Forward-time simulations using simuPOP, a tutorial

Bo Peng

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

June. 6, 2007
Programmers' Cross Training
U.T. M.D. Anderson Cancer Center

outline

Why population genetics simulations are needed

- High cost of genetic data collection
  Simulations are relatively inexpensive

Why population genetics simulations are needed

- High cost of genetic data collection
  Simulations are relatively inexpensive
- Inaccessibility to ancestral information
  Simulations can simulate very long evolutionary process

Why population genetics simulations are needed

- High cost of genetic data collection
  Simulations are relatively inexpensive
- Inaccessibility to ancestral information
  Simulations can simulate very long evolutionary process
- Complexity of real-world genetic effect
  Simulations can control the number and genetic effects of disease susceptibility loci (DSL)

Two major simulation methods

Backward-time

- 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

Two major simulation methods

Backward-time

- 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-time

- 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

Comparison between forward- and backward-time simulations

Backward-time

- Sample based, efficient.

Forward-time

- Population based, inefficient.

Comparison between forward- and 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

Comparison between forward- and 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

Comparison between forward- and 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
- Wider application area

On the simulations of complex human diseases

Backward-time

- Haploid simulation only

Forward-time

- No limit on ploidy

On the simulations of complex human diseases

Backward-time

- Haploid simulation only
- Additive selection and penetrance models

Forward-time

- No limit on ploidy
- Arbitrary selection and penetrance models

On the simulations of complex human diseases

Backward-time

- Haploid simulation 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

On the simulations of complex human diseases

Backward-time

- Haploid simulation only
- Additive selection and penetrance models
- One disease susceptibility locus
- Generate independent samples

Forward-time

- No limit on ploidy
- Arbitrary selection and penetrance models
- Multiple DSL with interaction
- Simulate populations, which allows more flexible sampling

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)

What simuPOP can do

simuPOP provides

- a large number of objects and functions to manipulate populations,
  copy, split, merge, save, load, modify genotype,
  determine affection status, generate sample, ...
- and a mechanism to evolve populations forward in time subject to all sorts of demographic and genetic forces such as population size changes, mutation, migration, selection...

Why is simuPOP needed?

How to use simuPOP

Just like R/S-Plus or Matlab, you can

- Interactively manipulate populations and evolve them
- Write a script (in Python)
- Run existing script

Partial feature list

- Scripting language...
- Binary, short and long allele types
- Arbitrary demographic changes
- 60+ operators supporting all sorts of

Oohm, Python??

The core of simuPOP is written in C++, and is provided (wrapped) as Python modules.

- Python is easy to learn
- Python is easy to write and maintain
- Python is

A few quick statistics

- 6 modules (short, long, binary) $\times$ (standard, optimized)

A few quick statistics

- 6 modules (short, long, binary) $\times$ (standard, optimized)
- 60+ operators

A few quick statistics

- 6 modules (short, long, binary) $\times$ (standard, optimized)
- 60+ operators
- ...

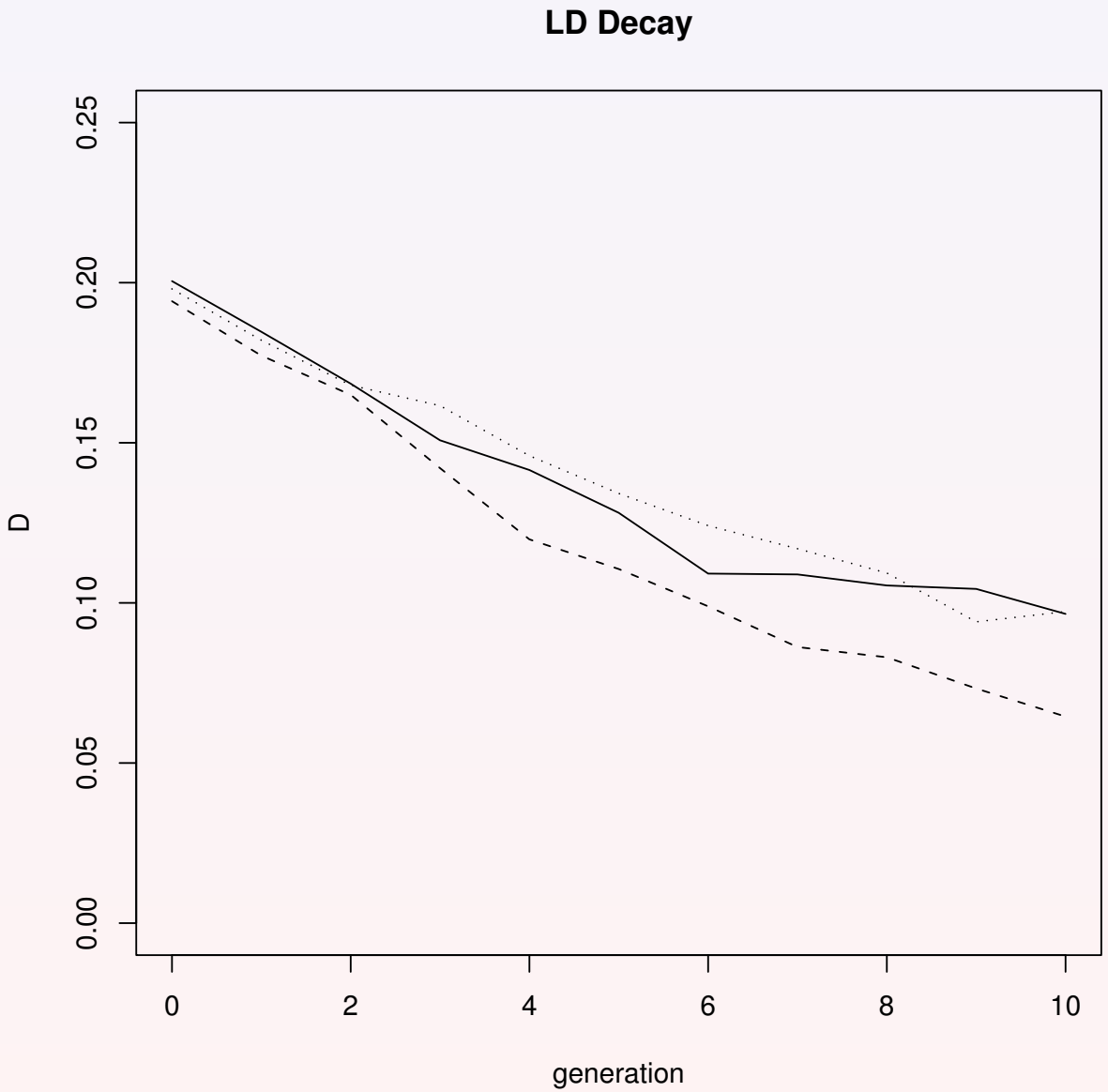
Outline

3 Forward-time population generation

- simuPOP - a forward-time simulation environment
- A simple example

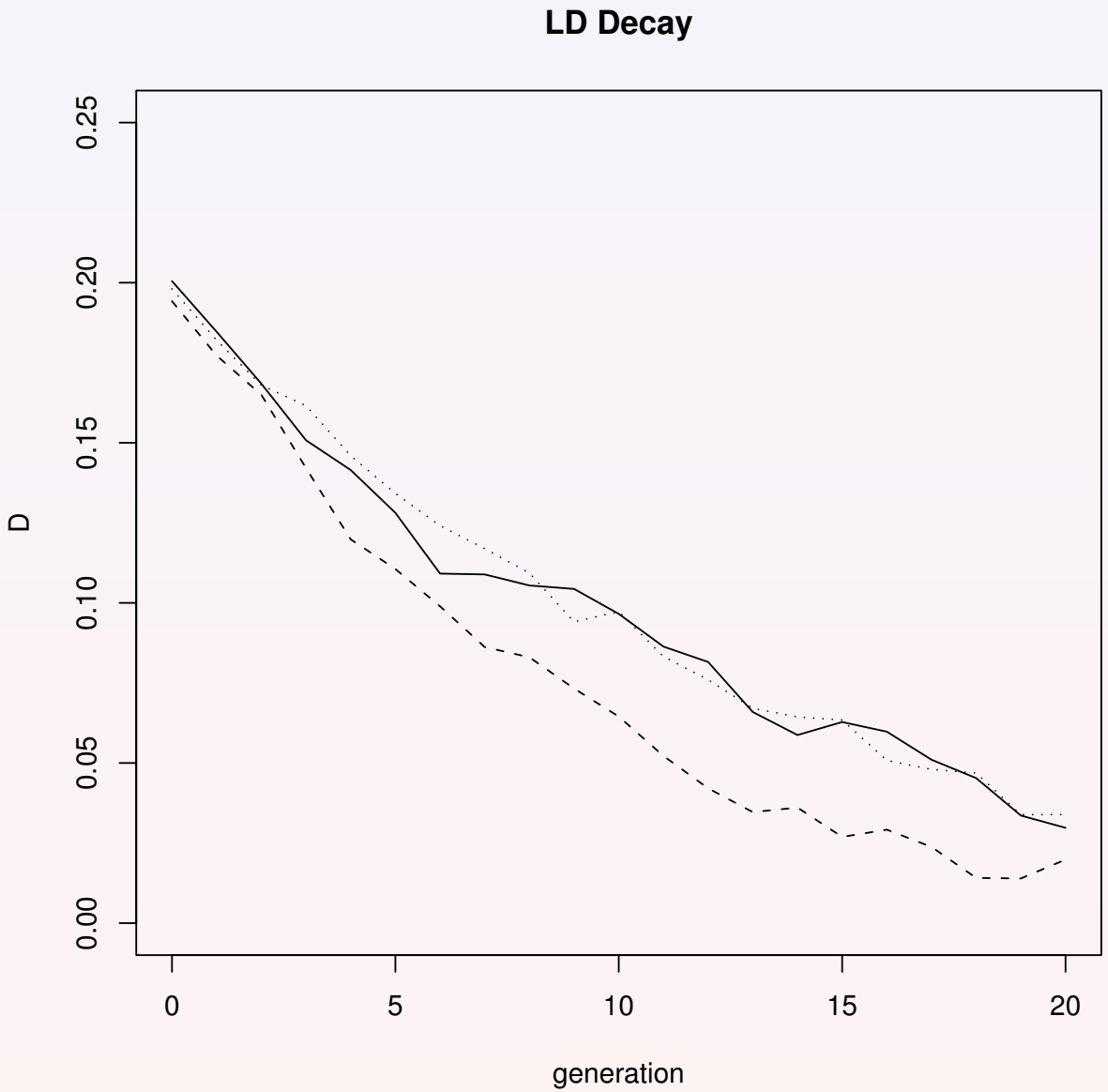
A simple example


```
>>> from simuPOP import *
>>> from simuRPy import *
>>> simu=simulator(
...   population(size=1000,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,
...       ylim=[0,.25],xlab="generation",
...       ylab="D",title="LD Decay")],
...   end=100 )
```



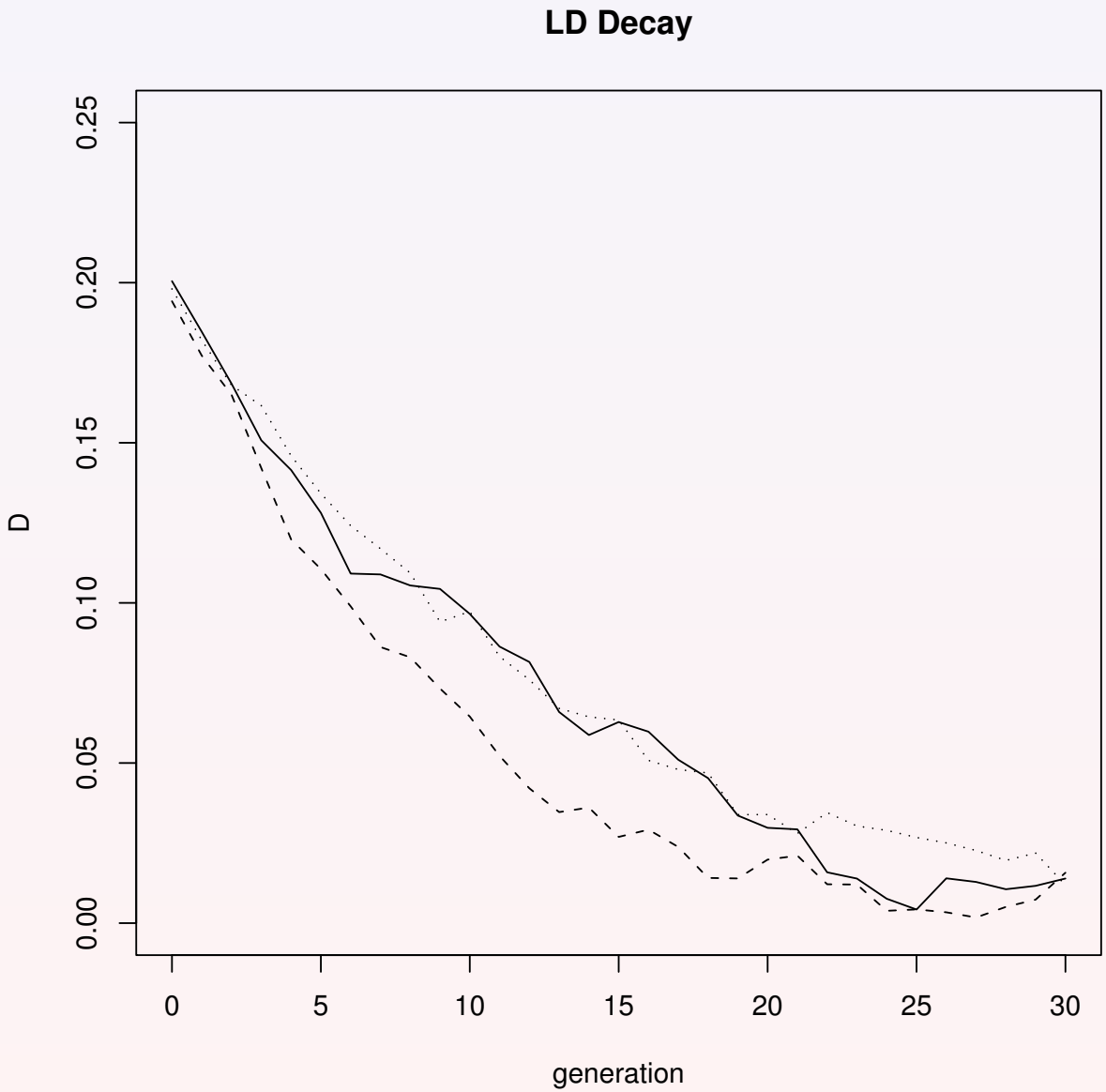
- Update at every 10 generations
- LD=0.25 before generation 0
- LD calculated at the end of each generation

Replay



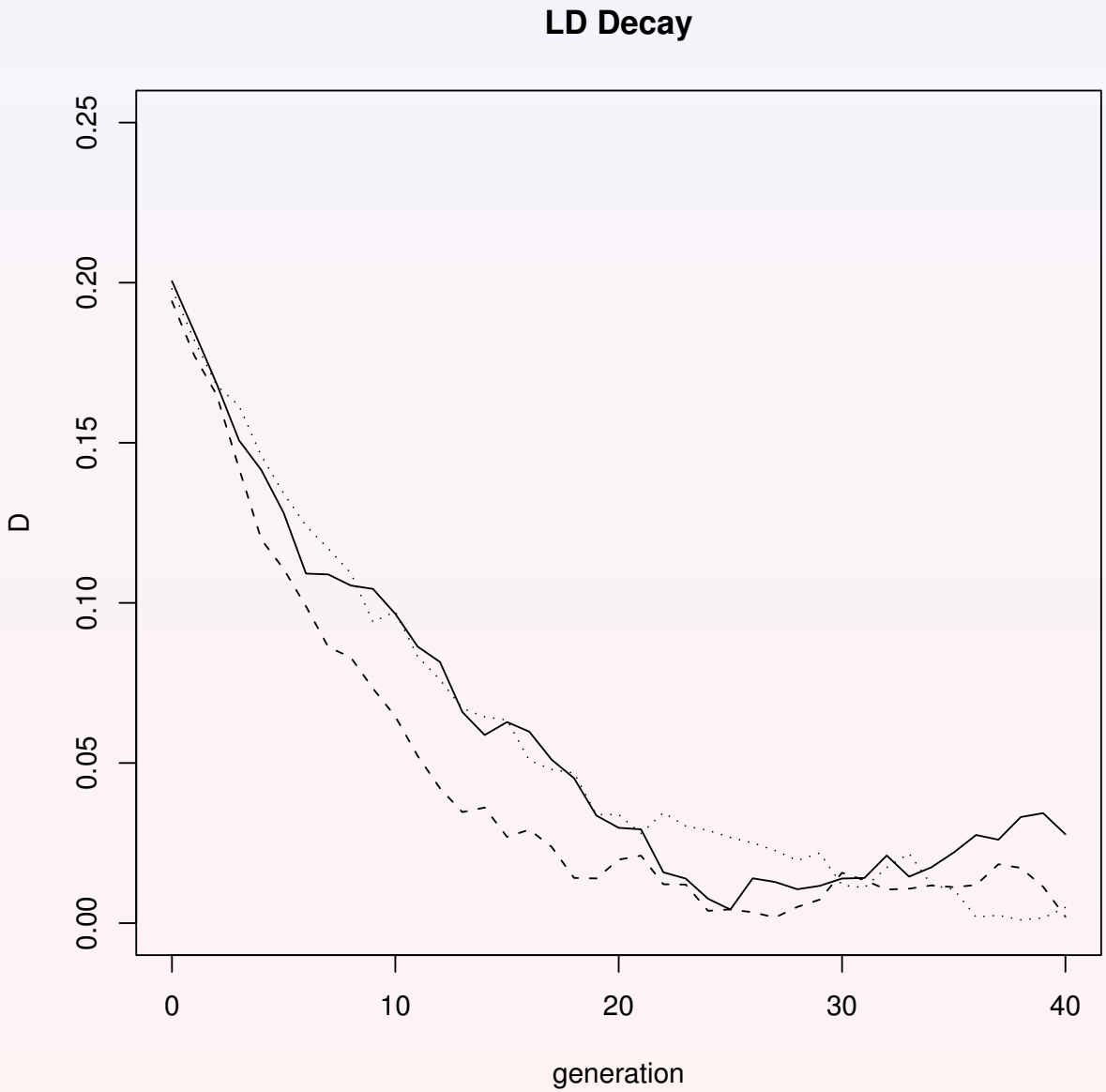
- Update at every 10 generations
- LD=0.25 before generation 0
- LD calculated at the end of each generation

Replay



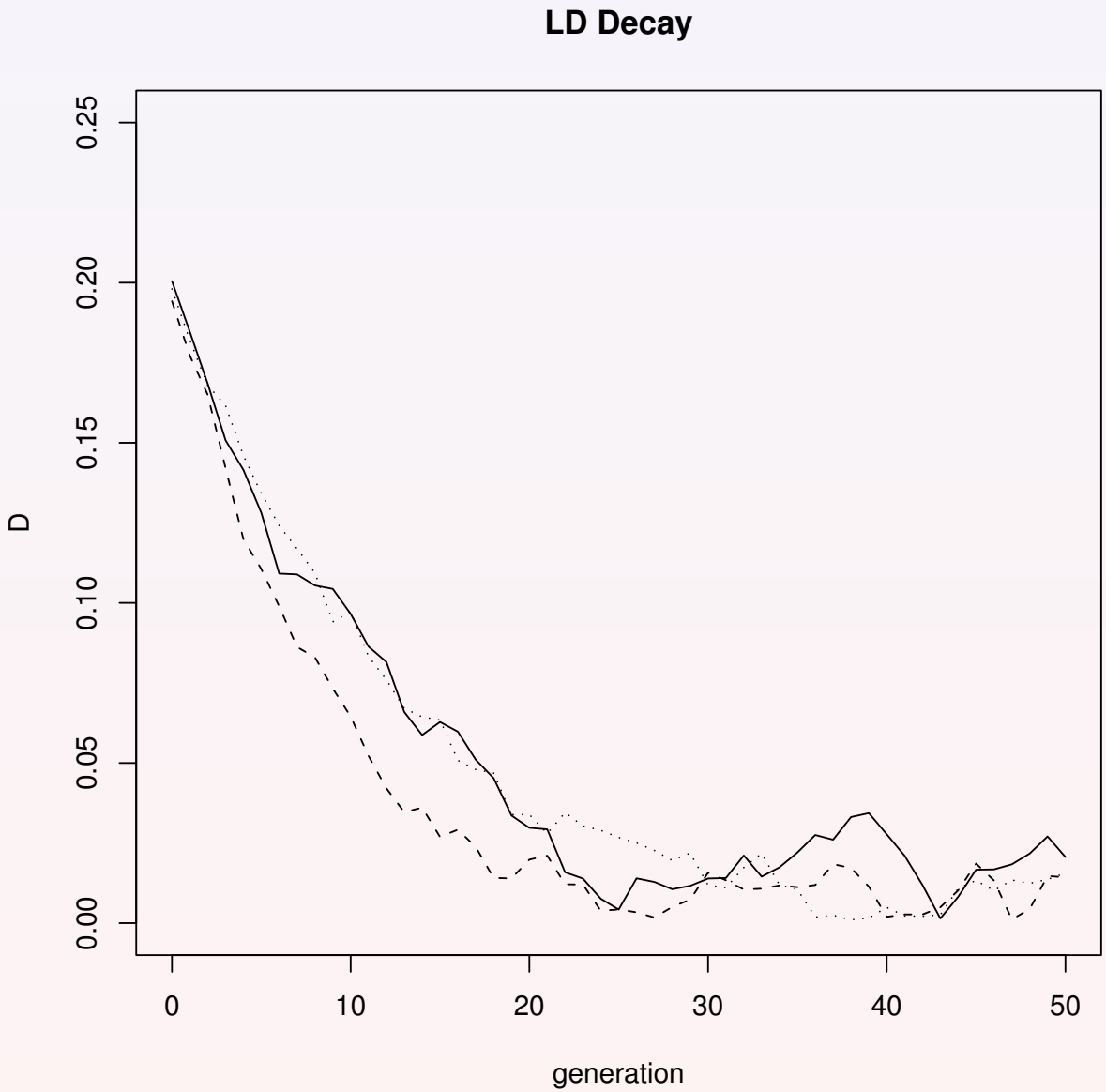
- Update at every 10 generations
- LD=0.25 before generation 0
- LD calculated at the end of each generation

Replay



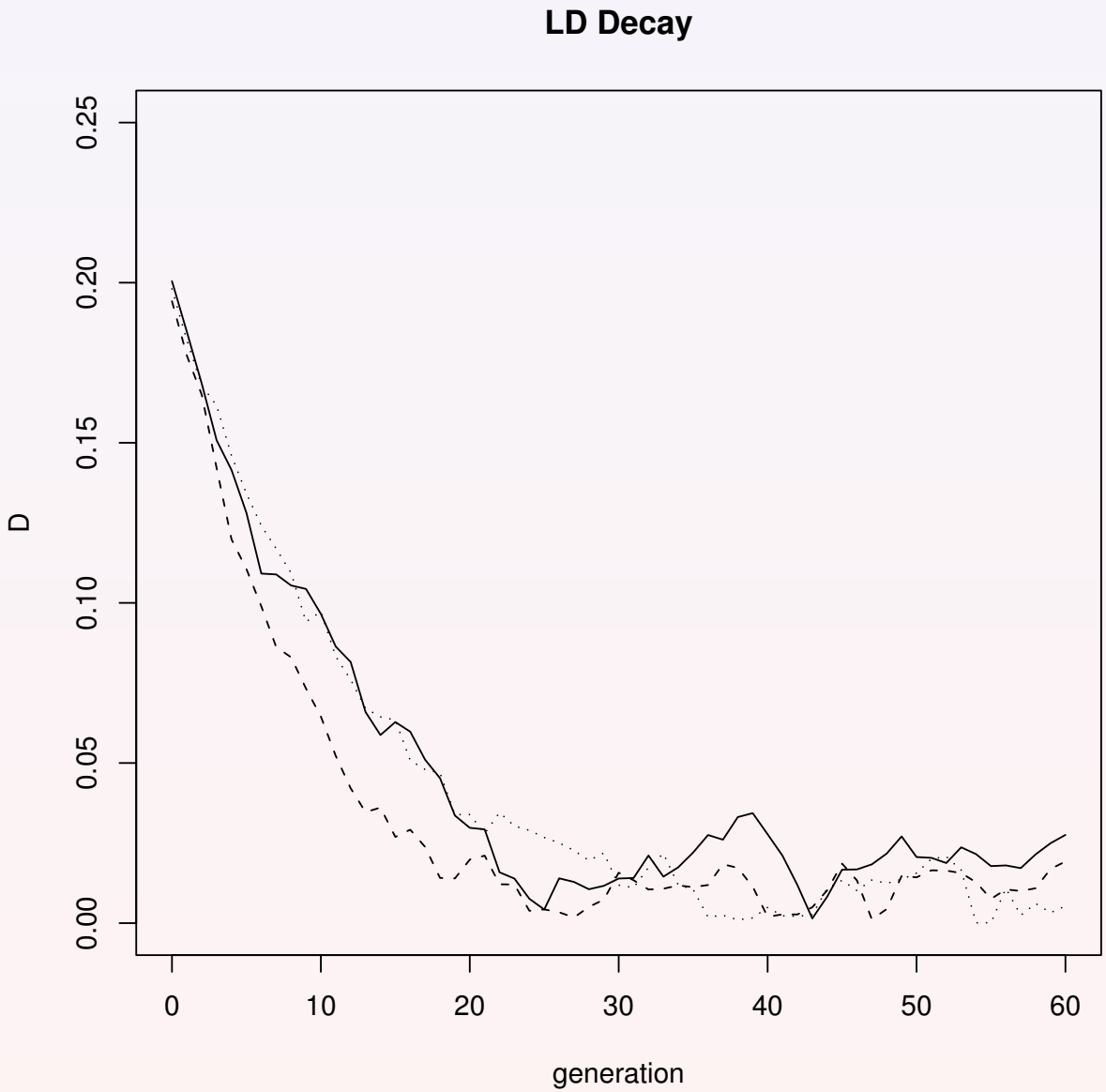
- Update at every 10 generations
- LD=0.25 before generation 0
- LD calculated at the end of each generation

Replay



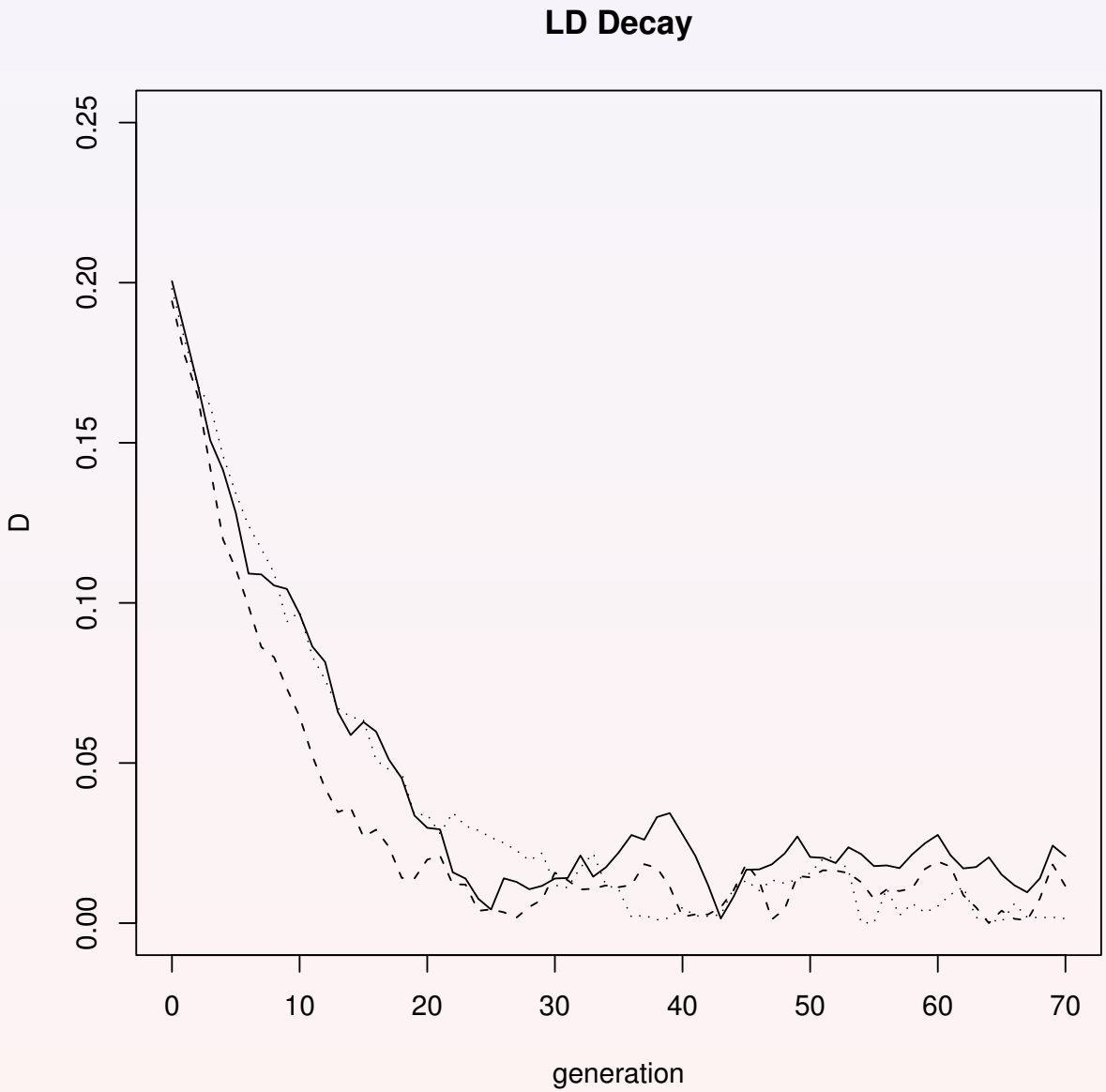
- Update at every 10 generations
- LD=0.25 before generation 0
- LD calculated at the end of each generation

Replay



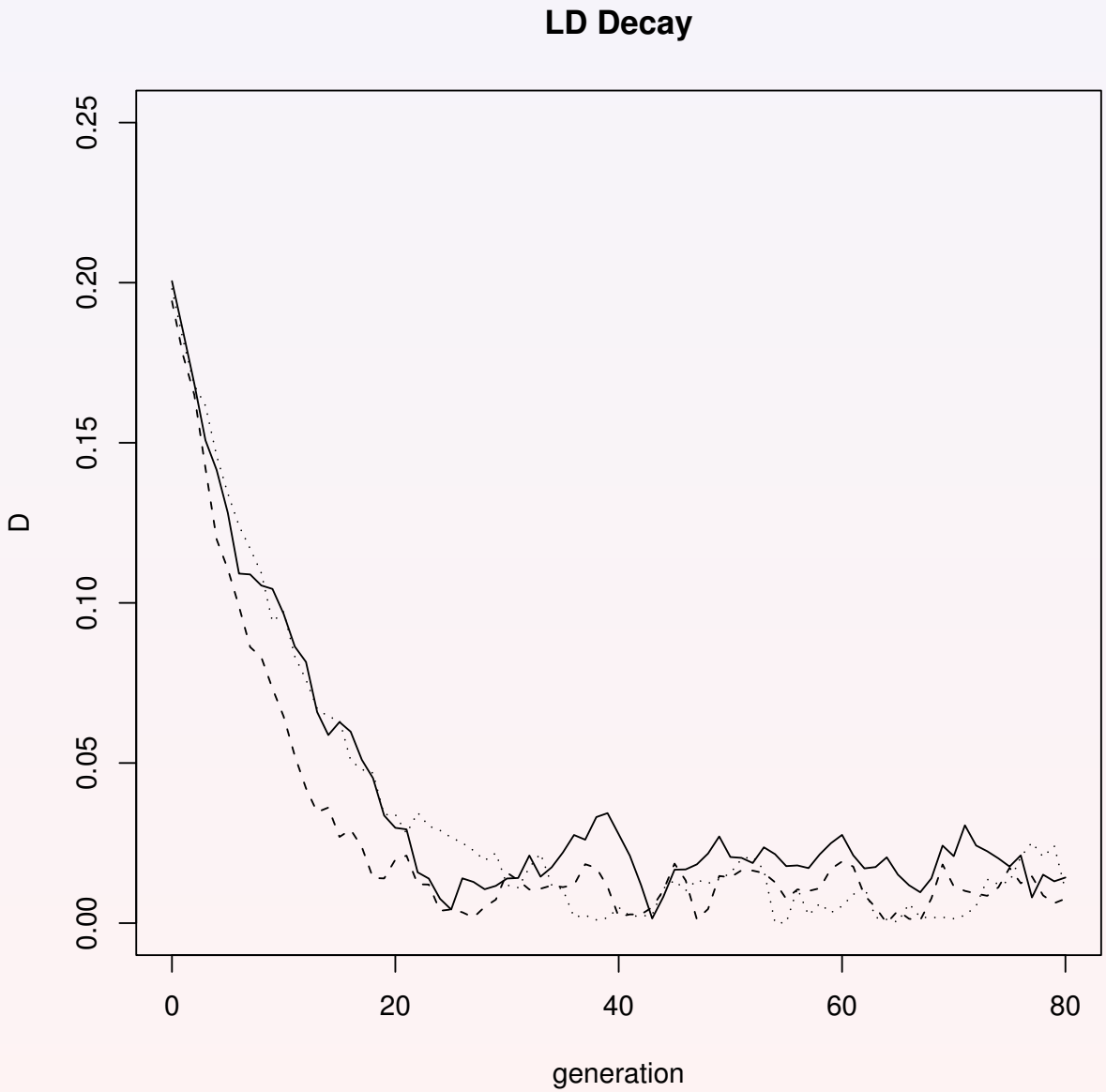
- Update at every 10 generations
- LD=0.25 before generation 0
- LD calculated at the end of each generation

Replay



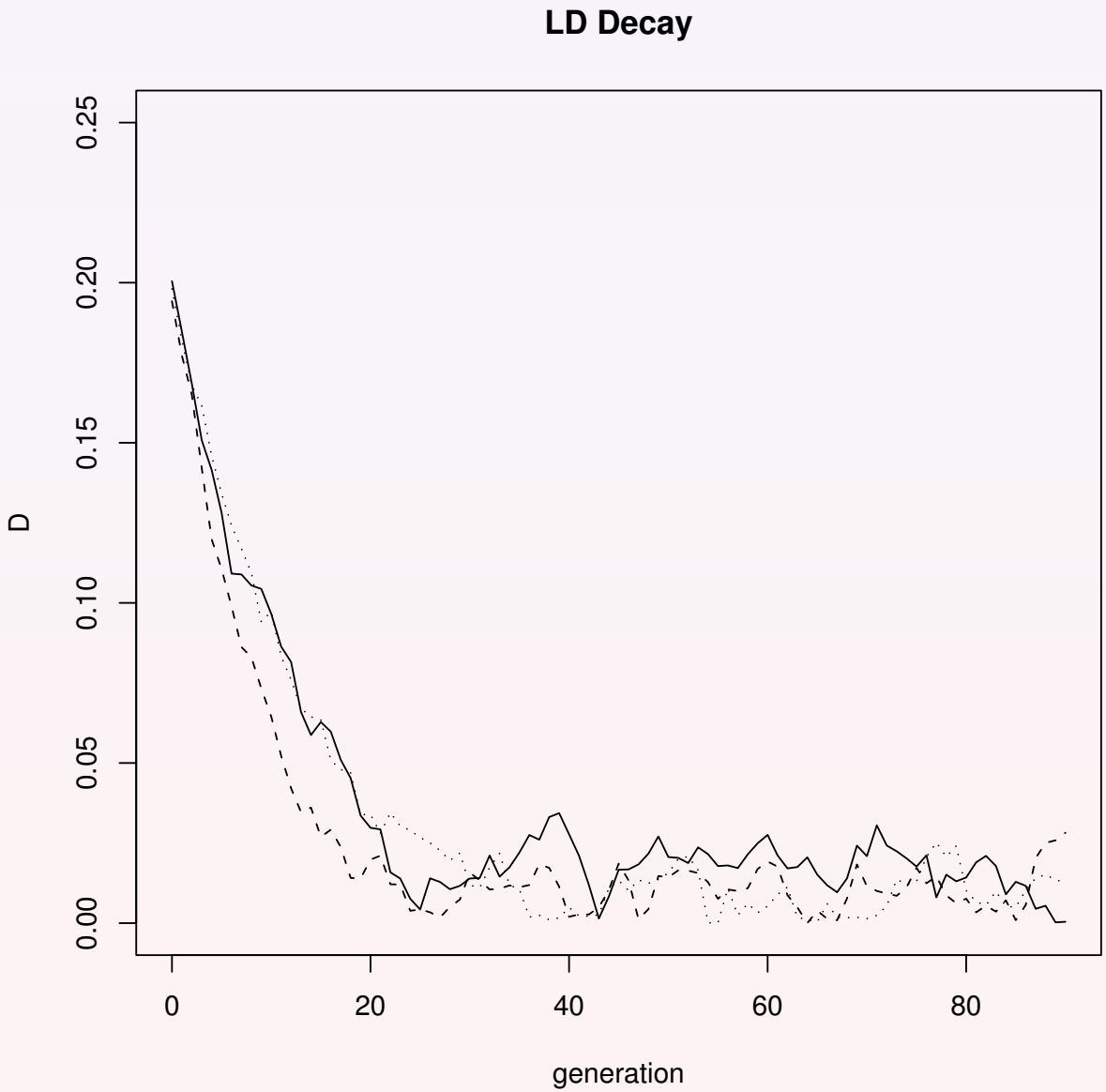
- Update at every 10 generations
- LD=0.25 before generation 0
- LD calculated at the end of each generation

Replay



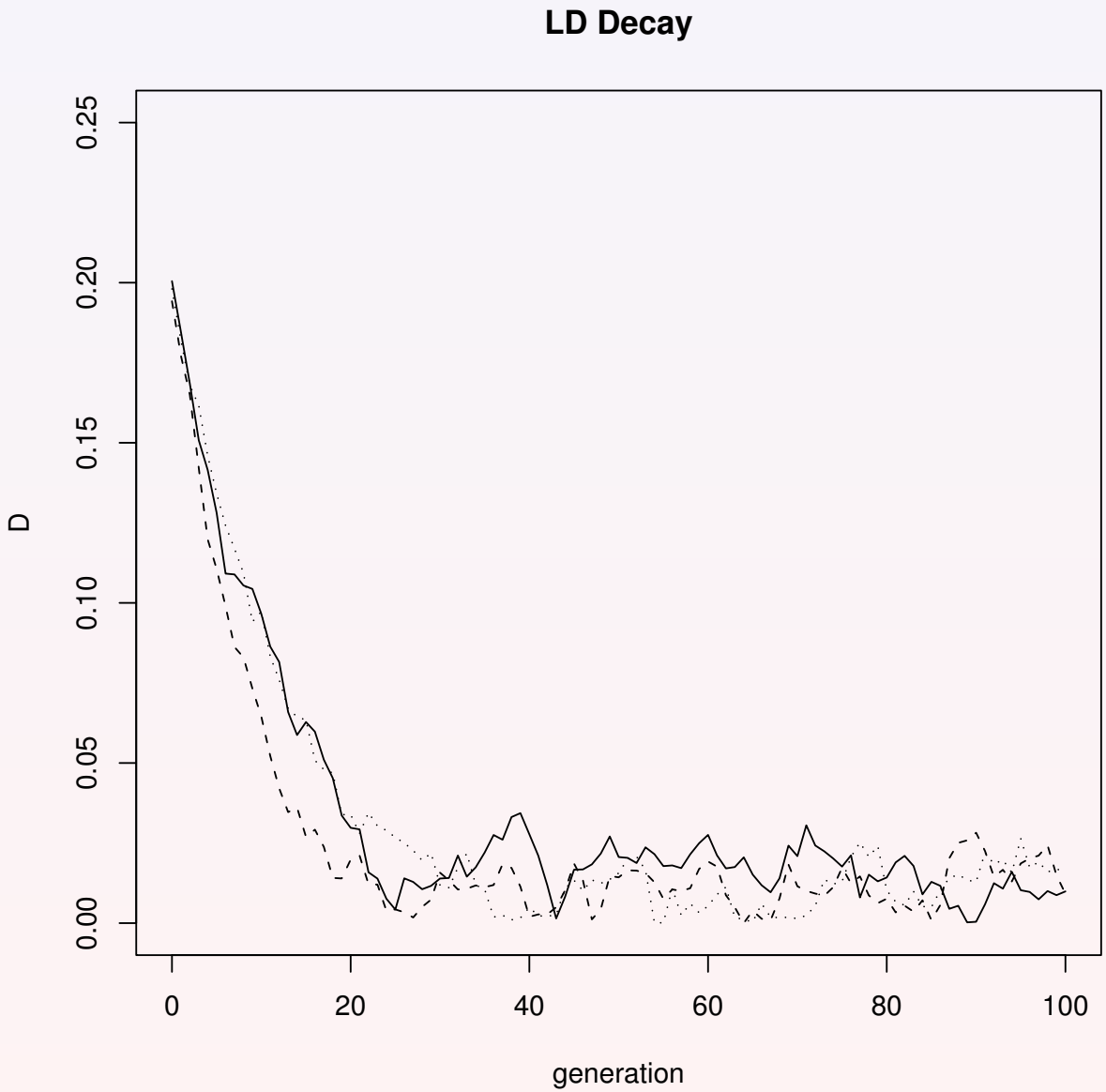
- Update at every 10 generations
- LD=0.25 before generation 0
- LD calculated at the end of each generation

Replay



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

Replay



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

Replay

Outline

3 Forward-time population generation

- simuPOP - a forward-time simulation environment
- A simple example

```
>>> 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,
...                    ylim=[0,.25], xlab='generati
...                    ylab='D', title='LD Decay'),
...         pyEval(r"'%3d   ' % gen", rep=0, step=2
...         pyEval(r"'%f    ' % LD[0][1]", step=25)
...         pyEval(r"'\n'", rep=REP_LAST, step=25)
...     ],
...     end=100
... )
```

```
  0   0.198531    0.198929    0.197586
Traceback (most recent call last):
  File "/usr/lib64/python2.3/site-packages/simuRPy.
    self.setDev()
  File "/usr/lib64/python2.3/site-packages/simuRPy.
```

simuPOP modules

population

operator

simulator

mating scheme

evolve!

Exercise time!

simuLDDecay.py

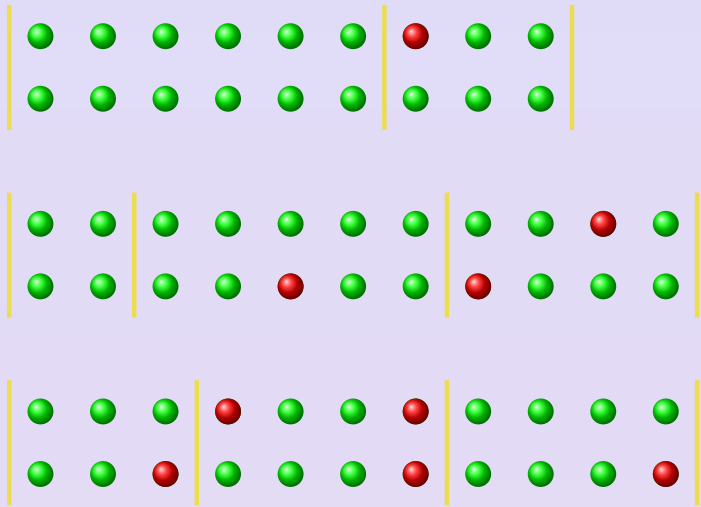
Outline

4 **simuPOP components**

- Population object
- Operators
- Mating scheme, Simulator and forward-time simulation

Structure of a population

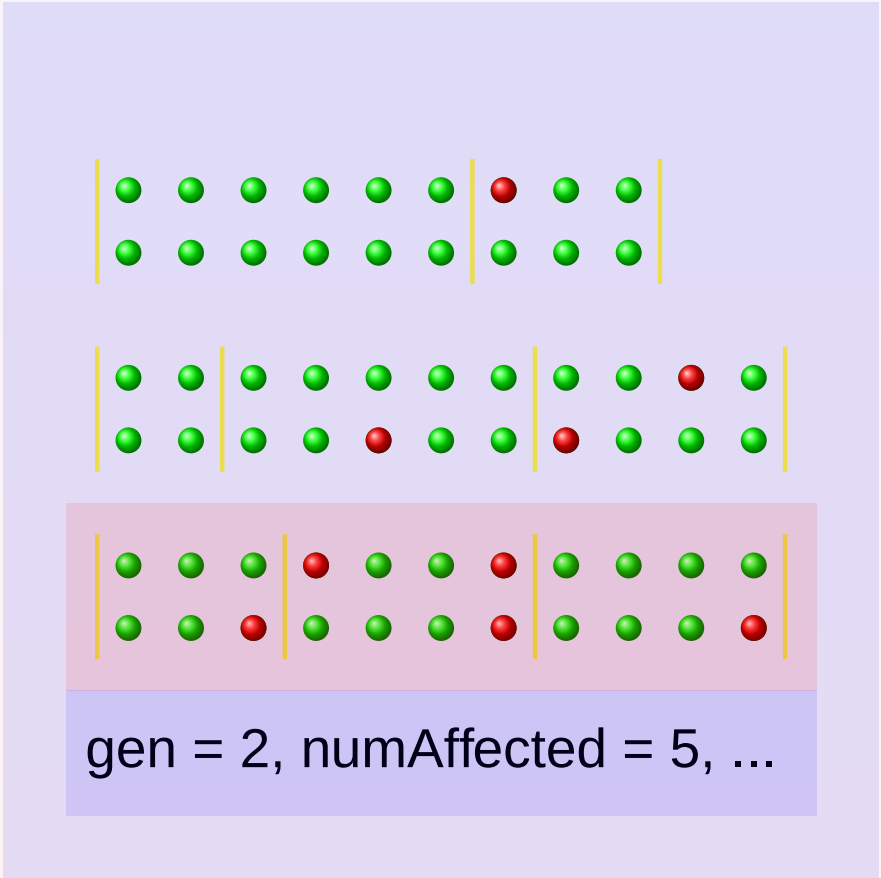
- Unaffected
- Affected



gen = 2, numAffected = 5, ...

Structure of a population

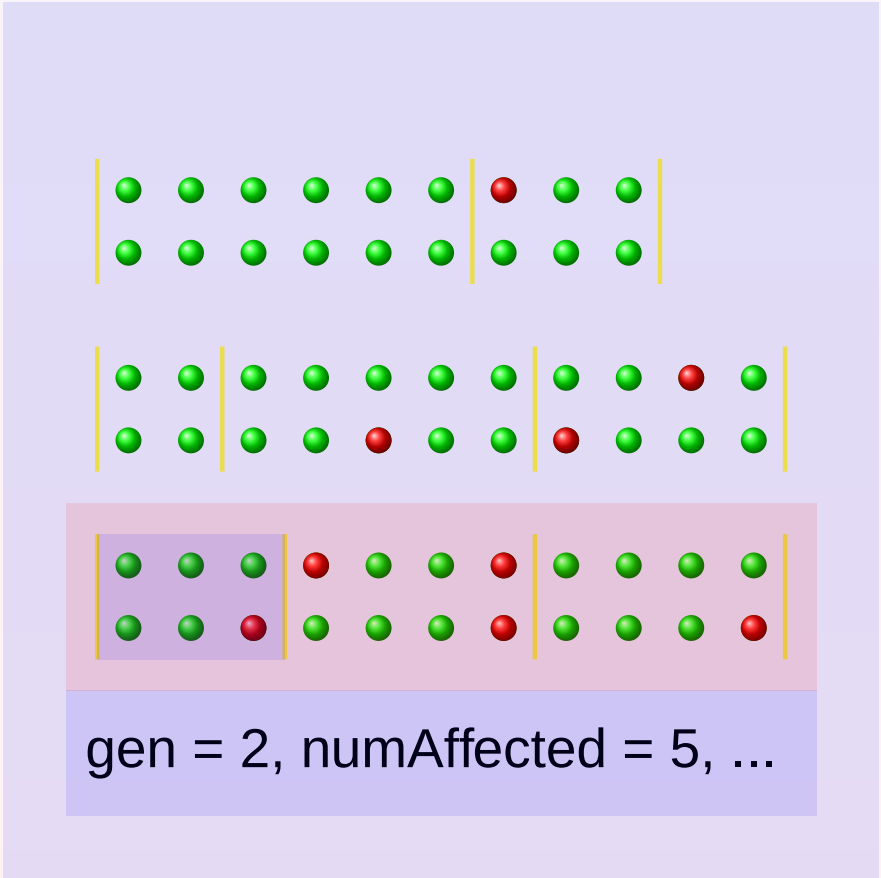
- Unaffected
- Affected



Current generation

Structure of a population

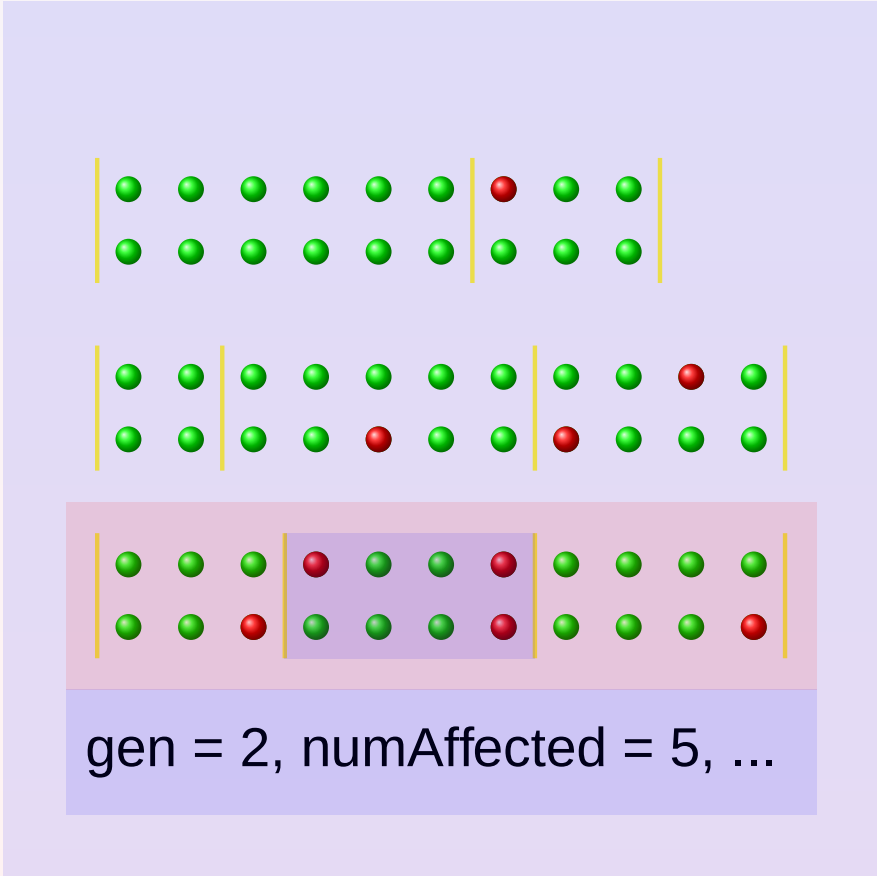
- Unaffected
- Affected



Current generation

Structure of a population

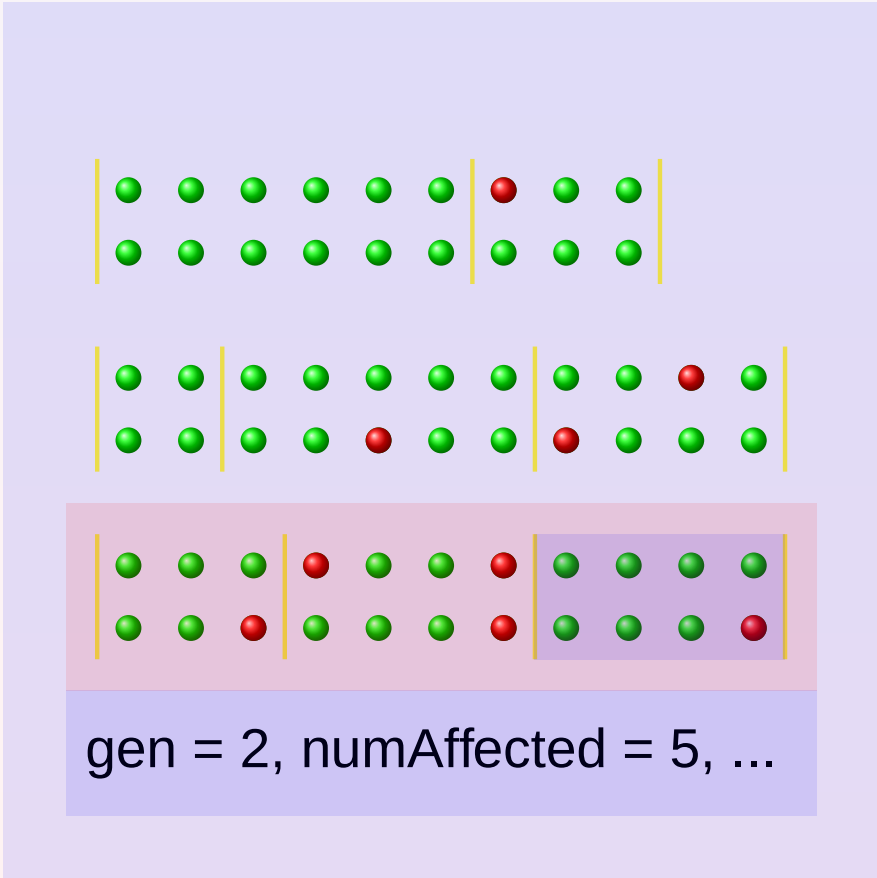
- Unaffected
- Affected



Current generation

Structure of a population

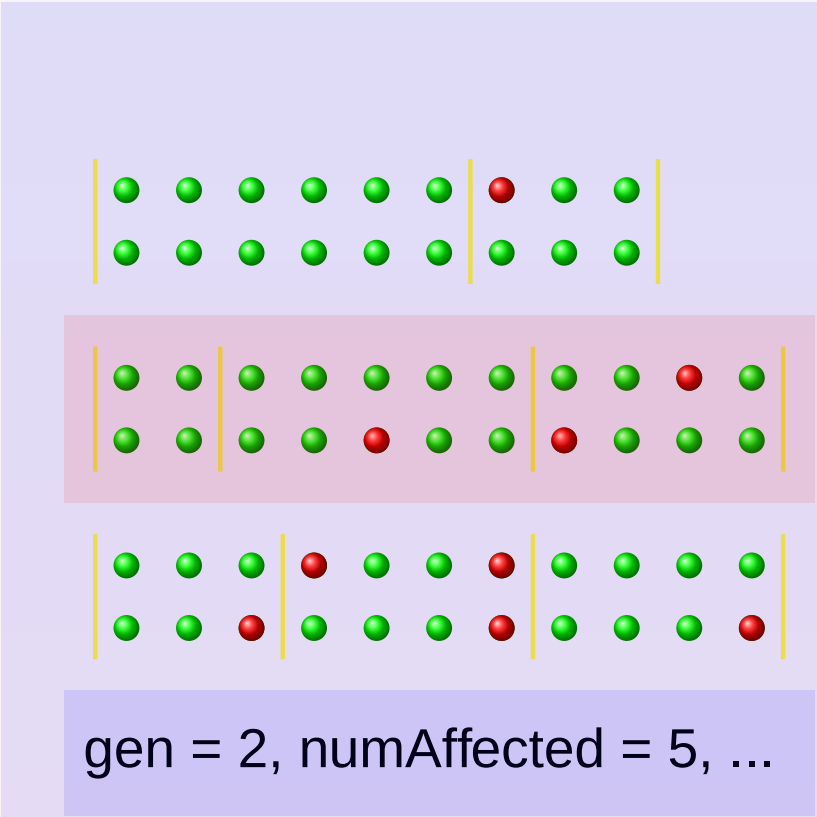
- Unaffected
- Affected



Current generation

Structure of a population

- Unaffected
- Affected

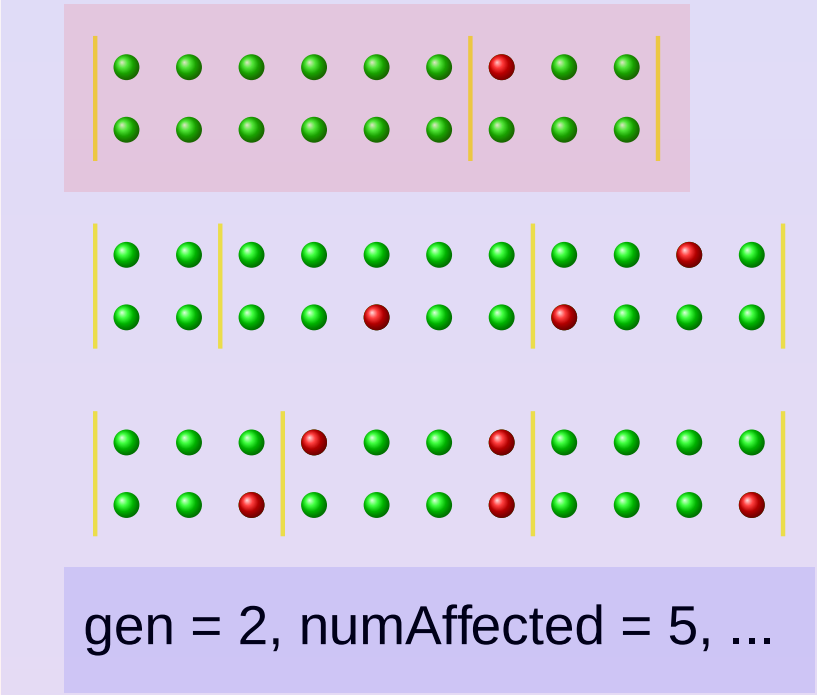


Ancestral generation 1

Current generation

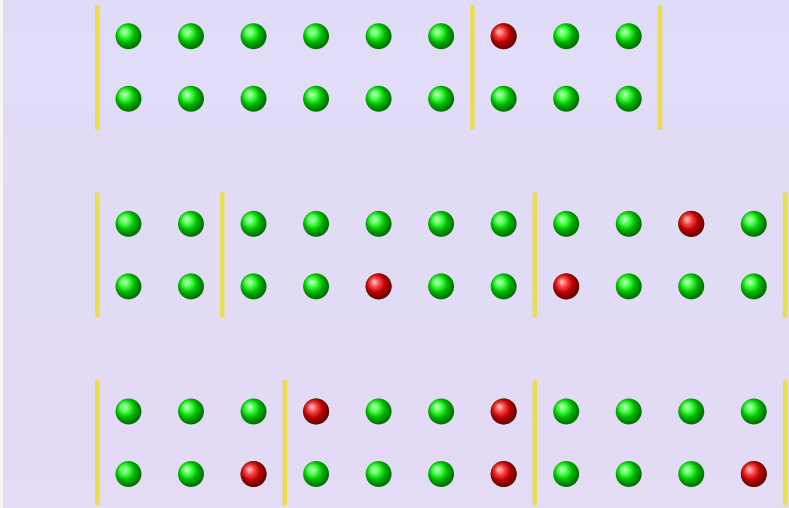
Structure of a population

- Unaffected
- Affected



Structure of a population

- Unaffected
- Affected



Ancestral generation 2

Ancestral generation 1

Current generation

gen = 2, numAffected = 5, ...

Population variables

Create a population

Genotypic structure

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

Male

- Affected

fitness	father_id	···

Chromosome 0

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

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

Chromosome 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

Population strcuture

Information fields

Variables

Outline

4 **simuPOP components**

- Population object
- Operators
- Mating scheme, Simulator and forward-time simulation

Stages

Stages, an example

Output

Table-like output

Outline

4 **simuPOP components**

- Population object
- Operators
- Mating scheme, Simulator and forward-time simulation

Mating schemes

Simulator

Evolve?!

Outline

5 A real example

- Handling of HapMap data