Genetic Algorithms Notebook

This notebook walks through basic genetic algorithms and tracking their performance over time. These algorithms will make up the basis of the phenotypic evolution of honeybees within the colony,so this is a good place to start. I have written several functions in other .R files that will be utilized here.

Introduction

Genetic Algorithms

A genetic algorithm is a form of optimization used in computer science belonging to a larger class of evolutionary algorithms, which in general begin with a population of random inputs to a function and impose slow changes to approach a better solution. Genetic algorithms are used to optimize inputs according to some fitness function, however complex or abstract it may be. As its name implies, genetic algorithms are inspired by the process of natural selection and, specifically, some factors of genetics that allow for beneficial mutations such as crossover.

In a genetic algorithm, a population of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem is evolved toward better solutions. Each candidate solution has a set of properties (its chromosomes or genotype) which can be mutated and altered; traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible. (Wikipedia; Whitley, Darrell (1994))

Applying A Genetic Algorithm

The remainder of this notebook will be dedicated to a simple example of a genetic algorithm. In this example, I walk through creating a sets of individuals defined by phenotypes, assessing their "fitness", mutating them in a variety of ways and reassessing their fitness.

What is a "Phenotype?"

A phenotype is the lowest element level in this simulation. In nature, phenotypes are the physical of behavioral representations of the effects of genes. Then, some sort of fitness functions of these phenotypes determines the overall fitness of individual. Note that fitness functions in nature are a way to express naturally selective forces on reproduction. In this simulation, I bypass genes and represent them as phenotypes, a simple number that can be input into a fitness function. The most fit individual will have the combination of phenotyes that give the greatest fitness function value. Here is an example fo a phenotype:

```
phenotype <- rnorm(1,0,1)
phenotype
```

```
## [1] -0.5927112
```

What is an Individual?

An individual in this simulation is exactly what it is in nature, a single animal or single unit of selection. Individuals will be defined as a 2D vector of phenotypes determined by a random distribution. That is, each individual is made up of two phenotypes represented as a 2D vector that can easily be analyzed by a fitness function. Here is an example of an individual:

```
individual <- data.frame(var1=rnorm(1,0,1),
                         var2=rnorm(1,0,1))
individual
```

```
##       var1     var2
## 1 2.211985 1.457085
```

What is a Species?

I will call a single group of individuals sharing the same genetic foundation a species and are in the same reproduction and competition pool. A "genetic foundation" is the particular distribution used to define their phenotypes. Additionally, the reproduction and competition pools are groups that only produce and compete with each other. Here is an example of a species:

```
numIndivs <- 2
makespecies <- function(numIndivs)
{
  species <- data.frame(var1=rnorm(numIndivs,0,1),
                        var2=rnorm(numIndivs,0,1))
  return(species)
}
makespecies(numIndivs)
```

```
##        var1      var2
## 1  1.766207 -1.520160
## 2 -1.096691  2.549556
```

What is a Population?

A population, in this simulation, is a large group of individuals consisting of a number of species. That is, a number of groups of individuals, each of which are produced by differing distributions and only compete within groups. Here is an example of a population:

```
numSpecies <- 2
population <- data.frame(speciesNum=0,var1=0,var2=0)
for(i in 1:numSpecies)
{
  population <- rbind(population,
                      mutate(speciesNum = rep(i,numIndivs),
                             makespecies(numIndivs)))
}
population[-1,]
```

```
##   speciesNum       var1       var2
## 2          1 -0.7623985 -0.6072554
## 3          1 -0.9849049 -0.7655035
## 4          2 -0.5115234  0.2720412
## 5          2 -0.5340181 -2.7796069
```

Essentially, a population is a compilation of a number of species, identified by the number in the `speciesNum` column.

Fitness Functions

In nature, each individual has a "fitness" that is a function of their phenotypes. For example, a longer neck allows a giraffe to reach more food increasing its fitness. Therefore, the fitness function will reward genes that contribute to a higher neck, assuming it does not impede functions in any other way. As you can imagine, accounting for impeding other functions becomes very complex, especially given the number of genes and possible interactions. In this simulation, I use several fitness functions of differing complexity, all of only two variables. Here is an example of a fitness function:

```
fitnessFunction <- function(x,y)
{  return(x+y)  }
fitnessFunction(1,2)
```

```
## [1] 3
```

Reproduction and Mutations

In nature, individuals produce offspring that are *similar* to themselves. Several types of reproduction exist in nature, most of which involve the combination of two sets of DNA, followed by a possiblity for mutations. In this simulation, each new individual following the first generation is either a slightly mutated version of a "parent" or an entirely new individual, not dependent on the phenotypes of any other individual in their species. Notably, though, the phenotypes of new individuals are determiend by the same distributions as the species they belong to.

Here is an example of **new** type creation:

```
parent <- data.frame(var1=rnorm(1,0,.5),
                     var2=rnorm(1,0,.5))

child <- data.frame(var1=rnorm(1,0,.5),
                    var2=rnorm(1,0,.5))
```

Here is an example of **child** type creation:

```
parent <- data.frame(var1=rnorm(1,0,.5),
                     var2=rnorm(1,0,.5))

offspring <- data.frame(var1=parent$var1*rnorm(1,1,.1),
                        var2=parent$var2*rnorm(1,1,.1))
```

Notice here how the offspring only differ from their parents by `rnorm(1,1,.1)`, which gives a small yet noticeable mutation to their parent.

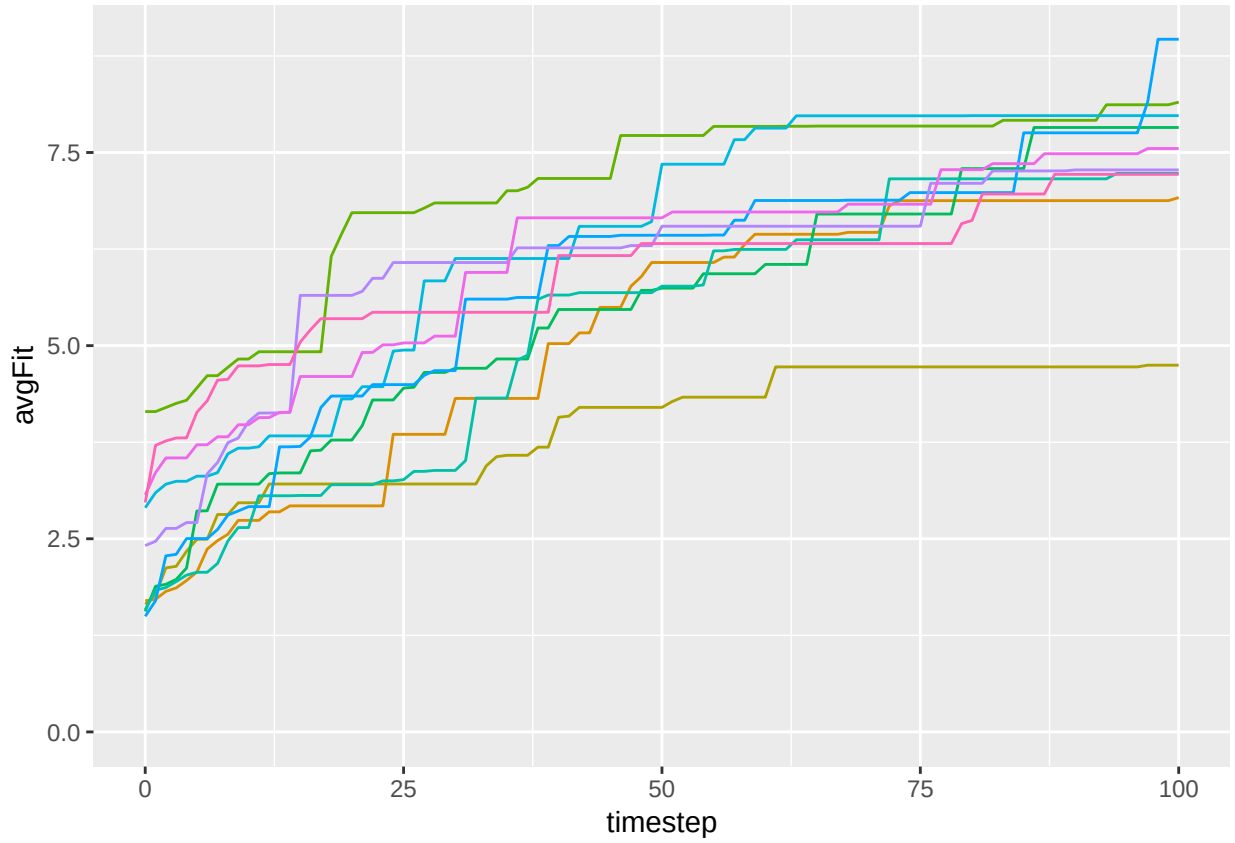
Evolution

```
fitfunc <- function(x,y)
{return((x^2+y^2))}
```

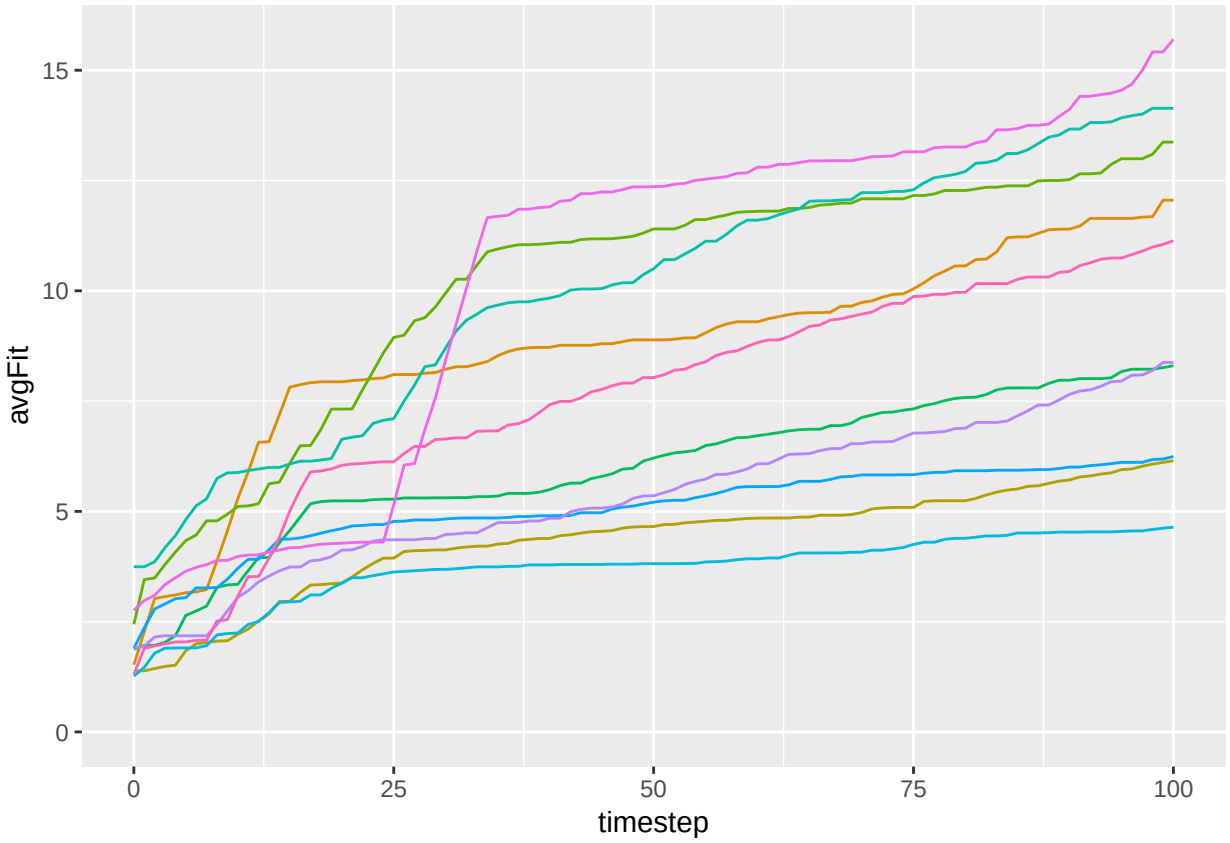
Function Running

```
geneticSpecies(fitfunc,gens=100,indivs=3,creation="child")
```

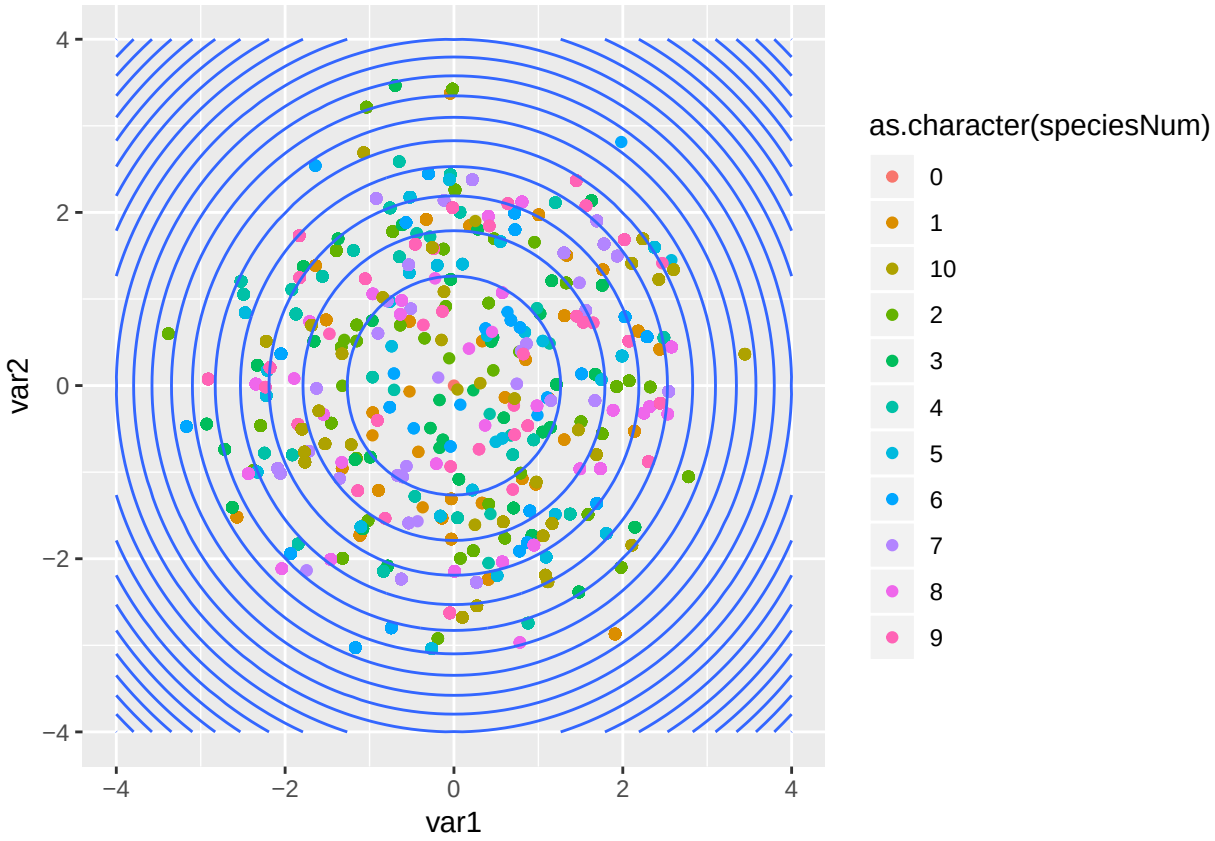
```
geneticPopulation(fitfunc,species=10,indivs=10,gens=100,creation="new",plot="time")
```



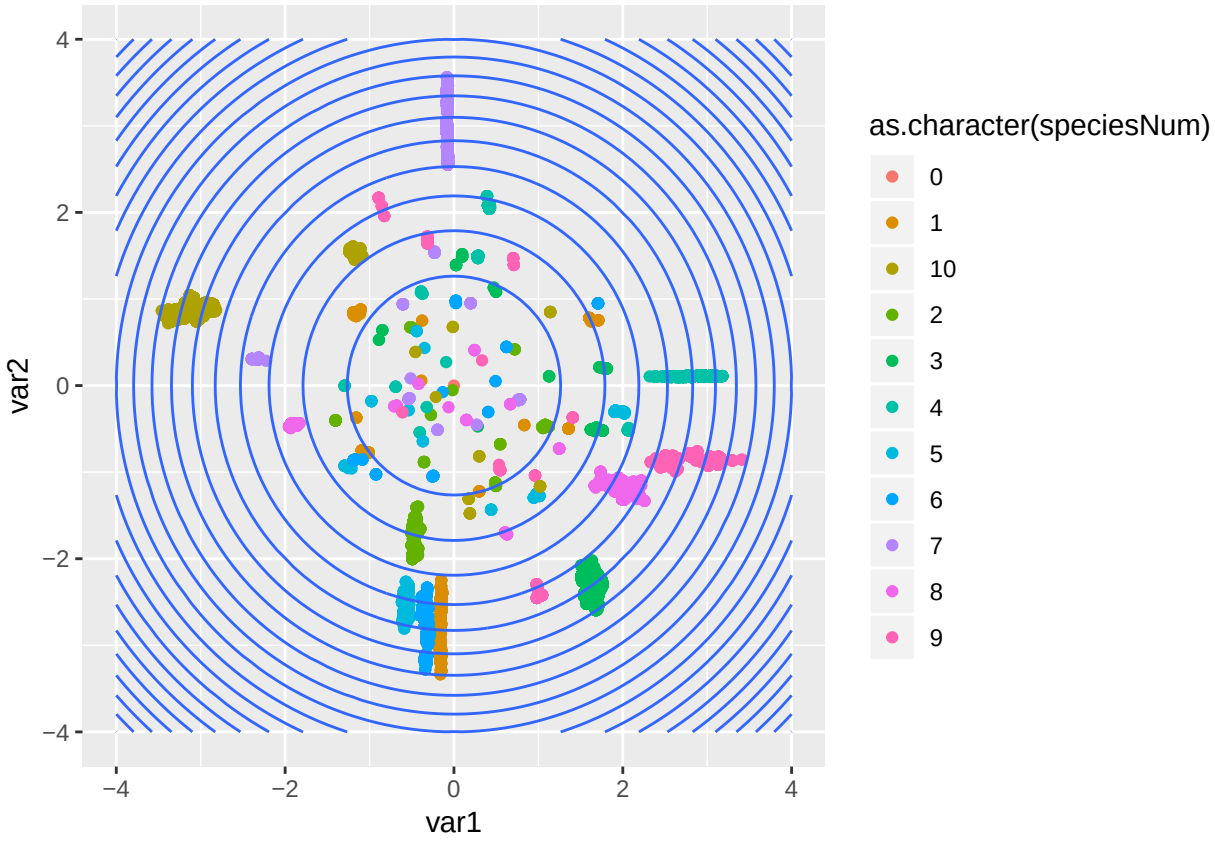

```
geneticPopulation(
  fitfunc,species=10,indivs=10,gens=100,creation="child",plot="time")
```




```
geneticPopulation(
  fitfunc,species=10,indivs=10,gens=100,creation="new",plot="maxVals")
```




```
geneticPopulation(
  fitfunc,species=10,indivs=10,gens=100,creation="child",plot="maxVals")
```


4
2
0
−2
−4
var2
−4
−2
0
2
4
var1
as.character(speciesNum)
0
1
10
2
3
4
5
6
7
8
9