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] -0.292987
```

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 -0.4061646 -0.2911063
```

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 1.2800950 -0.5219903
## 2 0.1243675 -1.9569059
```

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 -2.5228098  1.06321574
## 3          1 -0.9421543 -0.47809286
## 4          2  0.1943341  0.33867244
## 5          2  0.1536573  0.07954704
```

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

In nature, the most fit individuals are the most likely to reproduce and pass on their genes, creating a relatively fit individual compared to the population. In my simulation, as a way to accelerate the increase in maximum and mean fitness, a new individual each time step replaces the individual with the lowest fitness. Take a look at the following application to better understand and visualize this evolution:

The Application

First, I'll define the funciton used to calculate the fitness of each individual:

```
fitfunc <- function(x,y)
{return((y*sin(x)+x*cos(y)))}
```

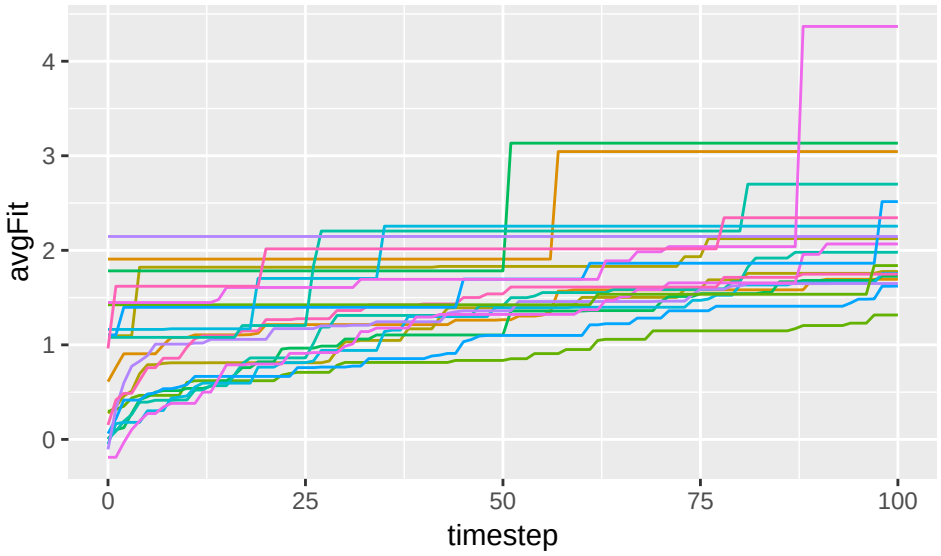
Now, let's look at the evolution of a population in terms of average fitness:

Average Fitness Over Time

The following plot is a visual representation of the increase in mean fitness of a population, separated species. This population created new individuals using the **new** creation type, meaning each individual comes from

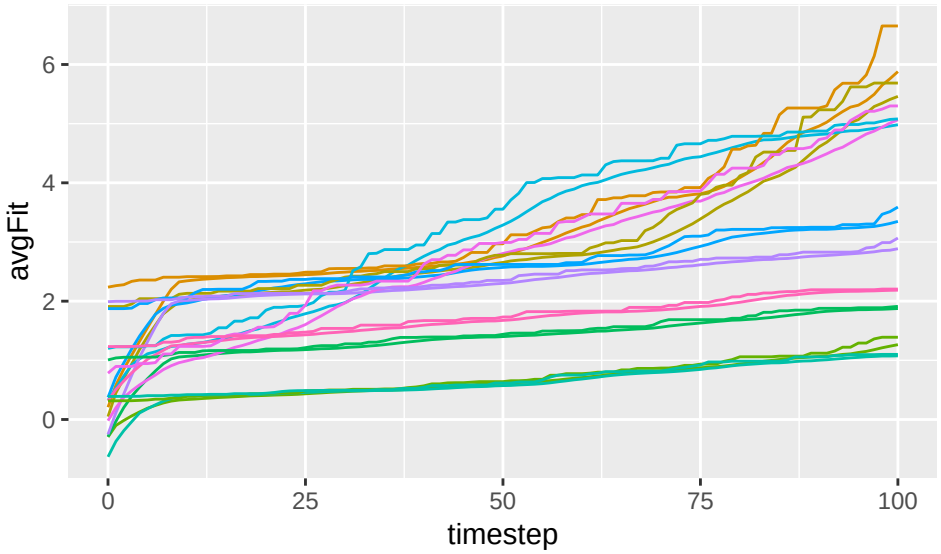
exactly the same distributions as its parents. Here we consider 10 species, each with 10 individuals over 100 generations.

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



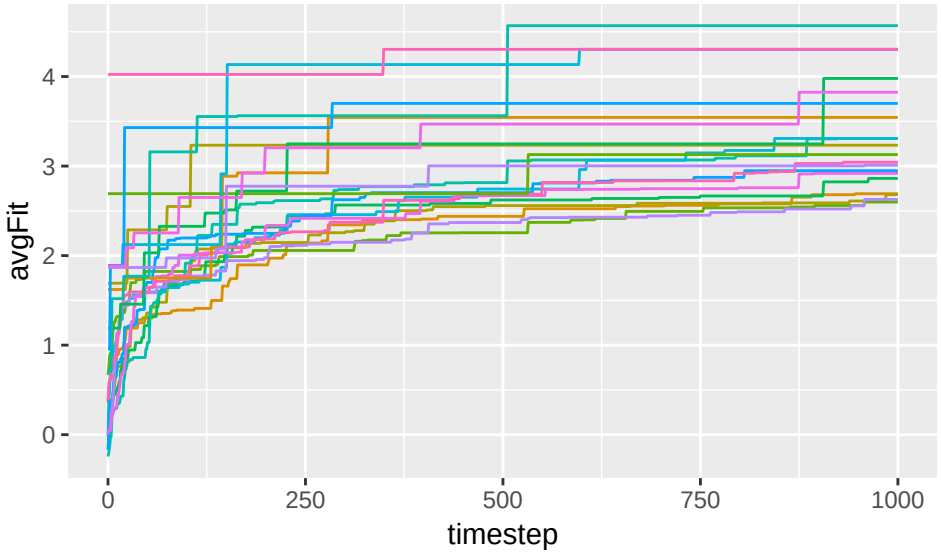
Now, let's look at the same plot from a population evolved by the **child** creation method, which slightly mutates a parent in order to create each new child.

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



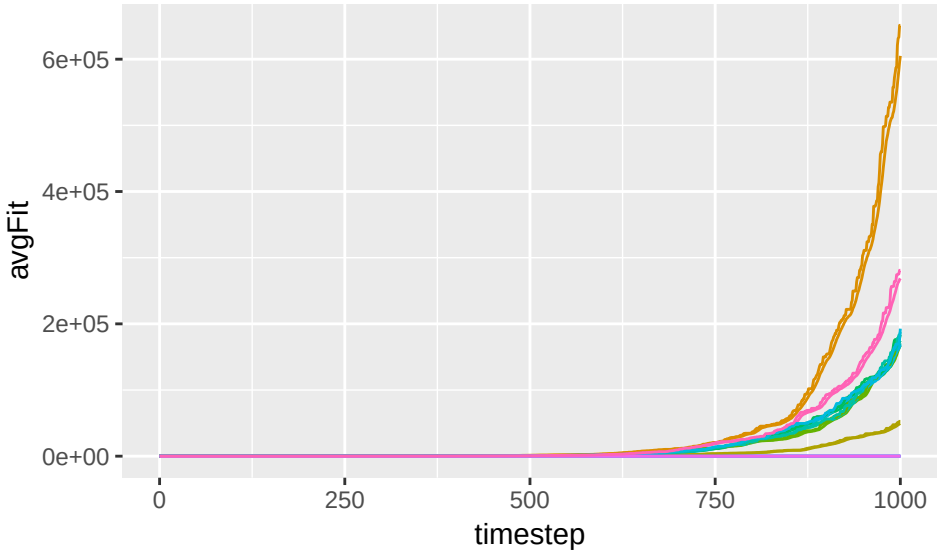
These two plots looks fairly similar in structure, but let's see what happens when we expand their evolution out to 1000 generations.
First from the **new** creation type:

```
geneticPopulation(fitfunc,species=10,indivs=10,gens=1000,creation="new",plot="time")
```



Now from the **child** creation type:

```
geneticPopulation(fitfunc,species=10,indivs=10,gens=1000,creation="child",plot="time")
```

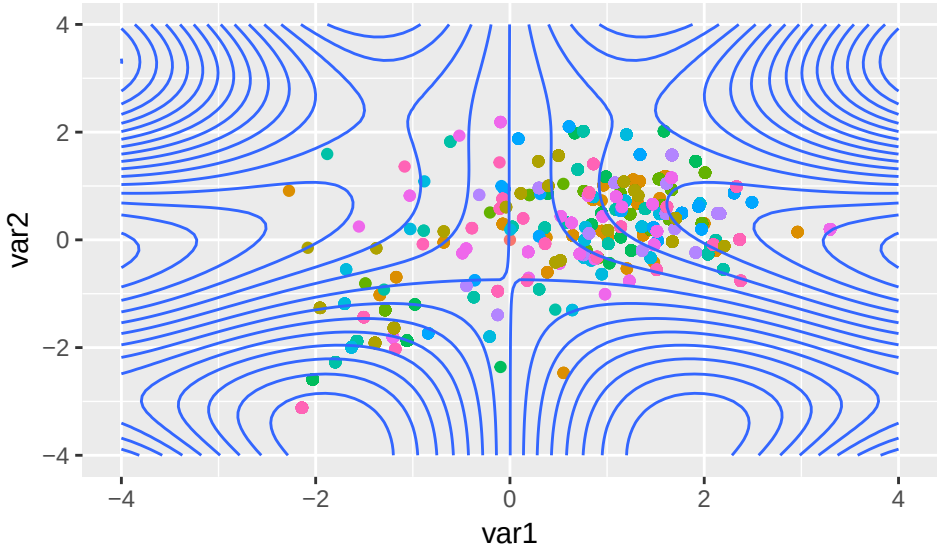


The **new** creation type population reaches a maximum at fitness score of ~3, while the **child** creation type population soars past that in fitness. This is because the **child** population is able to benefit from continuous mutations on a successful phenotype, while the **new** population is limited by the probability given in the distributions used to craete the individuals. That is, it is extremely unlikely for the distribution used to produce the phenotype found by the evolution of the **child** population.

This leads well into the great part about genetic algorithms: their ability to discover local or global maxima or minima to a "fitness" function that may not have been expected. The following plots are a great way to visualize the way that these population can "approach" a local maxima. Specifically, they show a contour plot of the fitness function overlayed onto a scatterplot of the `var1` and `var2` phenotypes of the individual with the highest fitness score from each population at each generation.

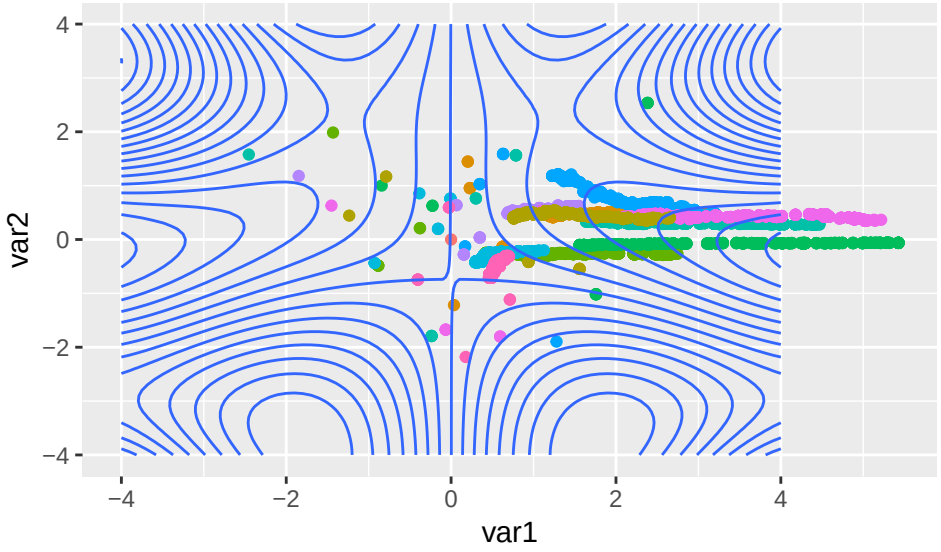
First, let's look at a population with **new** creation method with 10 species, each with 5 individuals over 100 generations.

```
geneticPopulation(fitfunc,species=10,indivs=5,gens=100,creation="new",plot="maxVals")
```



Now, here is the exact same plot using the **child** creation method.

```
geneticPopulation(fitfunc,species=10,indivs=5,gens=100,creation="child",plot="maxVals")
```

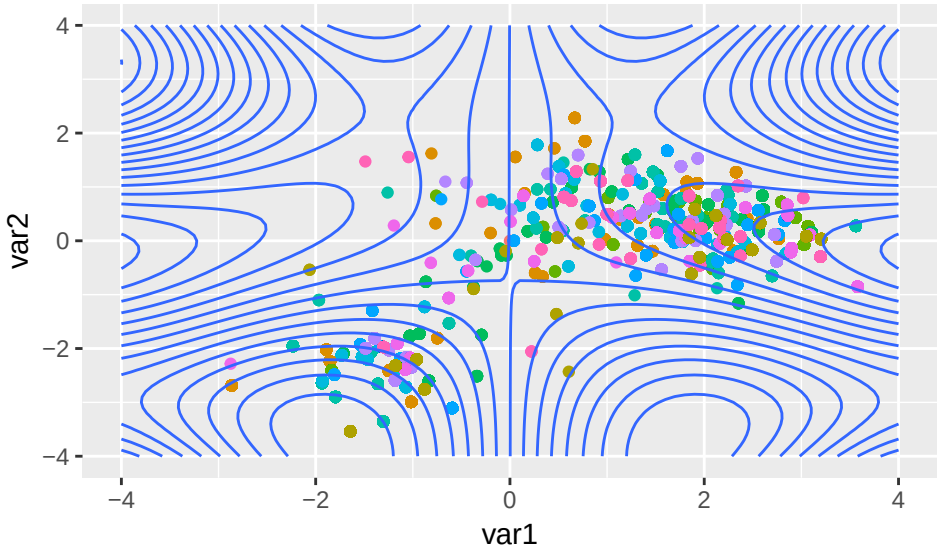


The major difference to note here is what look like clumps or lines of dots in the second plot. As would be expected using the **child** creation method, the children are closely related to their parents, which is represented by a spatial proxomity on this plot. That is, each progresive generation *should* theoretically be a small step from the last generation towards a local (ideally global) maxima of the fitness function. Note that the gradient of a contour plot is perpendicular to the plot lines, making this the optimal direction of travel. My hypothesis for the tendency of these point clusters to "travel" along a single axis is this:
It is relatively unlikely that both phenotypic mutations produce a great increase in fitness in any given generation, while it is fairly likely that a slight change in one of them cause a notable increase in fitness. Therefore, generational mutations are significantly more likely to be effected on the variable with a larger

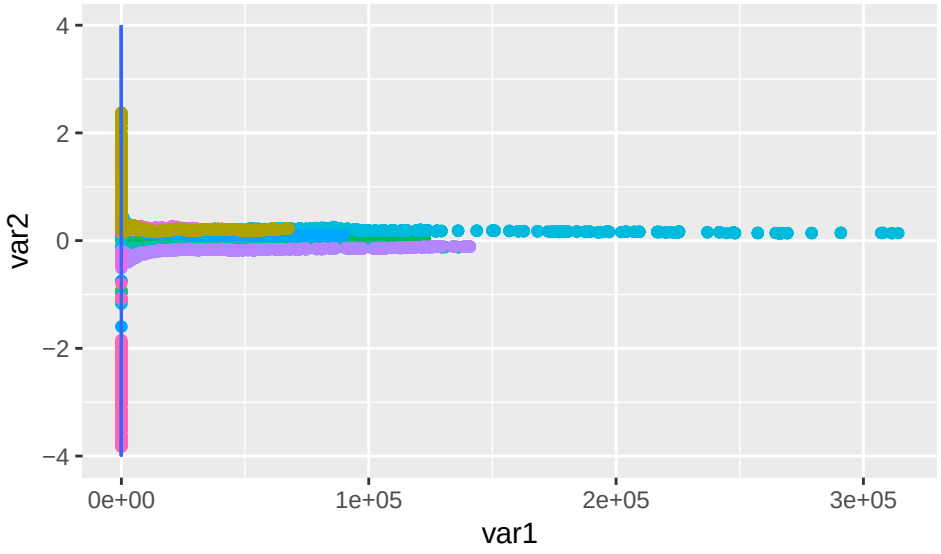
gradient value with respect to the fitness function than the other.

Lastly, let's look at these two plots over 1000 generations.

```
geneticPopulation(fitfunc,species=10,indivs=5,gens=1000,creation="new",plot="maxVals")
```

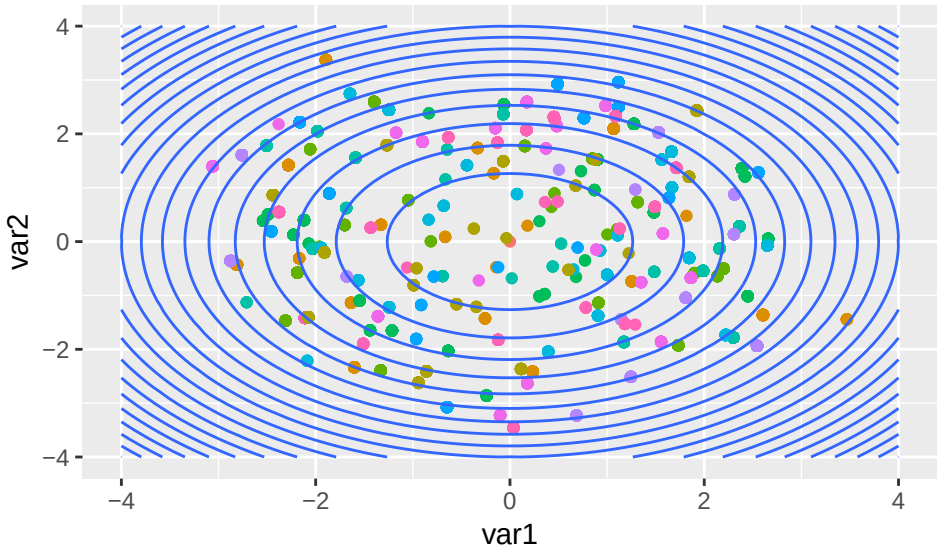



```
geneticPopulation(fitfunc,species=10,indivs=5,gens=1000,creation="child",plot="maxVals")
```

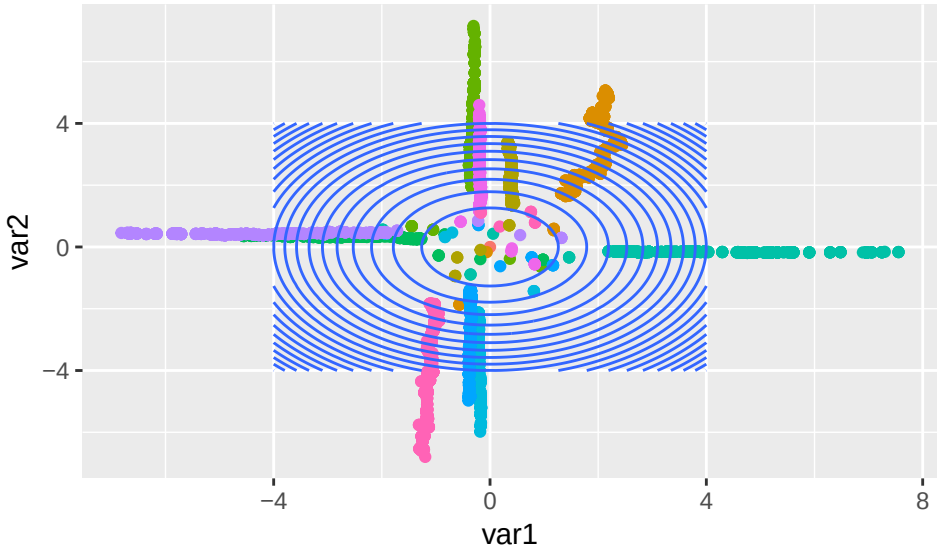


TODO: Consider other functions in the contour plot to show **local maxima** and **gradient ascent**

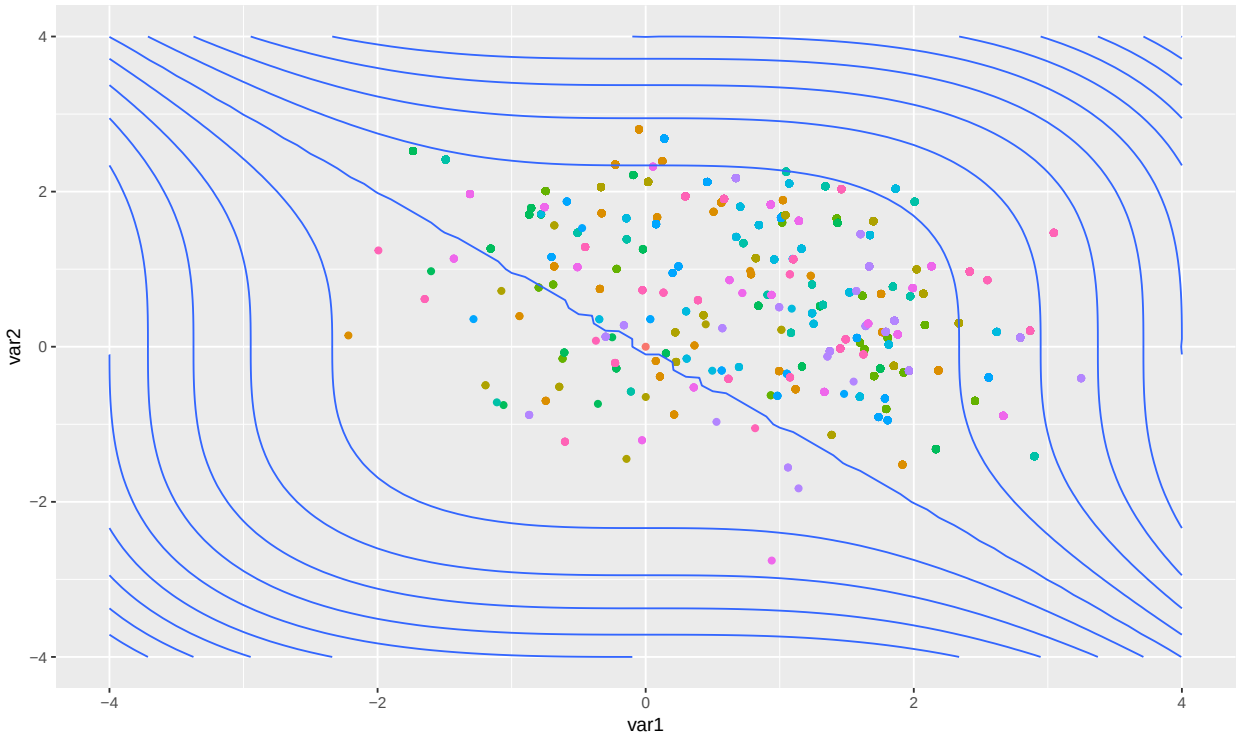
```
fitfunc <- function(x,y) {return(x^2+y^2)}
geneticPopulation(fitfunc,species=10,indivs=5,gens=100,creation="new",plot="maxVals")
```



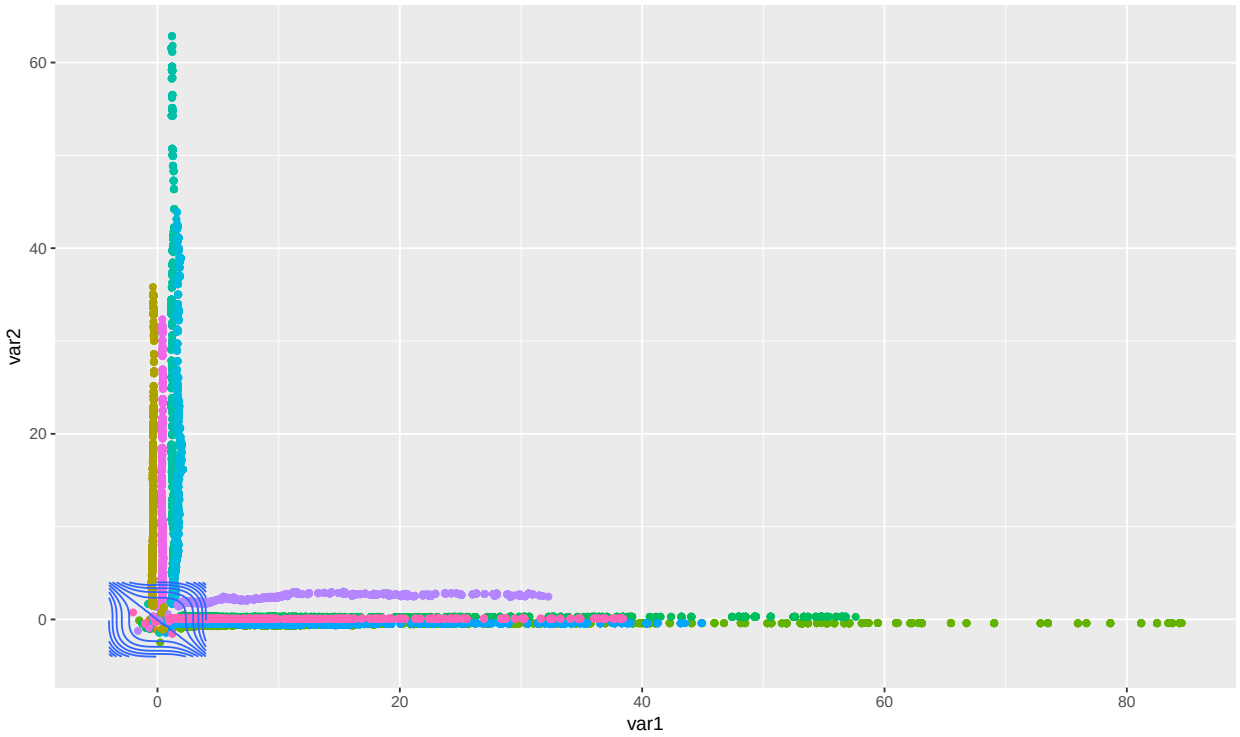

```
geneticPopulation(fitfunc,species=10,indivs=5,gens=100,creation="child",plot="maxVals")
```



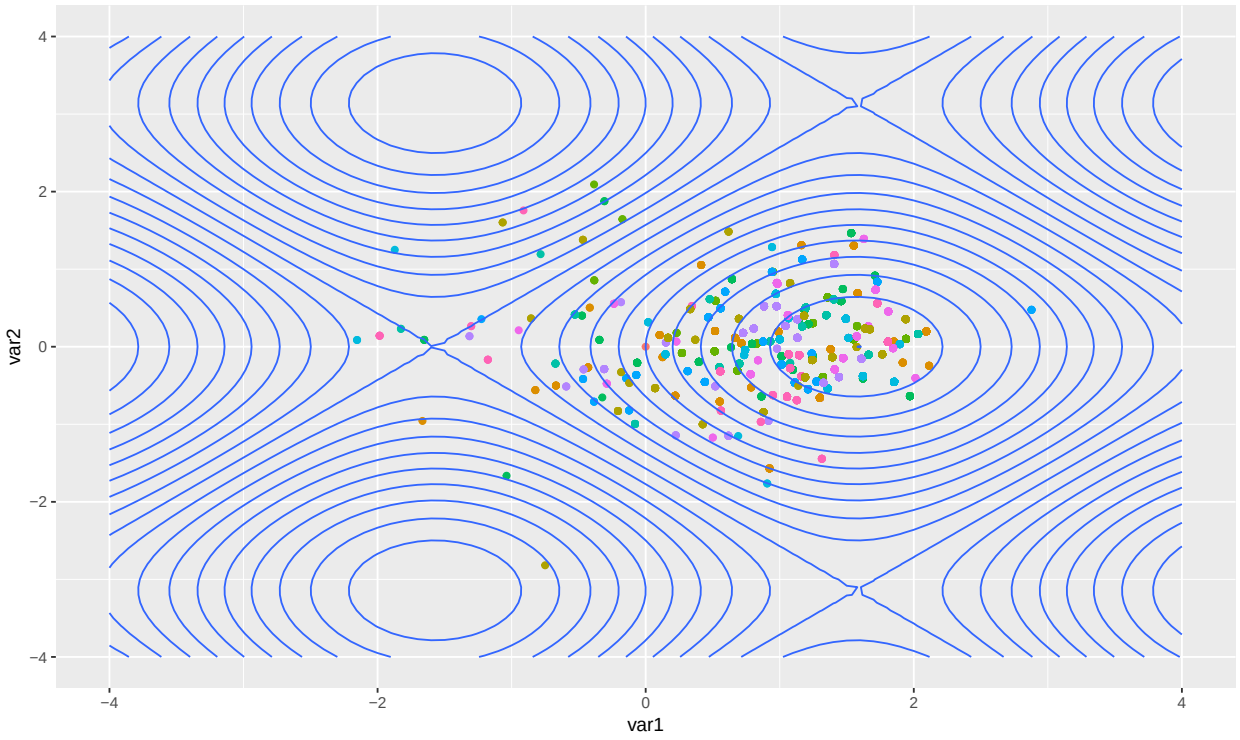

```
fitfunc <- function(x,y) {return(x^3+y^3)}
geneticPopulation(fitfunc,species=10,indivs=5,gens=100,creation="new",plot="maxVals")
```



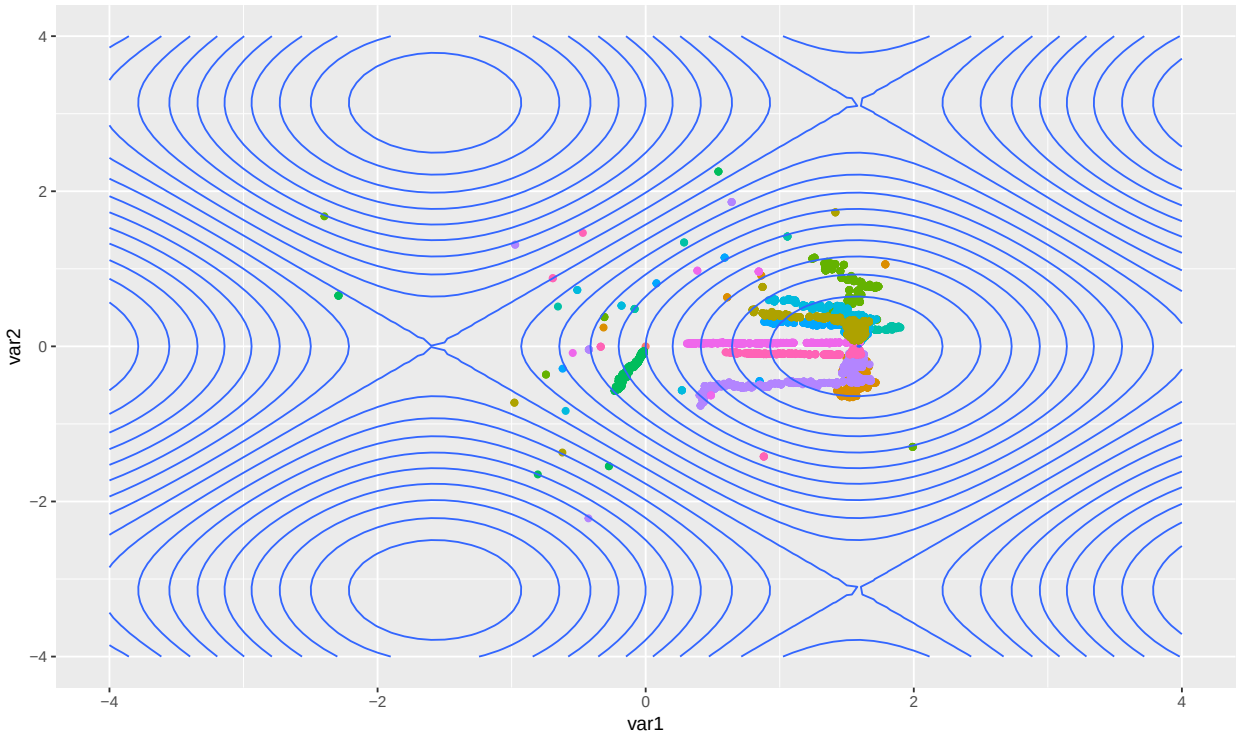

```
geneticPopulation(fitfunc,species=10,indivs=5,gens=300,creation="child",plot="maxVals")
```



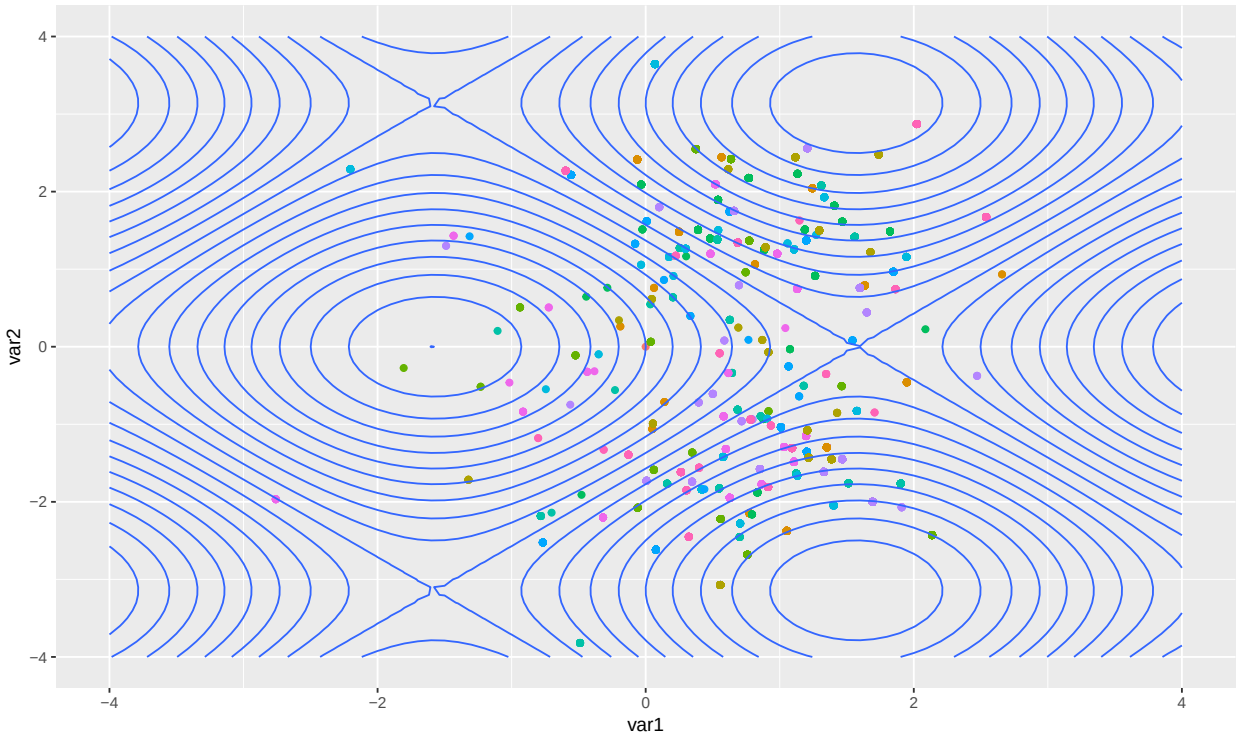

```
fitfunc <- function(x,y) {return(sin(x)+cos(y))}
geneticPopulation(fitfunc,species=10,indivs=5,gens=100,creation="new",plot="maxVals")
```



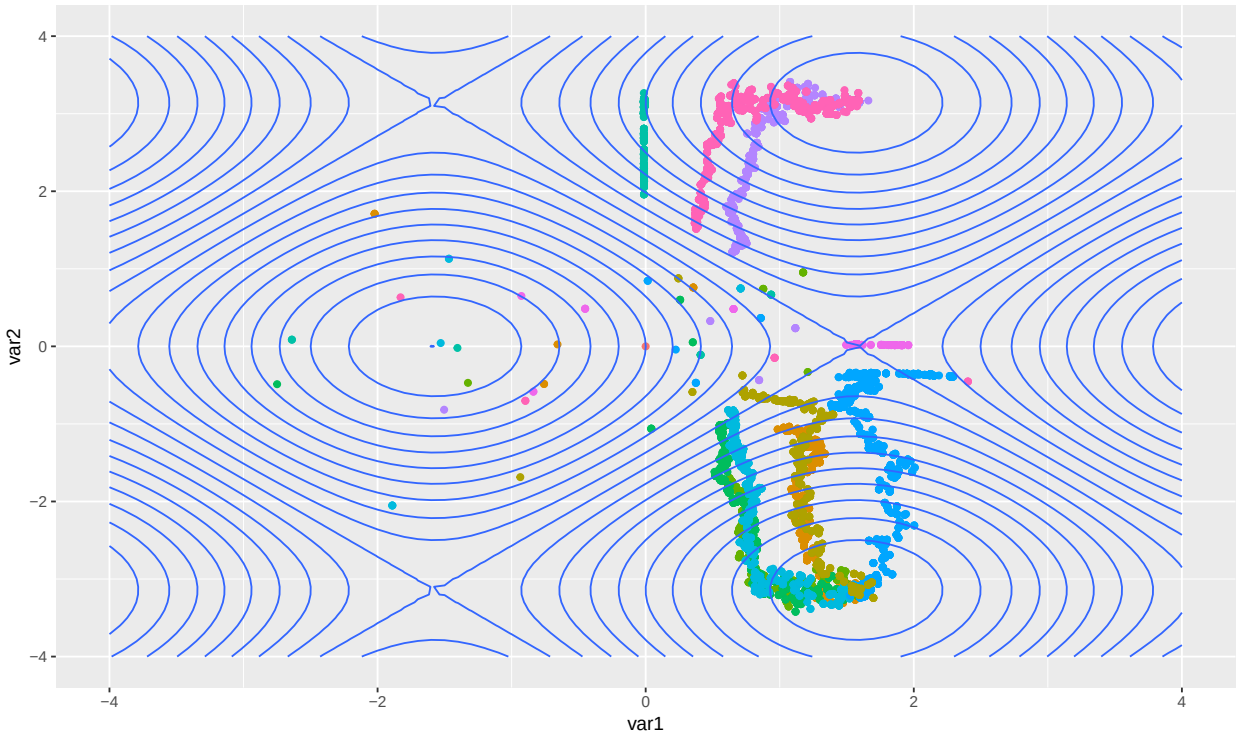

```
geneticPopulation(fitfunc,species=10,indivs=5,gens=300,creation="child",plot="maxVals")
```



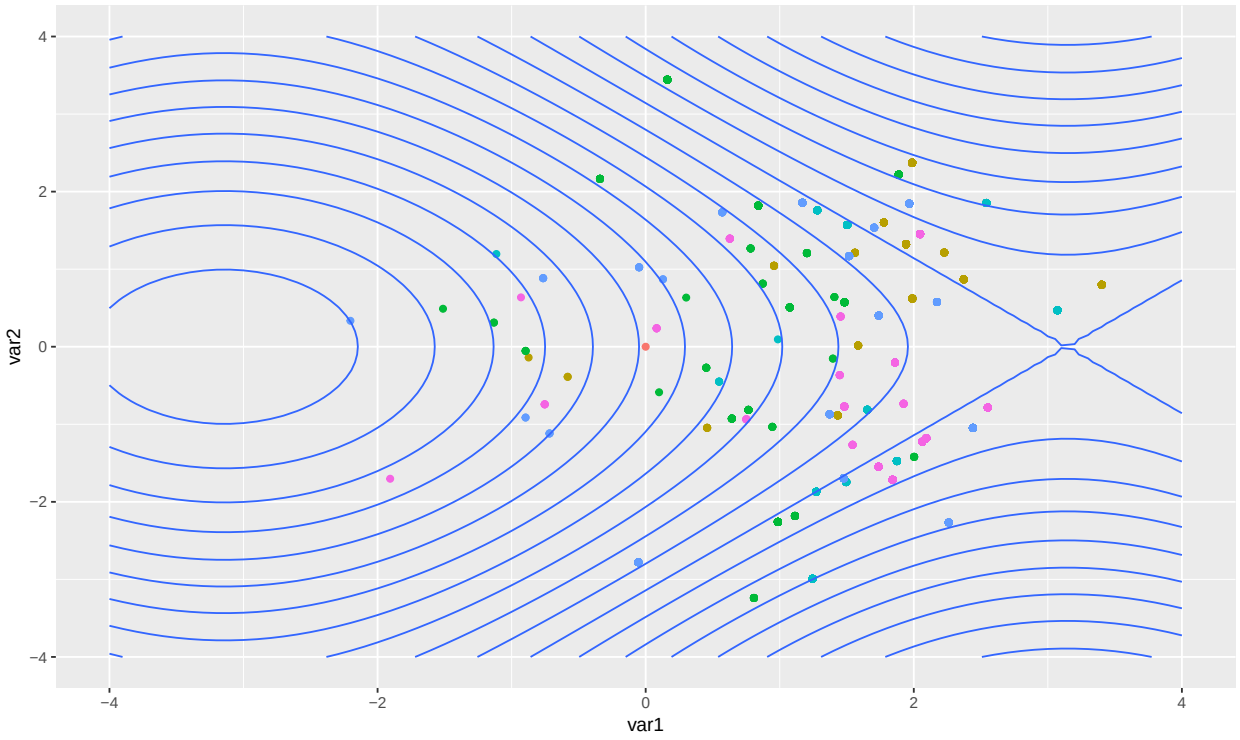

```
fitfunc <- function(x,y) {return(sin(x)-cos(y))}
geneticPopulation(fitfunc,species=10,indivs=5,gens=100,creation="new",plot="maxVals")
```



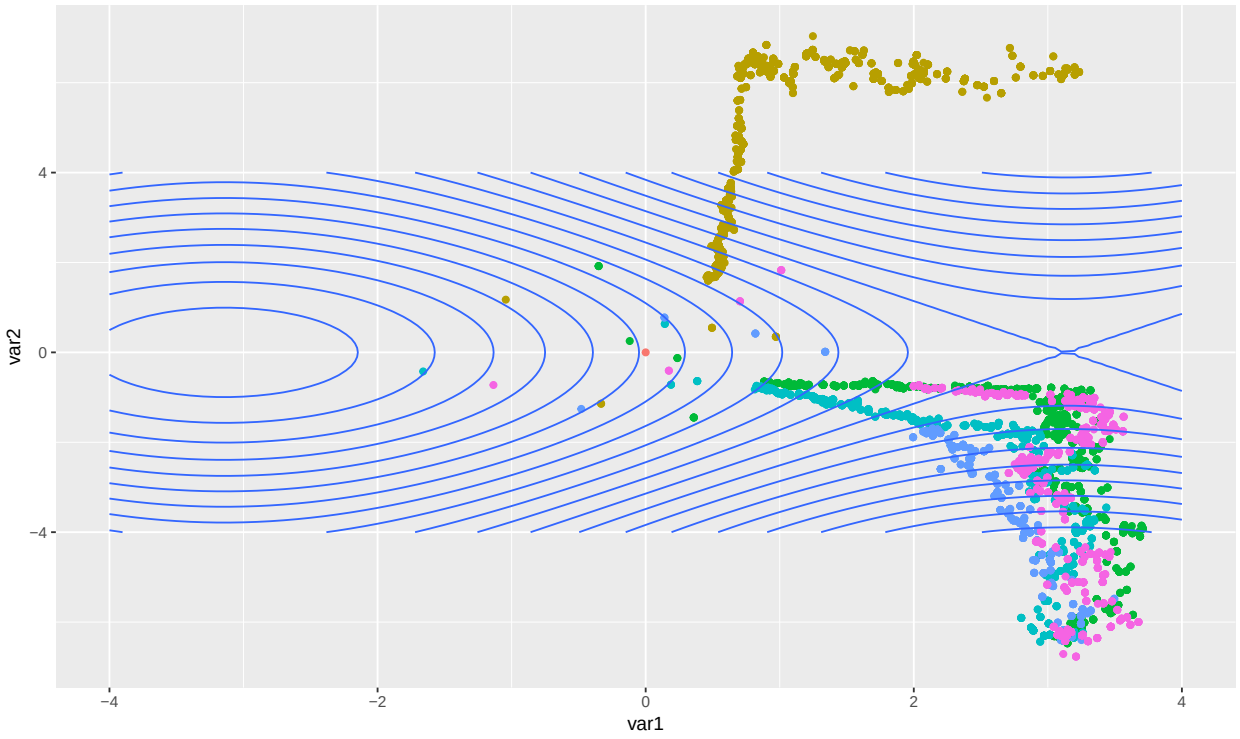

```
geneticPopulation(fitfunc,species=10,indivs=5,gens=300,creation="child",plot="maxVals")
```



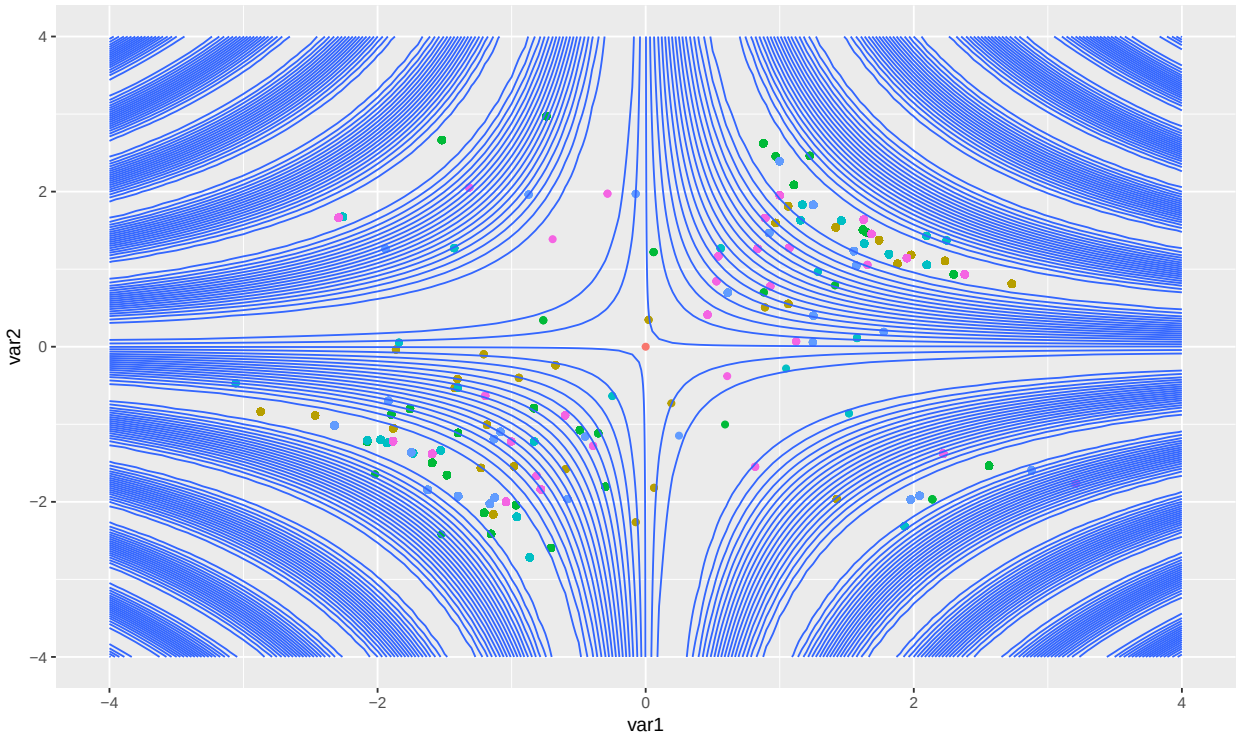

```
fitfunc <- function(x,y) {return(sin(x/2)-cos(y/2))}
geneticPopulation(fitfunc,species=5,indivs=5,gens=100,creation="new",plot="maxVals")
```



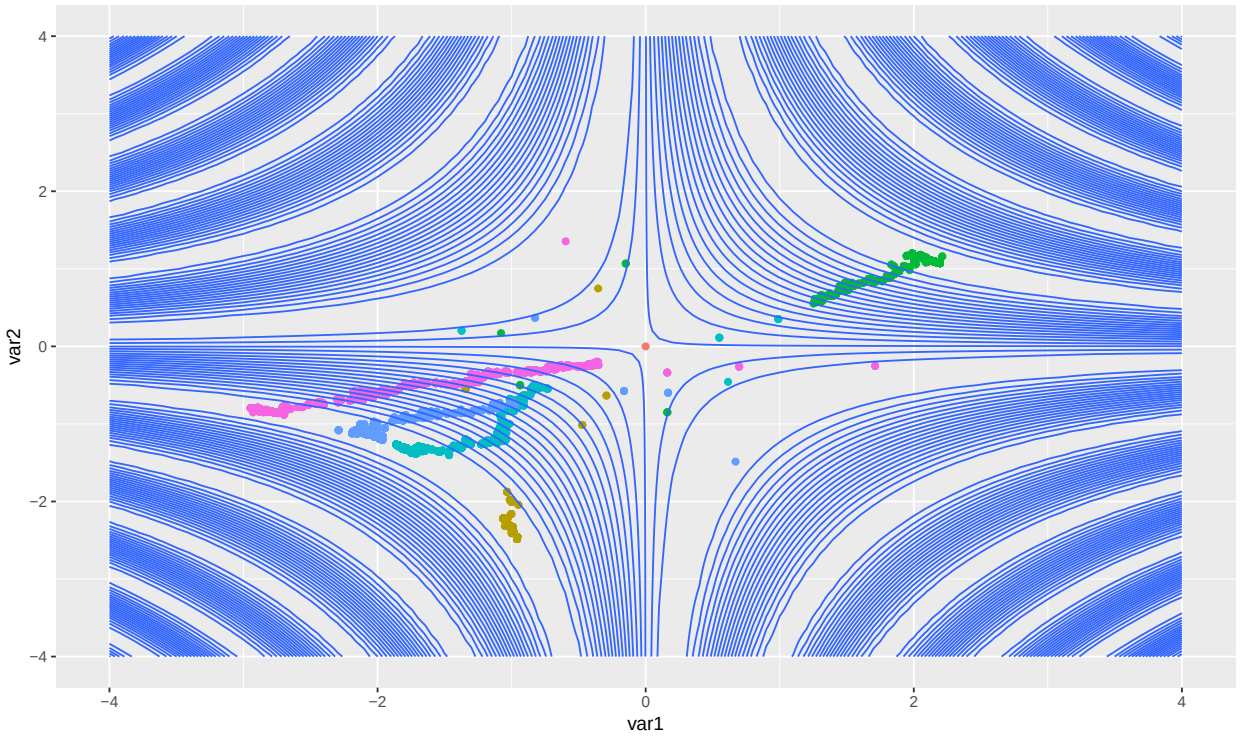

```
geneticPopulation(fitfunc,species=5,indivs=5,gens=300,creation="child",plot="maxVals")
```



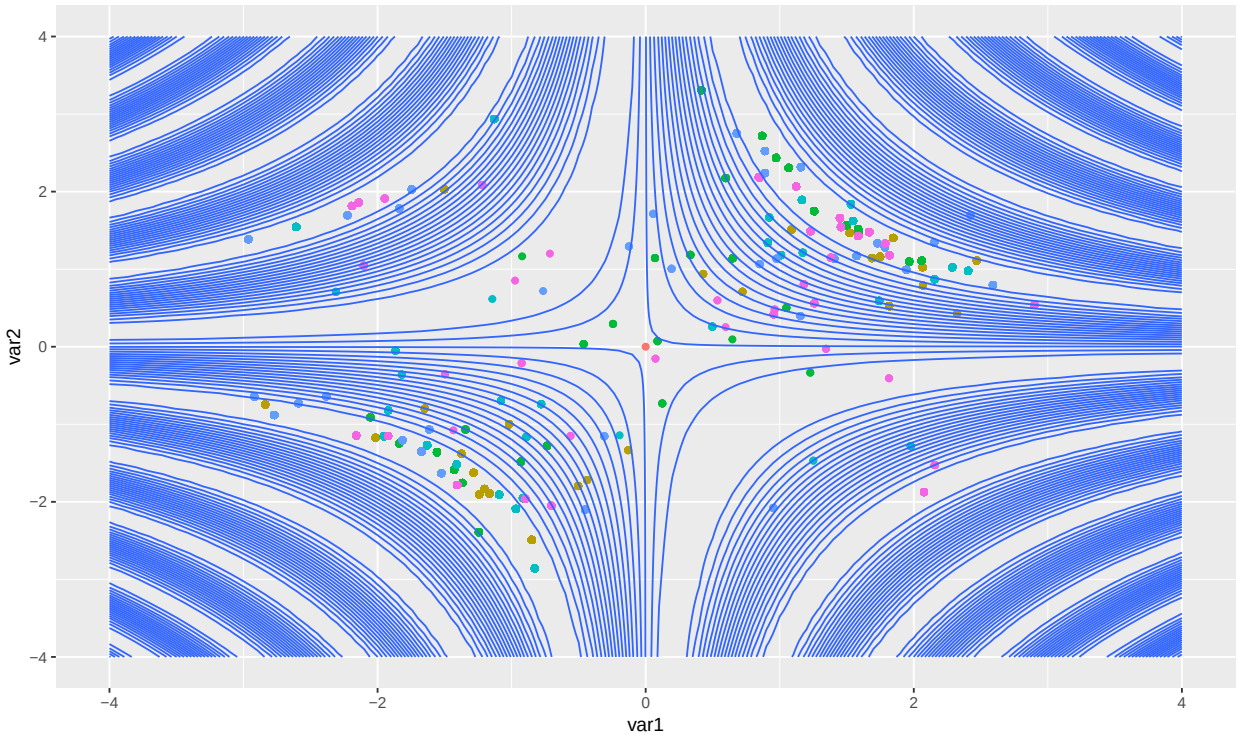

```
fitfunc <- function(x,y) {return(sin(x*y)-cos(x*y))}
geneticPopulation(fitfunc,species=5,indivs=5,gens=1000,creation="new",plot="maxVals")
```




```
geneticPopulation(fitfunc,species=5,indivs=5,gens=1000,creation="child",plot="maxVals")
```




```
fitfunc <- function(x,y) {return(sin(x*y)-cos(x*y))}
geneticPopulation(fitfunc,species=5,indivs=5,gens=1000,creation="new",plot="maxVals")
```




```
geneticPopulation(fitfunc,species=5,indivs=5,gens=1000,creation="child",plot="maxVals")
```