...         pyEval(r"'\n'", rep=REP_LAST, step=10)
...     ],
...     end=100
... )
```

`recombinator` is applied at every generation when an offspring is produced

Operators!


```
>>> from simuPOP import *
>>> simu = simulator(
...     population(size=1000, ploidy=2, loci=[2]),
...     randomMating(),
...     rep = 3)
>>> simu.evolve(
...     preOps = [initByValue([1,2,2,1])],
...     ops = [
...         recombinator(rate=0.1),
...         stat(LD=[0,1]),
...         pyEval(r"'%3d    ' % gen", rep=0, step=10),
...         pyEval(r"'%f    ' % LD[0][1]", step=10),
...         pyEval(r"'\n'", rep=REP_LAST, step=10)
...     ],
...     end=100
... )
```

`stat` is applied to the offspring generation at every generation

Operators!


```
>>> from simuPOP import *
>>> simu = simulator(
...     population(size=1000, ploidy=2, loci=[2]),
...     randomMating(),
...     rep = 3)
>>> simu.evolve(
...     preOps = [initByValue([1,2,2,1])],
...     ops = [
...         recombinator(rate=0.1),
...         stat(LD=[0,1]),
...         pyEval(r"'%3d    ' % gen", rep=0, step=10),
...         pyEval(r"'%f     ' % LD[0][1]", step=10),
...         pyEval(r"'\n'", rep=REP_LAST, step=10)
...     ],
...     end=100
... )
```

`pyEval` is applied every 10 generations

A simple example


```
>>> from simuPOP import *
>>> simu = simulator(
...     population(size=1000, ploidy=2, loci=[2]),
...     randomMating(),
...     rep = 3)
>>> simu.evolve(
...     preOps = [initByValue([1,2,2,1])],
...     ops = [
...         recombinator(rate=0.1),
...         stat(LD=[0,1]),
...         pyEval(r"'%3d    ' % gen", rep=0, step=10),
...         pyEval(r"'%f    ' % LD[0][1]", step=10),
...         pyEval(r"'\n'", rep=REP_LAST, step=10)
...     ],
...     end=100
... )
```

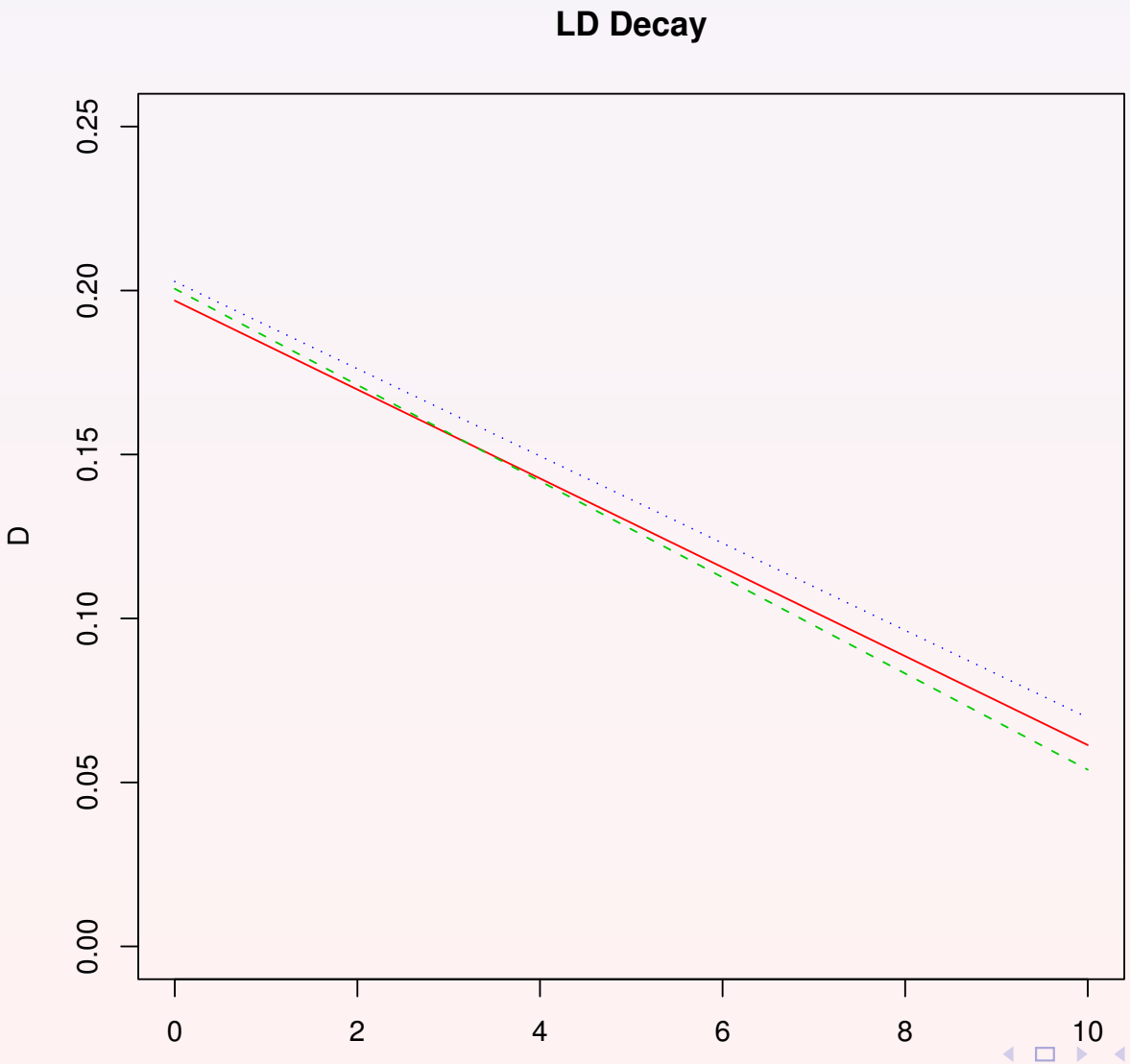
Output of the example


```
  0    0.197474    0.197991    0.202645
 10    0.074227    0.057794    0.068985
 20    0.022060    0.006820    0.032857
 30    0.022159    0.010874    0.012533
 40    0.006540    0.008600    0.007488
 50    0.011860    0.028355    0.002689
 60    0.000457    0.004552    0.008364
 70    0.006826    0.019827    0.006732
 80    0.031323    0.026524    0.014112
 90    0.015933    0.005628    0.003872
100    0.005808    0.001188    0.010402
```

Use R to plot

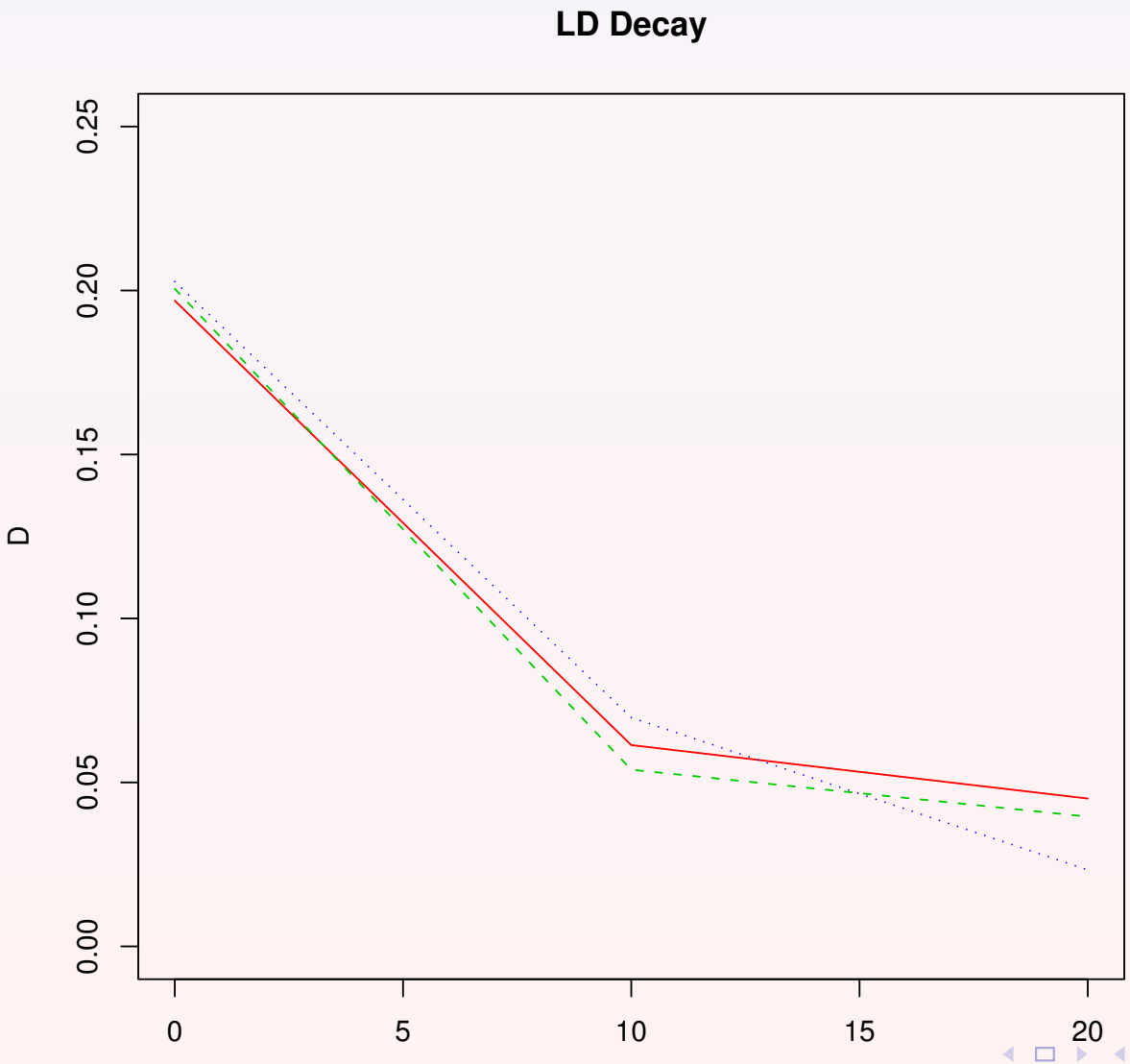

```
>>> from simuPOP import *
>>> from simuRPy import *
>>> simu = simulator(
...     population(size=1000, ploidy=2, loci=[2]),
...     randomMating(),
...     rep = 3)
>>> simu.evolve(
...     preOps = [initByValue([1,2,2,1])],
...     ops = [
...         recombinator(rate=0.1),
...         stat(LD=[0,1]),
...         varPlotter('LD[0][1]', numRep=3, step=10,
...             saveAs='ld', ylim=[0,.25],
...             lty=range(1, 4), col=range(2, 5),
...             xlab='generation', ylab='D',
...             title='LD Decay'),
...     ],
...     end=100
... )
True
>>>
```

Evolve!



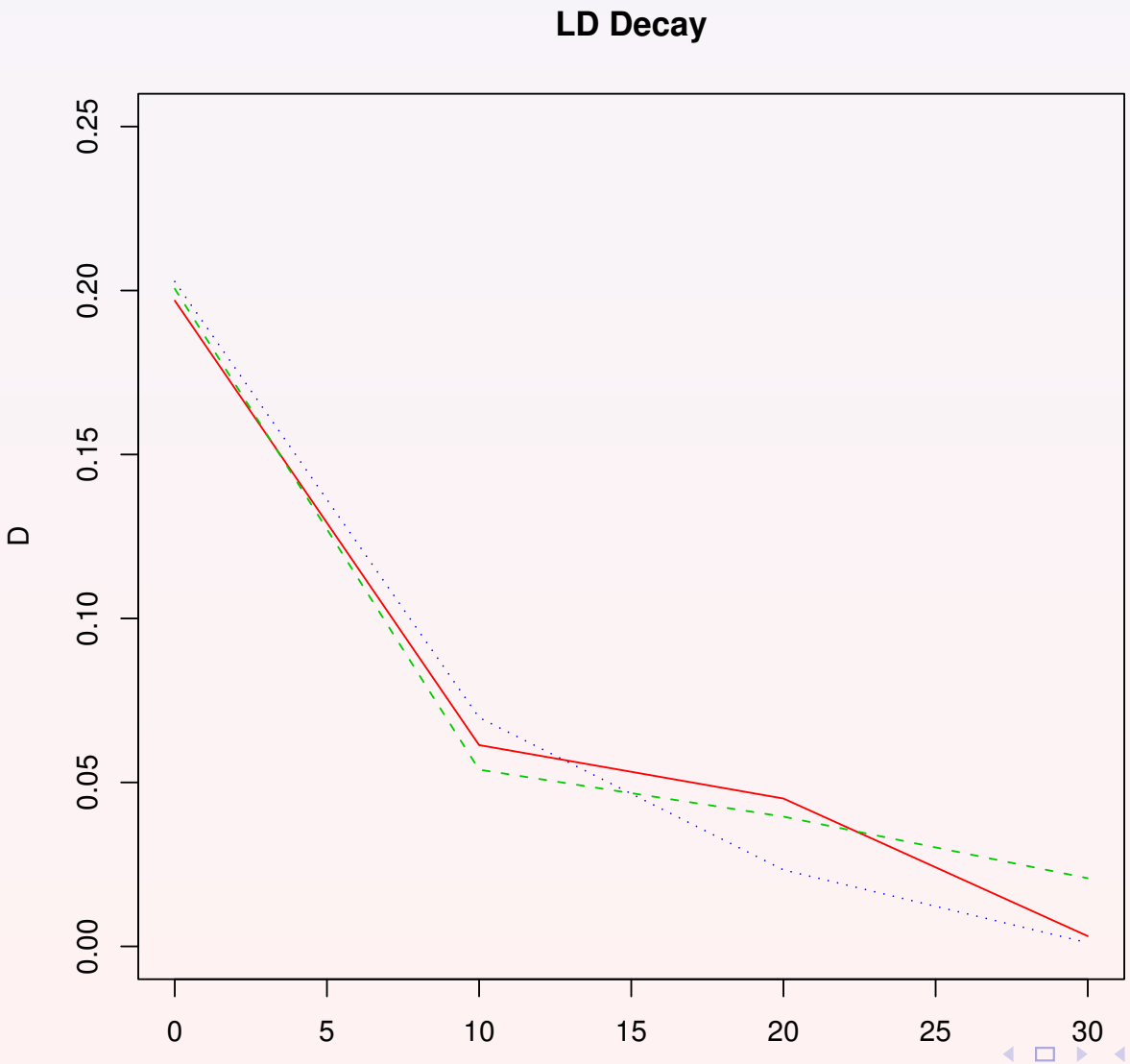
- Update at every 10 generations
- LD=0.25 before generation 0
- LD is calculated at the end of each generation

Evolve!



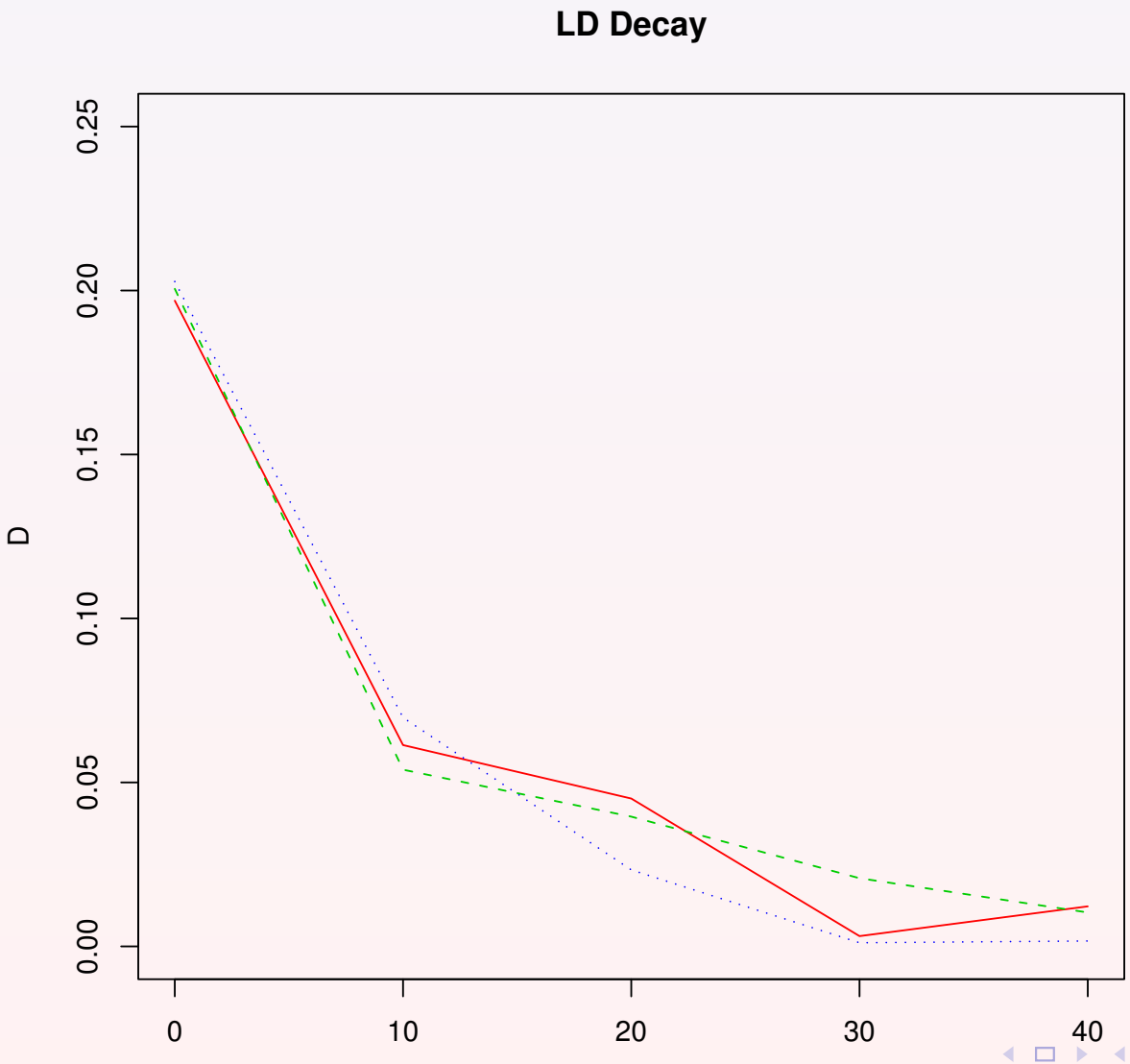
- Update at every 10 generations
- LD=0.25 before generation 0
- LD is calculated at the end of each generation

Evolve!



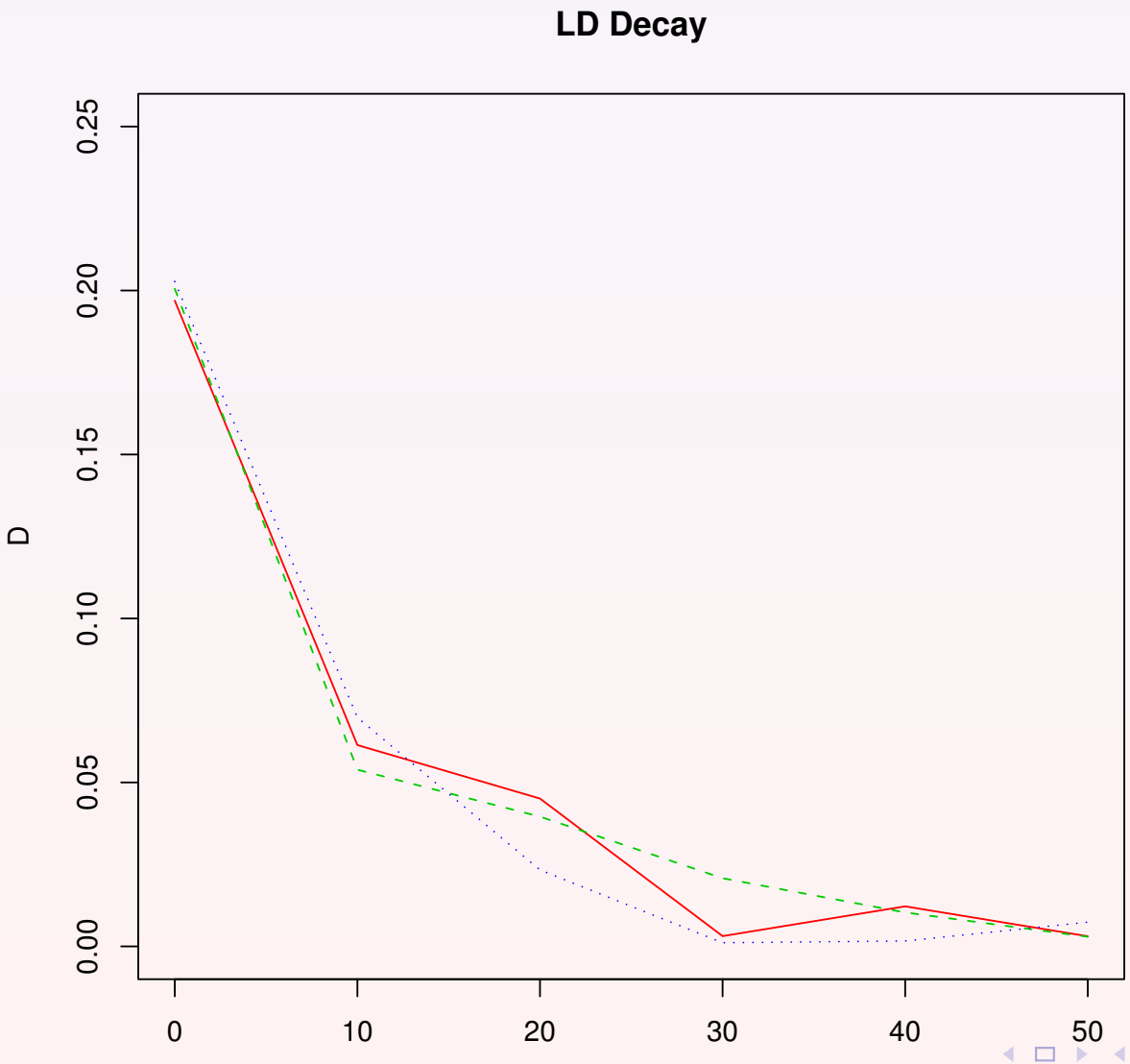
- Update at every 10 generations
- LD=0.25 before generation 0
- LD is calculated at the end of each generation

Evolve!



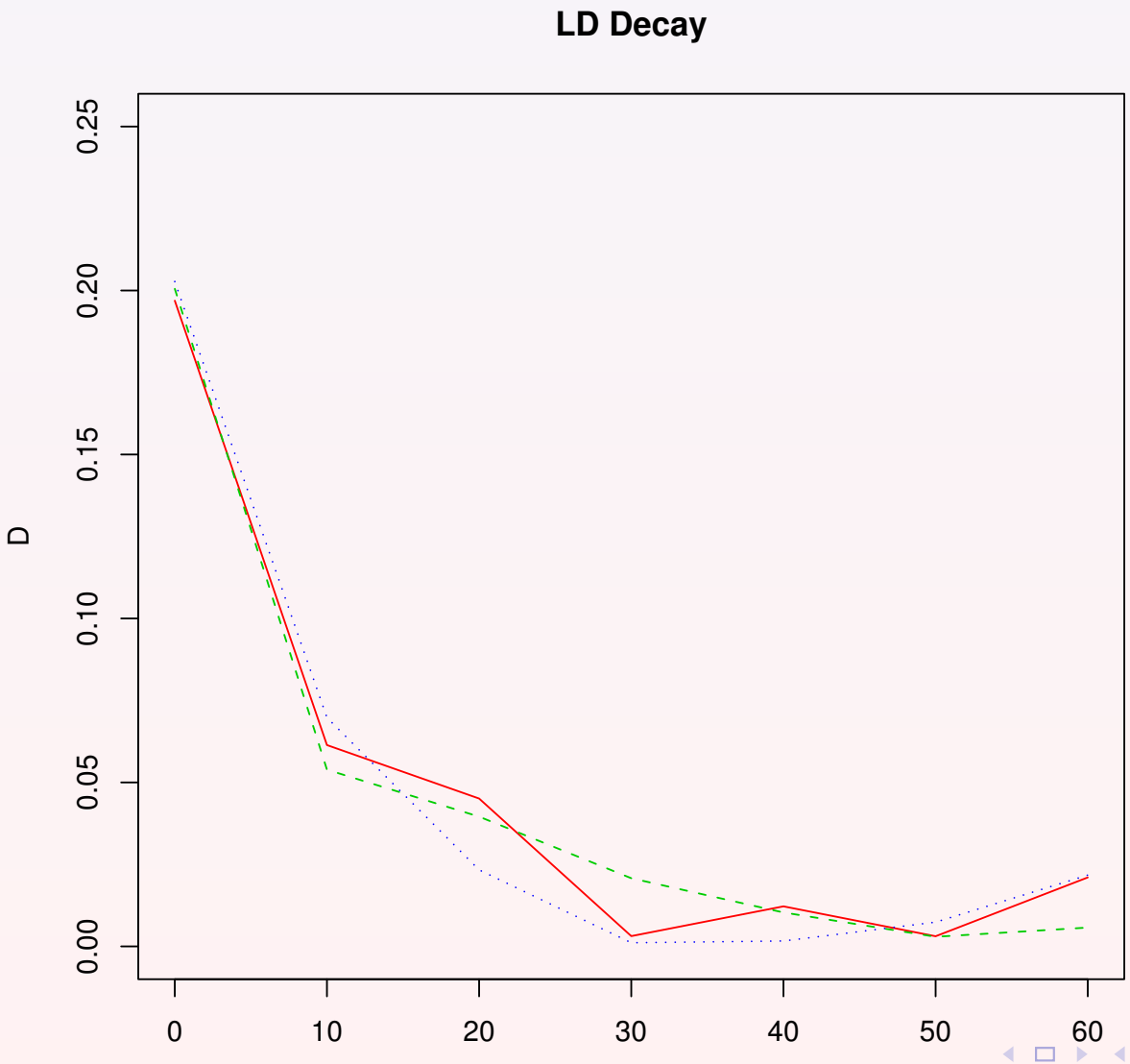
- Update at every 10 generations
- LD=0.25 before generation 0
- LD is calculated at the end of each generation

Evolve!



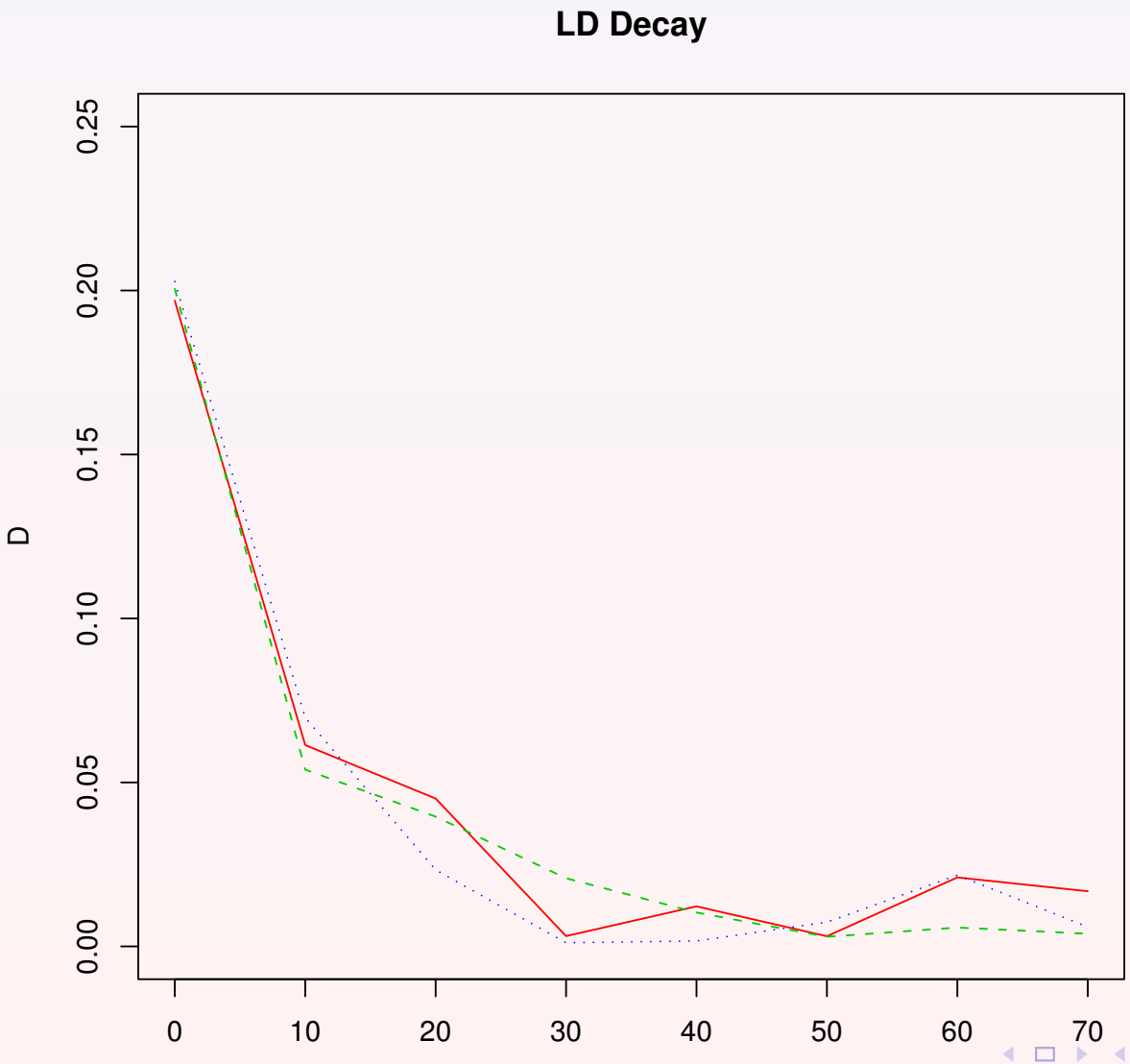
- Update at every 10 generations
- LD=0.25 before generation 0
- LD is calculated at the end of each generation

Evolve!



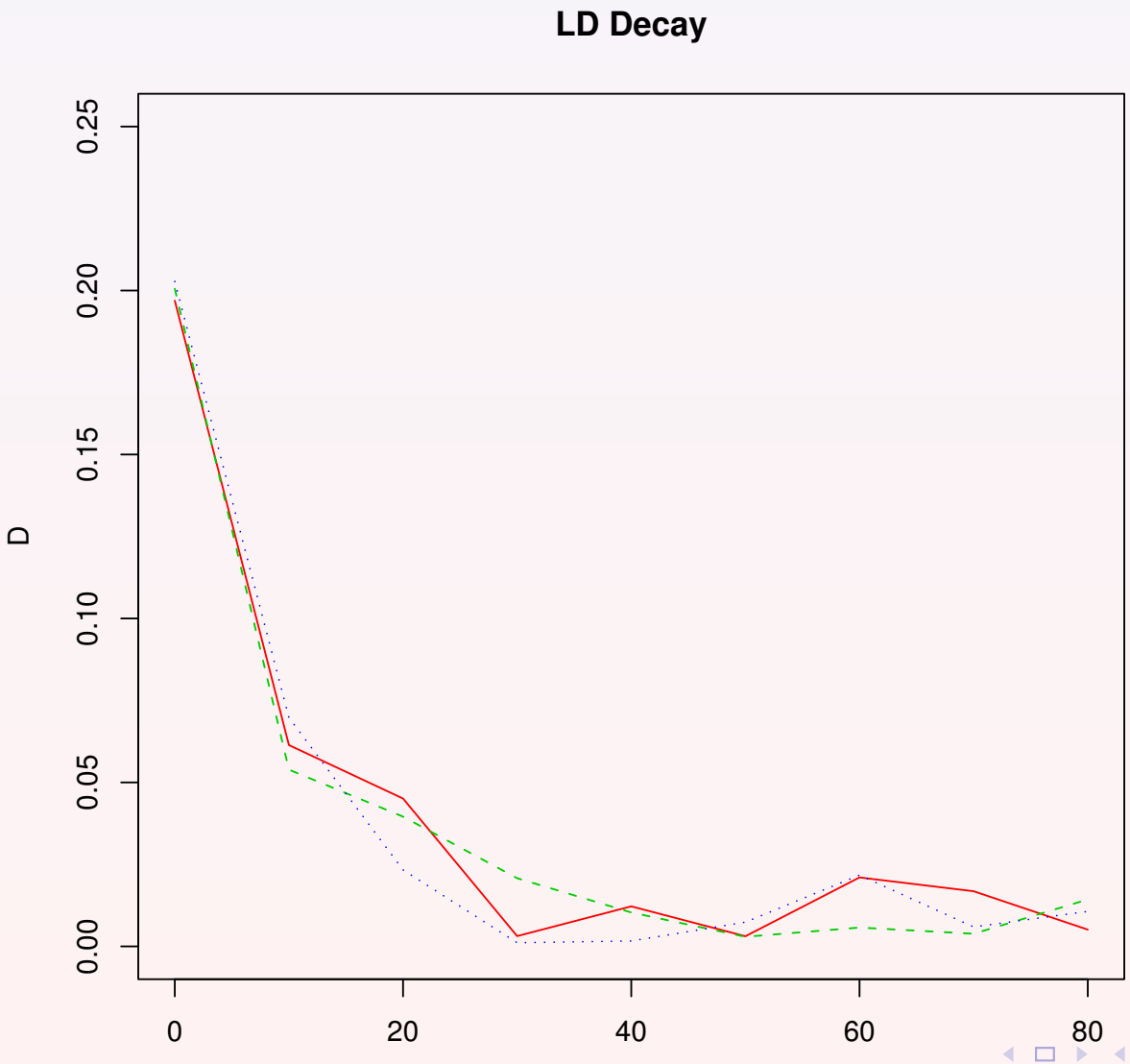
- Update at every 10 generations
- LD=0.25 before generation 0
- LD is calculated at the end of each generation

Evolve!



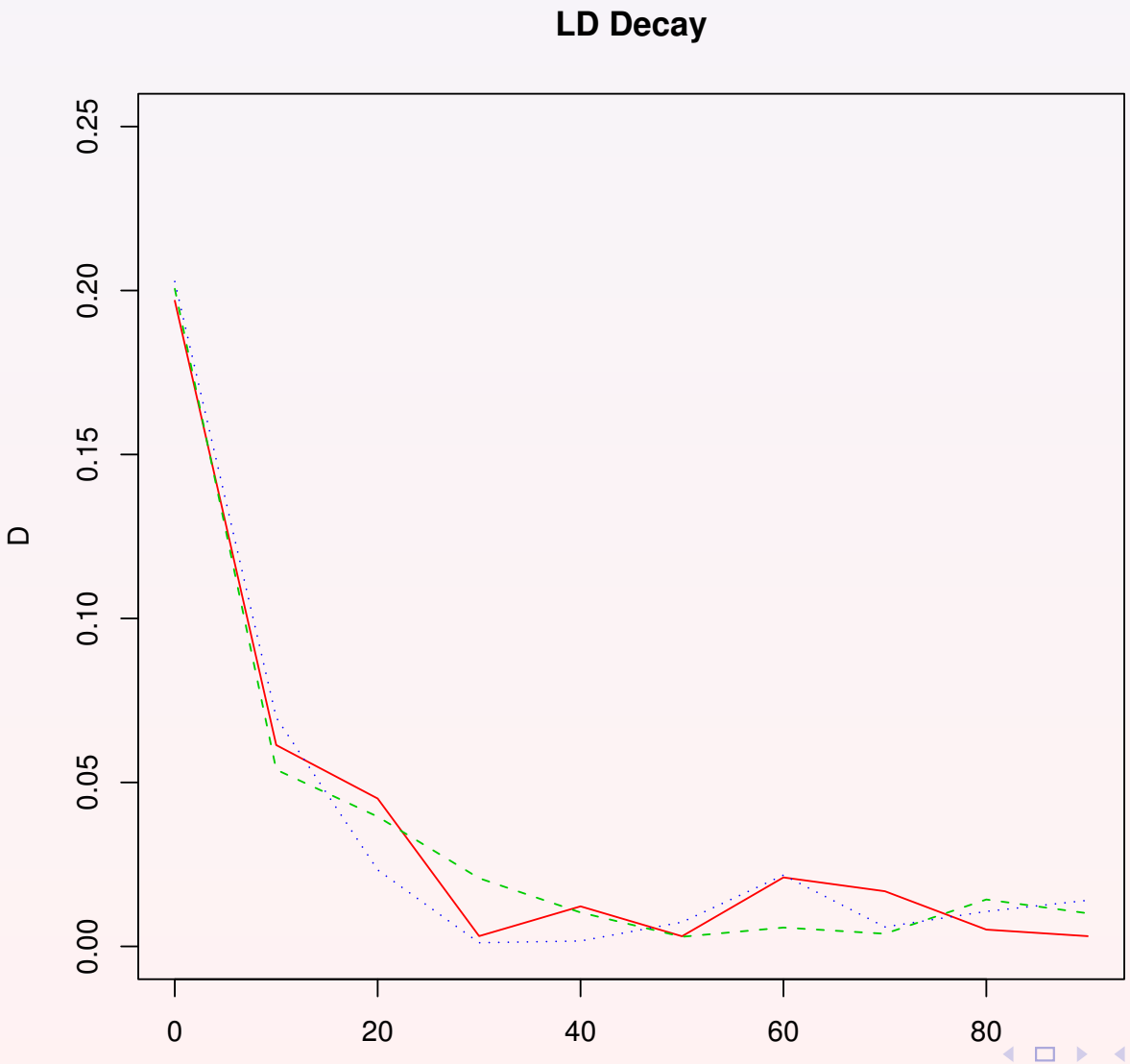
- Update at every 10 generations
- LD=0.25 before generation 0
- LD is calculated at the end of each generation

Evolve!



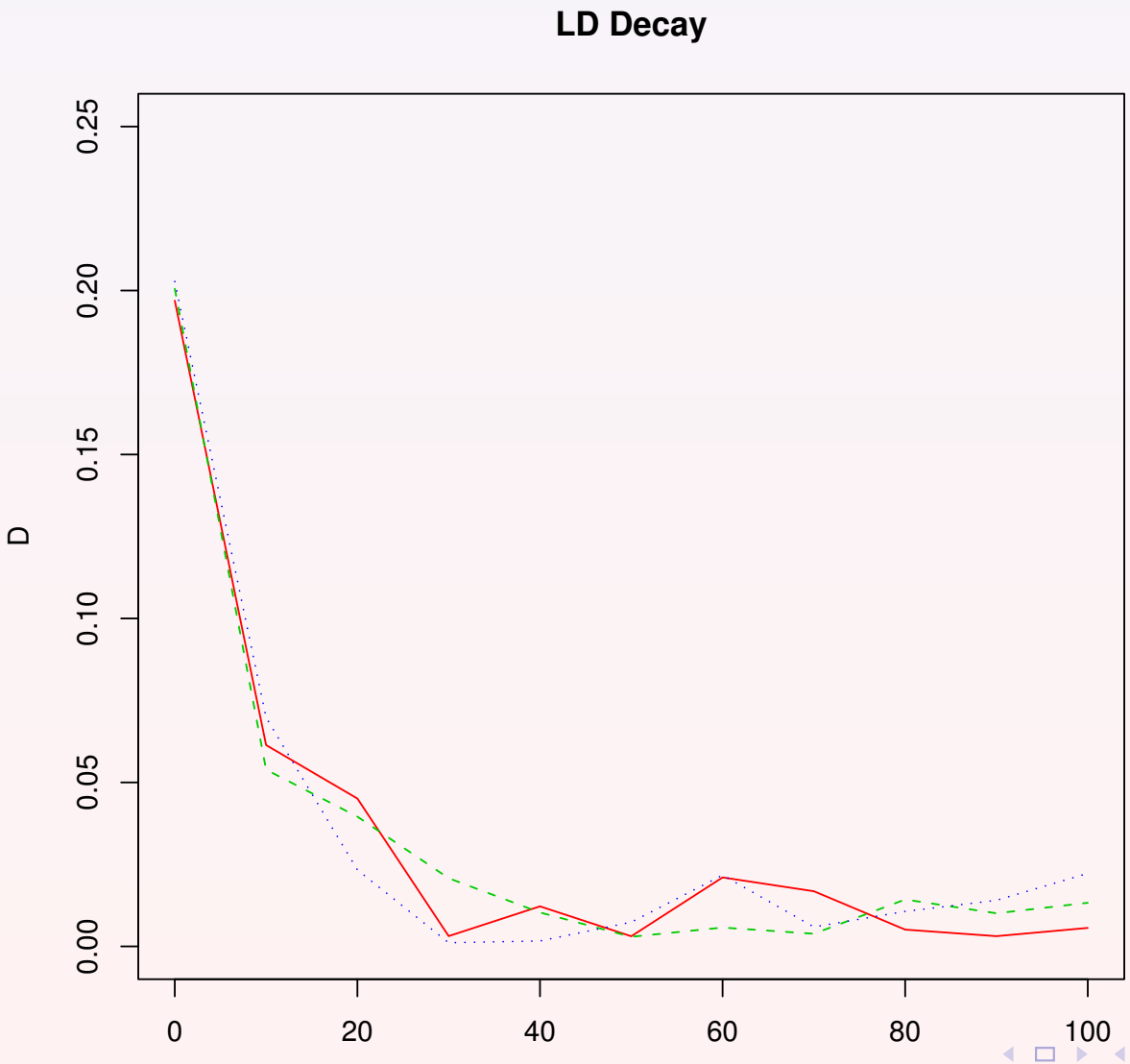
- Update at every 10 generations
- LD=0.25 before generation 0
- LD is calculated at the end of each generation

Evolve!



- Update at every 10 generations
- LD=0.25 before generation 0
- LD is calculated at the end of each generation

Evolve!



- Update at every 10 generations
- LD=0.25 before generation 0
- LD is calculated at the end of each generation

Exercise time

- Start python
- Load simuPOP
- Create a population and run

```
pop.ploidyName()
```

- run `tutorial_example1.py`

Outline

3 Various topics

- Dynamic population size
- Calculate statistics
- Hybrid Operator
- Self-defined statistics
- Read HapMap data
- Pick markers from HapMap data
- User interface

Dynamic population size


```
>>> def lin_inc(gen, oldsize=[]):
...     return [10+gen]*5
...
>>> simu = simulator(
...     population(subPop=[5]*5, loci=[1]),
...     randomMating(newSubPopSizeFunc=lin_inc)
... )
>>> simu.evolve(
...     ops = [
...         stat(popSize=True),
...         pyEval(r'"%d %d\n"%(gen, subPop[0]["popSize"])'),
...     ],
...     end=5
... )
0 10
1 11
2 12
3 13
4 14
5 15
True
>>>
```

Calculate statistics


```
>>> simu = simulator(
...     population(subPop=[10000]*2, loci=[10]),
...     randomMating()
... )
>>> simu.evolve(
...     preOps = [
...         initByFreq([0.2, 0.8], subPop=[0]),
...         initByFreq([0.8, 0.2], subPop=[1]),
...     ],
...     ops = [
...         stat(LD=[[0,1], [5,6]], Fst=range(10), step=100),
...         migrator(rate=[[0, 0.01], [0, 0.02]]),
...         pyEval(r'"Gen: %4d LD: %.3f R2: %.3f Fst: %.3f\n"' \
...             ' % (gen, LD[0][1], R2[0][1], AvgFst)',
...             step=100)
...     ],
...     end=1000
... )
```

Calculate statistics (cont.)


```
Gen:    0 LD: 0.088 R2: 0.123 Fst: 0.520
Gen:  100 LD: 0.065 R2: 0.067 Fst: 0.214
Gen:  200 LD: 0.045 R2: 0.032 Fst: 0.158
Gen:  300 LD: 0.035 R2: 0.020 Fst: 0.154
Gen:  400 LD: 0.045 R2: 0.033 Fst: 0.123
Gen:  500 LD: 0.052 R2: 0.044 Fst: 0.241
Gen:  600 LD: 0.049 R2: 0.039 Fst: 0.334
Gen:  700 LD: 0.052 R2: 0.044 Fst: 0.326
Gen:  800 LD: 0.037 R2: 0.022 Fst: 0.307
Gen:  900 LD: 0.028 R2: 0.013 Fst: -0.133
Gen: 1000 LD: 0.022 R2: 0.009 Fst: 0.000
True
>>>
```

A penetrance model

A penetrance model with two interating loci

	BB	Bb	bb
AA	0.1	0.1	0.5
Aa	0.1	0.1	0.5
aa	0.5	0.5	0.1

```
>>> def myPene(geno):
...     'geno is the genotype at the two given loci'
...     loc1 = geno[0] + geno[1]
...     loc2 = geno[2] + geno[3]
...     if (loc1 == 2 and loc2 < 2) or \
...         (loc1 < 2 and loc2 == 2):
...         return 0.1
...     else:
...         return 0.5
```

Apply this model


```
>>> pop = population(subPop=[1000], loci=[6])
>>> # initialize the population
>>> InitByFreq(pop, [0.1, 0.9])
>>> # apply penetrance and obtain affection status
>>> PyPenetrance(pop, loci=[3, 5], func=myPene)
>>> # draw case control sample
>>> (sample,) = CaseControlSample(pop, cases=3, controls=3)
>>> # save sample in Merlin QTDT format
>>> from simuUtil import SaveQTDT
>>> SaveQTDT(sample, output='sample', affectionCode=['U', 'A'],
...     fields=['affection'])
>>> # have a look at the sample in Merlin-QTDT Format
>>> print open('sample.map').read()
CHROMOSOME MARKER POSITION
1       loc1-1  1.000000
1       loc1-2  2.000000
1       loc1-3  3.000000
1       loc1-4  4.000000
1       loc1-5  5.000000
1       loc1-6  6.000000
```

Generated sample


```
>>> print open('sample.dat').read()
A       affection
M       loc1-1
M       loc1-2
M       loc1-3
M       loc1-4
M       loc1-5
M       loc1-6

>>> print open('sample.ped').read()
1 1 0 0 2 A 1 2 2 2 1 2 2 2 2 2 2 2
2 1 0 0 2 A 2 2 2 2 2 2 2 2 1 2 2 2
3 1 0 0 2 A 2 2 2 2 2 2 2 2 2 1 2 2
4 1 0 0 2 U 2 2 2 2 1 2 2 1 1 2 2 2
5 1 0 0 1 U 2 2 2 2 2 2 2 2 2 2 2 1
6 1 0 0 1 U 2 2 2 2 2 2 2 2 2 2 2 2

>>>
```

Calculate effective number of alleles

The effective number of alleles can be estimated from a population by

$$\hat{n}_e = \left(\sum_{i>0} \left(\frac{f_i}{f_0} \right)^2 \right)^{-1} = \frac{f_0^2}{\sum_{i>0} f_i^2}$$

where f_i is the frequency of allele i, and $f_0 = \sum_{i>0} f_i$ is the total disease allele frequency (assuming 0 is the only wildtype alelle).

```
>>> def Ne(pop, loci):
...     'Calculate effective number of alleles'
...     Stat(pop, alleleFreq=loci)
...     pop.dvars().Ne = {}
...     v = pop.dvars().alleleFreq
...     for locus in loci:
...         f0 = 1 - v[locus][0]
...         Ne = f0*f0/sum([x*x for x in v[locus][1:]])
...         pop.dvars().Ne[locus] = Ne
...     return True
```

Use a Python operator


```
>>> simu = simulator(
...     population(1000, loci=[1], infoFields=['fitness']),
...     randomMating())
>>> simu.evolve(
...     preOps = [ initByFreq([0.1]*10) ],
...     ops = [
...         maSelector(locus=0, fitness=[1, 0.999, 0.998]),
...         pyOperator(func=Ne, param=[0], step=100),
...         pyEval(r'"Ne=%.3f\n" % Ne[0]', step=100),
...     ],
...     end=500
... )
Ne=8.961
Ne=5.416
Ne=5.104
Ne=4.146
Ne=2.693
Ne=2.185
True
>>>
>>>
```

scripts/loadHapMap.py

Load genotype from hapmap data file

```
def load_population(pop, ch, type):
    '''Load population from file, with type (subpopulation type)
    subPop = {'CEU':0, 'YRI':1, 'JPT+CHB':2}[type]
    file = genotype_file % (ch, type, rev)
    print 'from %s...' % file
    for line_no,line in enumerate(open(file).readlines()):
        genotype = [int(x) for x in line.split()]
        ind = line_no / 2
        ploidy = line_no % 2
        ind = pop.individual(ind, subPop)
        for i,g in enumerate(genotype):
            # always chromosome 0, because each population has o
            ind.setAllele(g, i, ploidy)
```

Pick markers from HapMap data


```
>>> genes = [
...     "rs1042522",
...     "rs1625895",
...     "rs1799793",
... ]
>>> pops = []
>>> for i in range(1, 23):
...     print "Loading hapmap chromosome %d..." % i
...     pop = LoadPopulation('hapmap_%d.bin' % i)
...     markers = []
...     for name in genes:
...         try:
...             idx = pop.locusByName(name)
...             markers.append(idx)
...         except:
...             pass
...     if len(markers) > 0:
...         markers.sort()
...         pop.removeLoci(keep=markers)
...         pops.append(pop)
>>> all = MergePopulationsByLoci(pops)
```

Use of simuOpt.py


```
options = [
    {'arg':'h',
     'longarg':'help',
     'default':False,
     'description':'Print this usage message.',
     'jump':-1
    },
    {'arg':'s:',
     'longarg':'size=',
     'default':1000,
     'label':'Population Size',
     'allowedTypes':[types.IntType, types.LongType],
     'validate':simuOpt.valueGT(0),
     'description':'Population size'
    },
    {'arg':'r:',
     'longarg':'recRate=',
     'default':0.01,
     'label':'Recombination Rate',
     'allowedTypes':[types.FloatType],
     'description':'Recombination rate',
     'validate':simuOpt.valueBetween(0.,1.),
    },
]
```

Input methods


```
# get all parameters
allParam = simuOpt.getParam(options, __doc__)

if len(allParam) > 0:    # successfully get the params
    (help, popSize, endGen, recRate, numRep, saveFigure,
    saveConfig, method, verbose) = allParam
else:
    sys.exit(0)

if saveConfig != '':
    simuOpt.saveConfig(options, saveConfig, allParam)

if help:
    print simuOpt.usage(options, __doc__)
    sys.exit(1)
```

Outline

4 Bundled Scripts

- simuLDDecay.py
- simuNeutralSNPs.py
- simuForward.py
- simuComplexDisease.py
- simuCluster.py

simuLDDecay.py

- simulate the decay of linkage disequilibrium with recombination
- can control population size, recombination rate, number of replicates and generations
- use `simuRPy.py` to visualize the decay of LD

simuNeutralSNPs.py

- simulate the evolution of unlinked SNP markers
- observe the distribution of minor allele frequency
- no selection

simuForward.py

- Traditional forward-time simulation
- Use a dynamic-selector to control disease allele frequency in a disease introduction stage
- Restart simulation when a disease allele get lost

simuComplexDisease.py

- New forward-time simulation method (Peng, 2007)
- Simulate the trajectory of disease allele frequencies backward in time
- Controlled forward-time simulation method that follows simulated disease allele frequency

simuCluster.py

- A utility script to help running simuPOP scripts on a cluster system
- User provides a template scripts and a list of paramters
- This script generate scripts and submit the jobs

Acknowledgments

- W.M. Keck Foundation to the Gulf Coast Consortia through the Keck Center for Computational and Structural Biology
- Rice Terascale Cluster funded by NSF under Grant EIA-0216467, Intel, and HP
- M.D. Anderson Cancer Center High Performance Cluster
- BP was supported in part by a grant CA75432 from NCI
- Yaji Xu helped with all the figures in this talk

For further reading

- **Bo Peng** and Marek Kimmel (2005). simuPOP: a forward-time population genetics simulation environment. *Bioinformatics,* 21:3686–3687
- **Bo Peng** and Marek Kimmel (2007) Simulations provide support for the common disease common variant hypothesis. *Genetics.* 175:763-776.
- **Bo Peng**, Christopher I. Amos and Marek Kimmel (2007) Forward-time simulations of complex human diseases. *PLoS Genetics,* 3(3):e47.

That is all

For more details, please check out

- simuPOP user's guide
- simuPOP reference manual
- Another presentation about the details of each simuPOP components

Under the doc directory of your simuPOP distribution.