

Lecture 1: Modeling selection

chicken->egg->chick->chicken

Simple selection model

2 genotypes, 2 fitnesses,

finite population

constant size

asexual reproduction

no mutations

Who wins?

First, let us see what we are talking about:

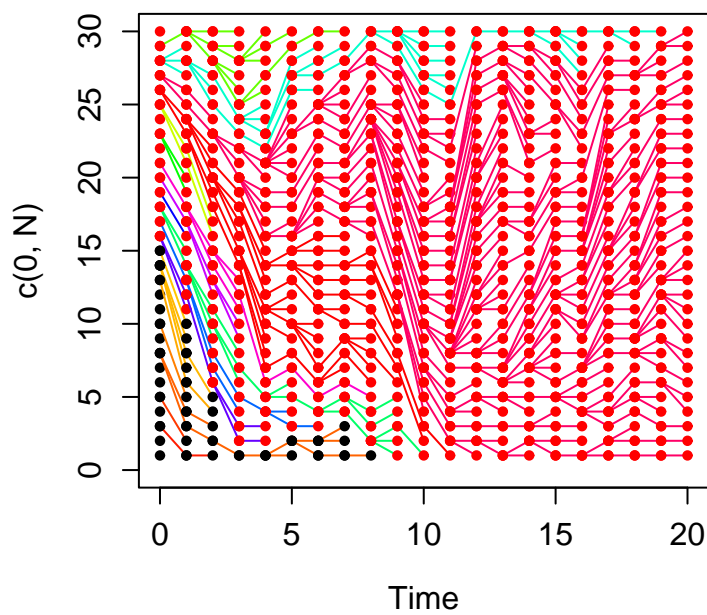
This is what generations will look like:

We execute a program in a file that I prepared.

```
> x11();source("draw1.R")
```

Writing `v()` by itself puts the drawing into the session.

```
> v()
```



Left to right you see successive generations. Each dot represents an individual. Each line connects parents to offspring. **Red** dots represent individuals with high fitness, **black** dots individuals with low fitness.

Each individual can have several offspring, or none.

>

Now let us slowly see how we make such a model.

We represent the population as a vector of genotypes, something like this:

(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2)

This vector is represented by g , with an index i that specifies the individuals:

g_i where $i = 1, 2, 3, \dots, 20$

Now, let us do the same in R:

In R everything is vectors. We prepare a variable g :

```
> g=1:20
```

g is now a vector that contains the numbers 1...20.

```
> g
```

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

If we enter a variable by itself, it gets printed out. Let us play with this:

```
> x=1:3
```

```
> x
```

```
[1] 1 2 3
```

```
> x=1:5
```

```
> x
```

```
[1] 1 2 3 4 5
```

```
>
```

```
> g[1:10]=1
```

We assign g_i with the index $i = 1...10$ to be equal to 1.

```
> g[11:20]=2
```

And those with index $i = 11...20$ to be equal to 2.

```
> g
```

```
[1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
```

Now our population is ready.

```
>
```

For simplicity, we will assume that the fitness of genotype 1 is 1, and of genotype 2 is 2.

That means that genotype 2 makes twice as many offspring as 1.

We will use fitness proportional selection. The chance of an individual to produce offspring will be proportional to its fitness.

To select parents, we will use the R function `sample`. It simply samples random entries.

```

> x=1:10
> sample(x,5)
[1] 8 6 1 10 2
> sample(x,7)
[1] 4 5 1 10 3 8 2
> sample(x,3)
[1] 4 2 8
> sample(x,10)
[1] 1 7 6 8 2 10 4 5 9 3

```

The sample function samples “without replacement”. This means that each entry can only be chosen once.

In our simulation, parents should be able to produce multiple offspring. Therefore, we want to sample “with replacement”

```

> sample(x,5,replace=TRUE)
[1] 10 2 3 8 5
> sample(x,20,replace=TRUE)
[1] 5 10 5 6 4 1 7 8 9 1 5 10 7 4 5 9 4 3 1 2
> sample(x,20)

```

```

Error in sample(length(x), size, replace, prob) :
  can't take a sample larger than the population
when replace = FALSE

```

>

Now, let us chose parents

These are the genotypes of our individuals

```

> g
[1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2

```

Our individuals go from 1 to 20

```

> i=1:20
> i
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

```

Now let us chose parents for the next generation (not taking fitness into account)

```

> sample(i,20,replace=T)
[1] 6 5 13 1 11 9 10 5 9 4 2 14 16 16 9 18 9 8 7 7

```

These could be the individuals that get to produce offspring in the next generation.

Notice that an individual occurs several times if it will produce several offspring

>

>

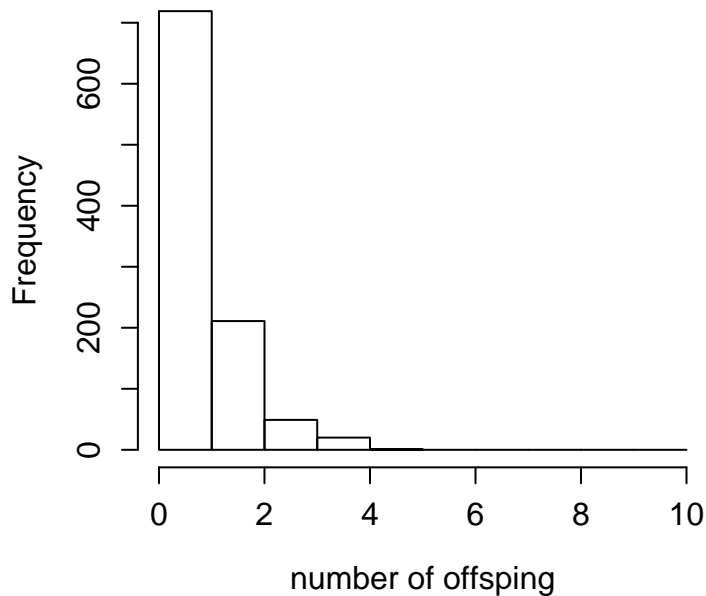
Note 1. How many offspring will an individual have with this scheme? We can easily plot a histogram for this. (Though ignore the R code for now...)

```

> x=rbinom(1000,size=20,prob=1/20)
> hist(x,breaks=seq(0,10,by=1),xlab="number of offspring", main="Histogram:
number of offspring");v()

```

Histogram: number of offspring



>

But we want to take fitness into account.

Remember that the fitnesses are 1 or 2.

> g

```
[1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
```

>

We want fitness proportional selection. The total fitness that we have is:

total fitness = $\sum_{i=1}^{20}$ (fitness of individual i)

which is

$\sum_{i=1}^{20}$ (fitness of individual i) = $1 + 1 + 1 + \dots + 1 + 2 + 2 + 2 + \dots + 2 = 1 \cdot 10 + 2 \cdot 10 = 30$

We can also calculate it like this:

> fitnesses=g

> fitnesses

```
[1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
```

> sum(fitnesses[1:20])

```
[1] 30
```

sum(fitnesses[1:20]) is just like $\sum_{i=1}^{20}$ fitness _{i}

We could have also done

> sum(fitnesses)

```
[1] 30
```

>

We want fitness proportional selection. Therefore, for each offspring, we chose one random parent from the population using these probabilities:

$$\left(\frac{1}{30}, \frac{1}{30}, \frac{1}{30}, \dots, \frac{1}{30}, \frac{2}{30}, \frac{2}{30}, \frac{2}{30}, \dots, \frac{2}{30}\right)$$

These probabilities sum to 1, and the chance for a parent with fitness 2 is double that of one with fitness 1.

```
> fitnesses=g
> sum(fitnesses)
[1] 30
> i=1:20
> sample(i,1,prob=fitnesses/30)
[1] 10
```

This is one random parent. Let us chose 20

```
> parents=sample(i,20,prob=fitnesses/30,replace=T)
> parents
[1] 13 16 9 9 15 17 2 14 18 15 10 10 5 18 14 4 20 8 9 12
```

It is easier to see what happend of we sort things

```
> sort(parents)
[1] 2 4 5 8 9 9 9 10 10 12 13 14 14 15 15 16 17 18 18 20
```

These are the individuals that get to reproduce. What are their genotypes?

```
> g[parents]
[1] 2 2 1 1 2 2 1 2 2 2 1 1 1 2 2 1 2 1 1 2
```

Before, we used `g[1:20]` to refer to g_i when $i = 1 \dots 20$.

```
> g[1:20]
[1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
```

We can also take other sub-parts of g:

```
> g[1:10]
[1] 1 1 1 1 1 1 1 1 1 1
```

```
> x=1:5
> x[1:5]=11
> x
[1] 11 11 11 11 11
```

We see entry 11 of g five times:

```
> g[x]
[1] 2 2 2 2 2
> x[1:3]=3;x[4:5]=12
> x
```

```
[1] 3 3 3 12 12
```

Now we see entry 3 three times, and entry 12 twice:

```
> g[x]
[1] 1 1 1 2 2
> g
[1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
> y=6:20
> y
[1] 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

```
> x
[1] 3 3 3 12 12
```

```
> y[x]
[1] 8 8 8 17 17
```

In this way we can refer to sub-parts of a vector.

```
> parents
[1] 13 16 9 9 15 17 2 14 18 15 10 10 5 18 14 4 20 8 9 12
```

```
> g[parents]
[1] 2 2 1 1 2 2 1 2 2 2 1 1 1 2 2 1 2 1 1 2
```

Thus we get the genotypes of the parents.

```
>
```

And that's it! We have a new set of genotypes for the next generation.

Let us sum this all up:

```
> g
[1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
```

```
> fitnesses=g
> sum(fitnesses)
```

```
[1] 30
```

```
> individuals=1:20
> parents=sample(individuals,20,prob=fitnesses/sum(fitnesses), replace=T)
> parents
```

```
[1] 11 13 10 2 17 15 10 16 13 10 13 17 11 13 11 2 20 9 15 4
```

```
> g[parents]
[1] 2 2 1 1 2 2 1 2 2 1 2 2 2 2 2 1 2 1 2 1
```

```
> g.offsprings=g[parents]
```

```
> g.offsprings
[1] 2 2 1 1 2 2 1 2 2 1 2 2 2 2 2 1 2 1 2 1
```

```
>
```

We have one generation!

Ok. That was one generation. Now we want to do several. To do that we need to do what I showed above several times. For this we use a loop.

A loop will execute several statement several times.

For example

```
> for(generation in 1:5) {
  print("hello")
}
+ + [1] "hello"
[1] "hello"
[1] "hello"
[1] "hello"
[1] "hello"
```

```

> for( generation in 1:3) {
  print("hello")
  print("there")
}

+ + + [1] "hello"
[1] "there"
[1] "hello"
[1] "there"
[1] "hello"
[1] "there"

```

This is how you construct loops:

```

for( variable in 1:N ) {
  what to do...
}

```

The inside part of the loop will get executed with the value of the variable changing over the range that we gave. Thus:

```

> for( x in 1:5) {
  print(x)
}

+ + [1] 1
[1] 2
[1] 3
[1] 4
[1] 5

```

Let us do 10 generations:

```

> for( generation in 1:10) {
  individuals=1:20
  fitnesses=g
  parents=sample(individuals,20,replace=TRUE,prob=fitnesses/sum(fitnesses) )
  g.offsprings=g[parents]
  print(g.offsprings)
  g=g.offsprings
}

+ + + + + + + [1] 2 2 1 2 2 1 1 2 1 2 2 1 2 1 2 2 2 1 2 1
[1] 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 1 1 1 2
[1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 2
[1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1
[1] 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

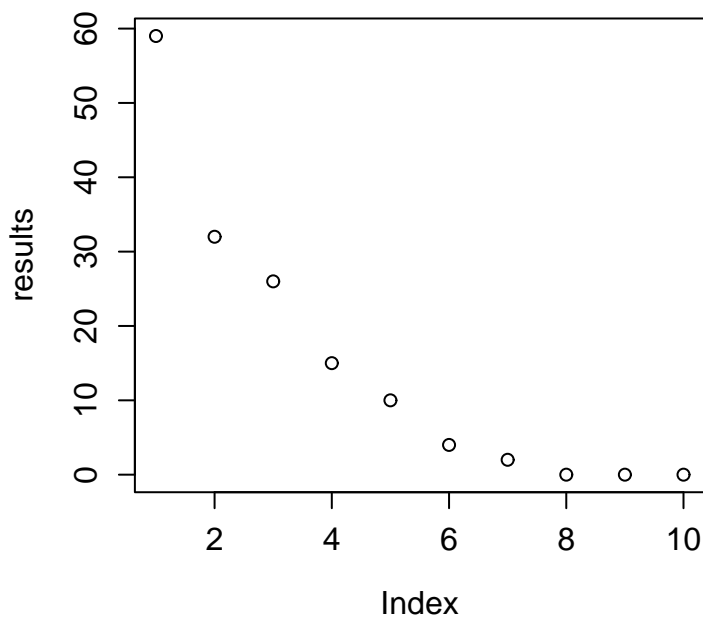
```

very quickly everybody has genotype 2!
 Selection is very strong, and the population is very small.


```

> g=1:200; g[1:100]=1; g[101:200]=2
> for( generation in 1:10) {
  individuals=1:200
  fitnesses=g
  parents=sample(individuals,200,replace=TRUE,prob=fitnesses/sum(fitnesses) )
  g.offsprings=g[parents]
  g=g.offsprings
  results[generation]=( sum(g==1) )
}
+ + + + + + + >
> plot(results);v()

```



>

Till now we used fitnesses 1 and 2. And we used the trick that the genotype of the organism was also its fitness. Let us try to overcome this problem.

We want individuals with genotype 1 to have fitness 1, and those with genotype 2 fitness 1.1

We actually already learned how to do this!

```

> f=1:2
> f[1]=1
> f[2]=1.1
> f
[1] 1.0 1.1
f contains the fitnesses.
Here are the genotypes:
> g=1:10; g[1:5]=1; g[6:10]=2
> g
[1] 1 1 1 1 1 2 2 2 2 2
> f[1]

```

```

[1] 1
> f[2]
[1] 1.1
> f[g]
[1] 1.0 1.0 1.0 1.0 1.0 1.1 1.1 1.1 1.1 1.1
>

```

We saw before how we can take parts of a vector, and each part can occur in the resulting vector several times. (When we constructed the offspring fitnesses). This is exactly what we do here.

So, we will use the following line:

```

> fitnesses=f[g]
>

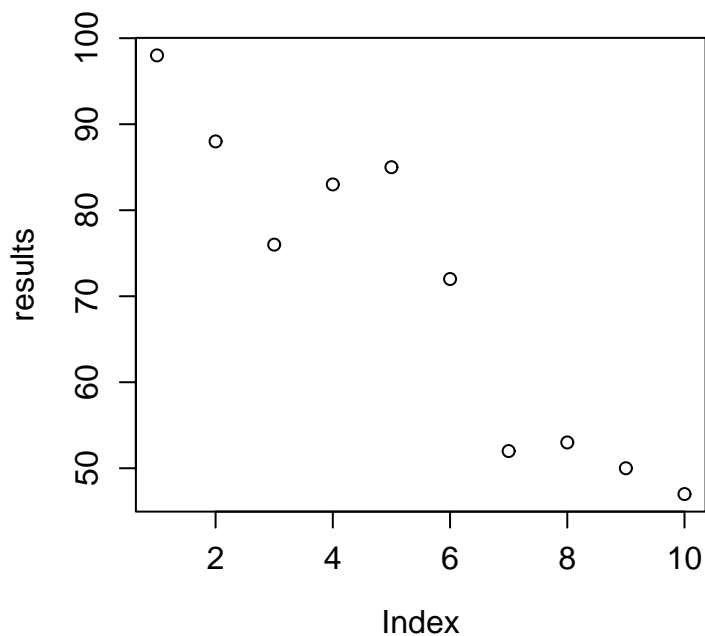
```

Now let us do a new simulation

```

> g=1:200; g[1:100]=1; g[101:200]=2
> f=1:2; f[1]=1; f[2]=1.1
> results=1:10
> for( generation in 1:10) {
  individuals=1:200
  fitnesses=f[g]
  parents=sample(individuals,200,replace=TRUE,prob=fitnesses/sum(fitnesses) )
  g.offsprings=g[parents]
  g=g.offsprings
  results[generation]=( sum(g==1) )
}
++++++>
> plot(results);v()

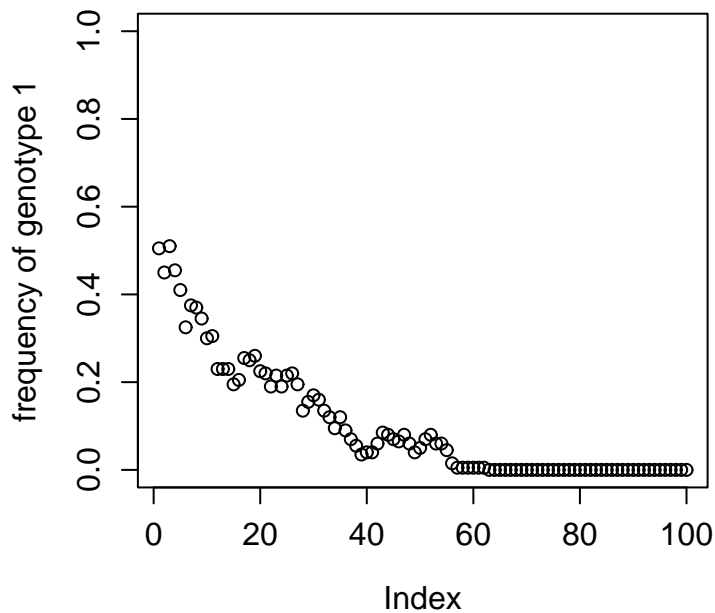
```



>

With 10% fitness difference and a population size of 200, there was just a bit of change over 10 generations. Let us do 100 generations:

```
> g=1:200; g[1:100]=1; g[101:200]=2
> f=1:2; f[1]=1; f[2]=1.1
> results=1:100
> for( generation in 1:100) {
  individuals=1:200
  fitnesses=f[g]
  parents=sample(individuals,200,replace=TRUE,prob=fitnesses/sum(fitnesses) )
  g.offsprings=g[parents]
  g=g.offsprings
  results[generation]=( sum(g==1)/200 )
}
>
> plot(results,ylim=c(0,1),ylab="frequency of genotype 1");v()
```



>

Selection of Altruists!

Now genotype 1 are altruists.

Genotype 1 has low fitness in the population, but causes the population to increase in size.

So, instead of a constant size population, the population increases in size.

The population will grow according to how many altruists there are in it:

$$N' = N \cdot \left(1 + \frac{n_1}{N}b\right)$$

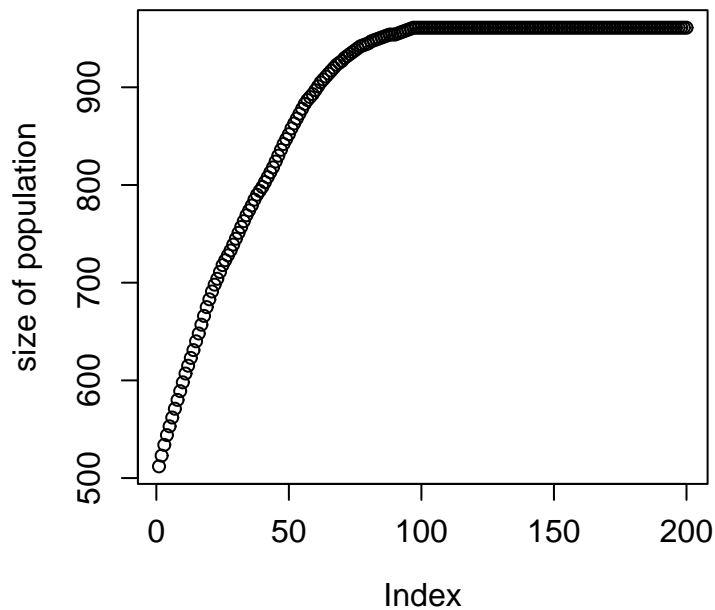
where b is the benefit of altruists

```
> N=500;b=0.05
> g=1:N; g[1:(N/2)]=1; g[(N/2+1):N]=2
> f=1:2; f[1]=1; f[2]=1.05
> results.p1=1:200;results.N=1:200
> for( generation in 1:200) {
  individuals=1:N
  fitnesses=f[g]
  Np=N*(1+sum(g==1)/N*b)
  parents=sample(individuals,Np,replace=TRUE,prob=fitnesses/sum(fitnesses) )
  g.offsprings=g[parents]
  g=g.offsprings
  N=length(g)
  results.p1[generation]=( sum(g==1) )
  results.N[generation]=N
}
```

We only play with N and N_p now - everything else is the same.

+

```
> plot(results.N,ylab="size of population");v()
```



```
> plot(results.p1,ylab="frequency of altruists");v()
```

