Lab 4: Optimization

Renato Molina updated by L. Denson

February 2019

Optimization Introduction

According to Wikipedia, optimization can be defined as: **"the selection of *a best element* (with regard to some criterion) from some set of available alternatives."**

Within the field of conservation we can think of optimization in terms of site selections. The best element in site collection can be interpreted as the "best" amount of land set aside as a reserve. The "best" element does not have to be singular, so let's refine the concept to: **the selection of the best set of elements (with regard to some criterion) from a set of alternatives.** To find that set, we need some sort of scoring mechanism that allows us to determine if we are actually choosing the best elements, when compared to the alternatives. That scoring mechanism is what we call the **objective function**.

The **objective function** is the mechanism that allows us to evaluate different elements with the same criteria. Depending on the nature of the problem, we can maximize or minimize the criteria for those elements. Typically, the set of elements we can choose from are constrained. The nature of these constraints depends on each specific problem (ex: budget caps).

Summary: the goal of optimization is to find the set of elements that maximizes or minimizes our objective function.

Optimization with R: Constrained and Unconstrained

Unconstrained Optimization

Let's work on an example where we first consider a function that evaluates the equation $z = 100 * (y - x^2)^2 + (1 - x)^2$ and takes the vector, $var.vec = \{x, y\}$ as inputs. See the code below:

```
simple.function <- function(var.vec) {
    x <- var.vec[1] # Tells R that x_1 is the first element of var.vec
    y <- var.vec[2] # Tells R that x_2 is the second element of var.vec
    100 * (y - x^2)^2 + (1 - x)^2
} # end function
```

Now we can evaluate the function for any combination of values. Let's try $x = .5$ and $y = .5$.

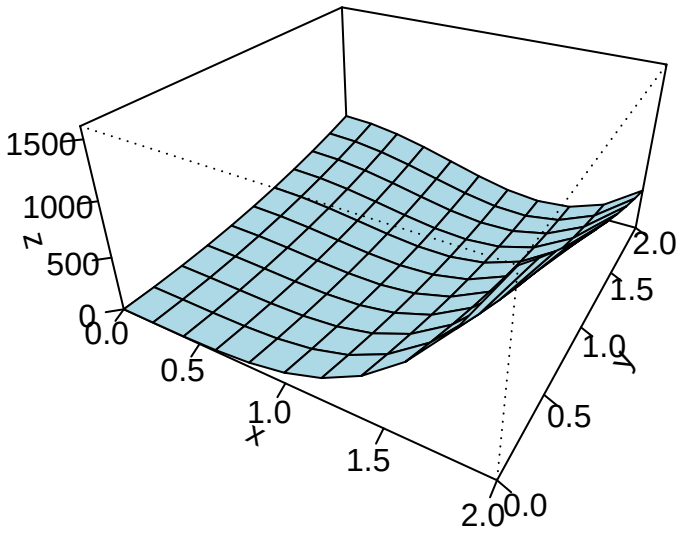
```
var.vec <- c(.5,.5) # var.vec contains both values of x

z <- simple.function(var.vec) # Evaluate the function at var.vec

z # Report results
```

```
## [1] 6.5
```

Looking at the graph of our function evaluated between 0 to 1 for both x and y we can see that this function has a global minimum of zero at $x_1 = x_2 = 1$.



If we wanted to find that minimum with R, we need a couple things:

i) an objective function (simple.function in this case)

ii) starting values (a vector $(0, 0)$)

iii) boundaries for our decision variables $(-\infty, \infty)$

iv) the method of optimization

This is an **unconstrained optimization**, the only restrictions are the upper and lower bounds of the elements that go into the objective function.

```
OO <- optim( # Store optimization output in a list called OO
      c(0,0), # Guess starting values
      simple.function, # Objective function
      upper = Inf, # Upper boundary for the decision variables
      lower = -Inf, # Lower boundary for the decision variables
      method = "L-BFGS-B") # Optimization algorithm that allows for interval specification

OO # Report output of optimization
```

```
## $par
## [1] 0.9998007 0.9996013
##
## $value
## [1] 3.973185e-08
##
## $counts
## function gradient
```

```
##       26       26
##
## $convergence
## [1] 0
##
## $message
## [1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

Note: See additional resources for explaination on differnet optimization methods. Click here, or here.

The values reported above indicate the optimal combination of x_1 and x_2 ($par) that minimize our objective function. The minimum value it achieves using those values is 3.97×10^{-8} ($value). To explore further, impose additional constraints on the values that minimize the objective function. We can extract the values of our optimal choice of variables by indexing the output element we created above:

```
var.vec <- OO$par # Extract optimal variable values

y <- simple.function(var.vec) # Evaluate the function at var.vec

y # Report results
```

```
## [1] 3.973185e-08
```

Constrained Optimization

Suppose that there are 10 lots available for purchase. We will index those lots with the letter i which is a vector from 1 to 10 , $i = \{1, 2, .., 10\}$. Each lot has some habitat potential for an endangered species, let's call it $hp_i = \{0, 1, 2\}$, meaning that lot i has habitat potential hp_i, and it could be zero, one, or two, with the latter being the highest potential. If we allow for the possibility of purchasing fractions of a lot, we can define $0 \leq F_i \leq 1$ as the fraction we purchase from lot i. It follows that $F_i = 0$ means we purchase nothing of that lot, while $F_i = 1$ means we purchase all of it. There is also a cost of purchasing each lot, which is given by c_i.

As a guideline, we know that the probability of survival of the species is equal to:

$$PS = \alpha \sqrt{\sum_{i=1}^{10} hp_i \times F_i}$$

If we want to maximize the probability of survival of the species, but we are constrained by a budget of $12 million. How much of each lot should we purchase?

To answer this question we need to set up the problem first. Let's create these lots, their habitat potential, and their cost.

```
rm(list = ls())
N <- 10 # Ten lots.

set.seed(0) # Set seed so R remembers the random numbers that we are going to generate.

hp <- sample(0:2,N,replace=T) # Habitat potential and stuff.

c <- sample(1:10,N,replace=T)  # The cost of each lot.

B <- 12 # Budget is 12 million
```

Under the assumption that purchasing all land gives probability of survival (PS) equal to one, α is a scaling factor that makes $PS = 1$ when we buy all lots in their entirety. If you purchase all lots the habitat potential

of your purchase is the sum of the habitat potential of all the plots. This is then scaled so that the probability of survival adds to one.

```
THP <- sum(hp) # Total habitat potential when all lots are purchased.

alpha <- 1/(THP)^.5 # Scaling factor so the probability adds to one.
```

Finally, the most important thing here is our policy function. As with any function, the idea is to give it some arguments and output the probability of survival. Recall that up to this point, we have two vectors or parameters, $\boldsymbol{hp}$ and $\boldsymbol{c}$ (Bold letters represent vectors), respectively; as well as our total budget, B, and the parameter α. Given any combination of purchases $\boldsymbol{F}$, our probability of survival is then given by $PS = f(\boldsymbol{F}, \boldsymbol{hp}, \boldsymbol{c}, B, \alpha)$. In R, we will now create a function that given all the information in the model, estimates the probability of survival of the species for any vector $\boldsymbol{F}$. Note that because R always minimizes, whenever we are trying to maximize something, we have to multiply the objective function by minus one.

```
surv.prob <-function(F){
  PB <- -alpha*(F%*%hp)^.5 # Probability of survival multiplied by minus one
  as.numeric(PB)
}
```

Let's look at our constraints. In our case, how much we spend on a parcel is equal to the cost of the parcel multiplied by how much we purchase from each parcel, we get the total expenditure, which has to be less than our total budget,B. We also imposed a limit on the values $\boldsymbol{F}$ can take, so now we have everything to formalize our problem mathematically:

$$\max_{\boldsymbol{F}} \left\{ PS = \alpha \sqrt{\sum_{i=1}^{10} hp_i \times F_i} \right\}$$

$$\begin{aligned} &S.T. \\ &\boldsymbol{F} \bullet \boldsymbol{c} \leq B \\ &0 \leq F_i \leq 1, \ \forall \, i = \{1, .., 10\} \end{aligned}$$

The custom function below is how we ensure that our total cost stays under our budget constraint:

```
total.cost <- function(F){ # Function that calculates the total cost, it ONLY takes F as a parameter
  TC <- F%*%c # Estimate total cost
  as.numeric(TC)}
```

All that is left for us is to solve this problem. To do so, we will install and load the `Rsolnp` package and use the constrained optimization function within it, `solnp` and start with some initial values for the fractions of the lots we plan to purchase. The general formulation is as follows:

```
library("Rsolnp")

F <- rep(1e-3,N) # Create a set of starting values and make them slightly greater than zero.


OO <- solnp(F, # Starting values
      fun = surv.prob, # Function to minimize
      ineqfun = total.cost, # Parameters for the inequality constraint
      ineqUB = B, # Upper bound of inequality
      ineqLB = 0, # Lower bound of inequality
      LB = rep(0,N), # Lower bound for decision variables, must be a vector
      UB = rep(1,N)) # Upper bound for decision variables, must be a vector as well
```

```
##
## Iter: 1 fn: -0.7638   Pars:  0.999999986120 0.000000053863 0.999999784351 0.000000035205 0.999999899
```

```
## Iter: 2 fn: -0.7638   Pars:  0.999999988902 0.000000044188 0.999999819239 0.000000028291 0.999999917
## solnp--> Completed in 2 iterations
```

```
00
```

```
## $pars
##  [1] 1.000000e+00 4.418752e-08 9.999998e-01 2.829051e-08 9.999999e-01
##  [6] 3.322778e-09 9.999998e-01 6.546218e-08 1.000653e-08 1.457522e-07
##
## $convergence
## [1] 0
##
## $values
## [1] -0.03162278 -0.76376259 -0.76376259
##
## $lagrange
##             [,1]
## [1,] -0.01841625
##
## $hessian
##               [,1]         [,2]         [,3]         [,4]        [,5]
##  [1,]  0.004569192 -0.003367869 -0.004532751 -0.004856366 -0.01934906
##  [2,] -0.003367869  0.128267313  0.008955978  0.004151112 -0.02478424
##  [3,] -0.004532751  0.008955978  0.622600277  0.049240911 -0.05346096
##  [4,] -0.004856366  0.004151112  0.049240911  0.020648153  0.06215696
##  [5,] -0.019349064 -0.024784245 -0.053460961  0.062156960  0.81277641
##  [6,] -0.008033293  0.005785045  0.113950787  0.040722215  0.13386418
##  [7,] -0.007015455  0.136436006  0.015773950  0.048160576  0.06809398
##  [8,] -0.011737331 -0.004026467  0.127362286  0.048295431  0.15824730
##  [9,] -0.020983088 -0.017105999  0.160560863  0.069472985  0.26189257
## [10,] -0.011179737  0.011026879  0.045839151  0.044712315 -0.05970706
## [11,] -0.009653805 -0.010757730  0.104386111  0.039968905  0.11686257
##               [,6]         [,7]         [,8]        [,9]        [,10]
##  [1,] -0.008033293 -0.007015455 -0.011737331 -0.02098309 -0.01117974
##  [2,]  0.005785045  0.136436006 -0.004026467 -0.01710600  0.01102688
##  [3,]  0.113950787  0.015773950  0.127362286  0.16056086  0.04583915
##  [4,]  0.040722215  0.048160576  0.048295431  0.06947299  0.04471231
##  [5,]  0.133864184  0.068093978  0.158247298  0.26189257 -0.05970706
##  [6,]  0.099128508  0.098379251  0.100885892  0.14439300  0.11325837
##  [7,]  0.098379251  0.643670013  0.102310172  0.14188200  0.22471497
##  [8,]  0.100885892  0.102310172  0.123657916  0.17682521  0.12681471
##  [9,]  0.144393002  0.141882001  0.176825209  0.27324316  0.18296669
## [10,]  0.113258372  0.224714973  0.126814712  0.18296669  0.84370916
## [11,]  0.082778234  0.088331378  0.099132447  0.14471185  0.08512002
##              [,11]
##  [1,] -0.009653805
##  [2,] -0.010757730
##  [3,]  0.104386111
##  [4,]  0.039968905
##  [5,]  0.116862571
##  [6,]  0.082778234
##  [7,]  0.088331378
##  [8,]  0.099132447
##  [9,]  0.144711852
## [10,]  0.085120016
```

```
## [11,]  0.098412383
##
## $ineqx0
## [1] 12
##
## $nfuneval
## [1] 459
##
## $outer.iter
## [1] 2
##
## $elapsed
## Time difference of 0.06000996 secs
##
## $vscale
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1
```

The interpretation of the output is the same as in the unconstrained case. $par indicates the optimal fraction of each lot that should be purchased, and $value indicates the value or the optimal probabilty of survival, obtained with those parameters. To finalize, let's make sure the computer actually did what we asked.

```
# Retrieve optimal lot purchase fractions
F_star <- OO$par

# Estimate total cost

TC_star <- F_star%*%c

TC_star
```

```
##      [,1]
## [1,]   12
```

We can see that we are right on point in terms of total cost (TC_star). The conservation performance (76% probabilty of survival) is actually quite good given that the budget is about 23% of the total cost of the lots. Obviously, some of the fractions are pretty small, but that's where your expertise comes into play when analyzing what the computer tells you. Broadly speaking, all optimization algorithms work in a similar way as to what we have shown to you here.