Lab4: Optimization

Renato Molina updated by M. Varkony

2/3/2020

Optimization Introduction

The goal of optimization is to solve for the most efficient solution using rigorous computational methods. The solution that we generate from these methods, depends on the objective of the task. In economics examples of optimization problems include, profit maximization or cost minimization. In these two examples the objectives are profit and cost respectively. There are two types of optimization, 1) constrained and 2) unconstrained optimization. For our purposes in this class we will be focusing on constrained optimization.

Within the field of conservation we can think of optimization in terms of site selections. The best element in site selection can be interpreted as the "best" amount of land set aside as a reserve. The "best" element does not have to be singular, so we can defined optimization as **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: available land or 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]
    y <- var.vec[2]
    100 * (y - x^2)^2 + (1 - x)^2
}
```

Now we can evaluate the function for any combination of values. Let's try

$$x = .5$$

and

$$y = .5$$

.

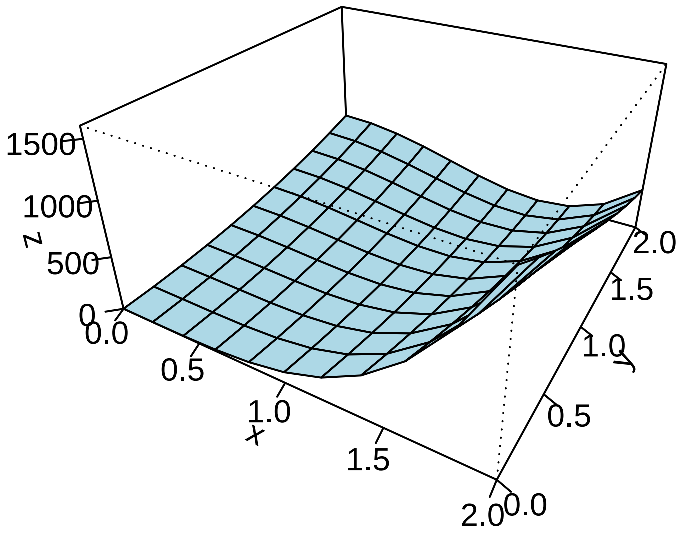
```r
var.vec <- c(.5,.5)

z <- simple.function(var.vec)

z
```

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

Looking at the graph of our function evaluated between 0 to 2 for both x and y we can see that this function appears to have a global minimum near zero at $x = y = 1$. However, eyeballing it doesn't provide much more than a guess. Therefore, a more approximate analysis would be to use R.



If we wanted to find that minimum with R, we need to define a couple of 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. We also need to decide on a function to use. There are many different optimization methods. For this class, the methods we discuss in the labs will be sufficient to solve the homework problems. However, if you are interested in learning more about optimization you can check out this website for a list on all of the different functions R employs to solve optimization problems. There are literally so many different functions that one can use, and how you decide depends on the type of problem you are faced with solving. The best method to determine the appropriate method is to look up what people who have worked on similar problems have used.

The function we use `optim` works for n-dimensional optimization (in this case we are optimizing over x, y so we would have a 2-dimensional optimization). This function is used for continuous functions, which we have so it is a good one to use.

```r
OO <- optim(
      c(0,0),
      simple.function,
      upper = Inf,
      lower = -Inf,
      method = "L-BFGS-B")

OO
```

```
## $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 We include the `upper` and `lower` bound estimates because of the method that we employ. This method `L-BFGS-B` is known as a quais-Newton method. It allows for the box constraints, which is part of the reason we use it in this situation. To learn more about optimization methods you can check out these links 1 or 2. Although there are plenty of other resources online for those interested.

The values reported above indicate the optimal combination of $[x, y]$ is [0.9998, 0.9996]. that minimize our objective function. The minimum value it achieves using those values is 0. 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. This allows us to check that work through our optimization to see that it worked appropriately.

```
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

Now we can turn to an optimization technique that we will be using specifically in this class. 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 , $i = \{1, 2, \ldots, 10\}$. 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 0, 1, or 2, with the latter being the highest habitat 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? Let's randomly select the habitat potential of each lot as well as the cost of each lot.

To answer this question we need to set up the problem first. Let's create these lots, their habitat potential, and their cost. Assume the cost of each lot is assigned randomly with values $1 \leq c_i \leq 10$ million.

```r
rm(list = ls())
set.seed(4200)

N <- 10

hp <- sample(0:2,N,replace=T)

c <- sample(1:10,N,replace=T)

B <- 12
```

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.

The breakdown of this in algebraic terms looks as follows. If $PS = 1$ and $F_i = 1$ for all $i = 1, \ldots, 10$ then we can rearrange the equation such that

$$\alpha = \frac{PS}{\sqrt{\sum_{i=1}^{10} hp_i * 1}}$$

. We then plug in our known values and get

$$\alpha = \frac{1}{\sqrt{THP}}$$

```r
THP <- sum(hp)

alpha <- 1/(THP)^.5
```

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.

```r
surv.prob <-function(F0){
  PB <- -alpha*(F0%*%hp)^.5
  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(F0){
  TC <- F0%*%c
  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. As opposed to the previous example, this is a nonlinear optimizer that uses the Lagrange optimization method. The general formulation is as follows:

```
library("Rsolnp")

F0 <- rep(1e-3,N)


OO <- solnp(F0,
      fun = surv.prob,
      ineqfun = total.cost,
      ineqUB = B,
      ineqLB = 0,
      LB = rep(0,N),
      UB = rep(1,N))
```

```
##
## Iter: 1 fn: -0.7746   Pars:  0.000000004885 0.999999968747 0.000000012795 0.499999818800 0.99999996
## Iter: 2 fn: -0.7746   Pars:  0.0000000005173 0.9999999789324 0.0000000066824 0.4999998752401 0.99999
## solnp--> Completed in 2 iterations
```

```
OO
```

```
## $pars
##  [1] 5.173093e-10 1.000000e+00 6.682357e-09 4.999999e-01 1.000000e+00
##  [6] 1.000000e+00 1.000000e+00 3.094541e-09 9.300792e-08 1.207599e-08
##
## $convergence
## [1] 0
##
## $values
## [1] -0.03162278 -0.77459666 -0.77459666
##
## $lagrange
##             [,1]
## [1,] -0.01434438
##
## $hessian
##               [,1]        [,2]         [,3]        [,4]        [,5]
##  [1,]  0.004042923 -0.01202608 -0.006430204 -0.01787308 -0.01460807
##  [2,] -0.012026079  0.99171610  0.074832912 -0.01733679  0.14652184
##  [3,] -0.006430204  0.07483291  0.552732758  0.09069304  0.06700671
##  [4,] -0.017873079 -0.01733679  0.090693039  0.93766613  0.17094008
##  [5,] -0.014608075  0.14652184  0.067006712  0.17094008  0.13045938
##  [6,] -0.006430184  0.07483326 -0.447267174  0.09069358  0.06700681
##  [7,] -0.003885791 -0.01080566  0.045195624  0.02213382  0.01745274
##  [8,] -0.004039426  0.07273250 -0.004020802  0.05559515  0.04173690
##  [9,] -0.005626639  0.05890446  0.080955625  0.04870967  0.09648400
## [10,] -0.017285368  0.17149792  0.076017241  0.19855692  0.15047774
## [11,] -0.021932334 -0.01155043  0.106117767 -0.09229901  0.19011518
##               [,6]         [,7]         [,8]         [,9]       [,10]
##  [1,] -0.006430184 -0.003885791 -0.004039426 -0.005626639 -0.01728537
##  [2,]  0.074833257 -0.010805663  0.072732496  0.058904459  0.17149792
##  [3,] -0.447267174  0.045195624 -0.004020802  0.080955625  0.07601724
##  [4,]  0.090693581  0.022133825  0.055595149  0.048709668  0.19855692
##  [5,]  0.067006805  0.017452737  0.041736902  0.096484004  0.15047774
##  [6,]  0.552732893  0.045195492 -0.004020521  0.080954763  0.07601732
##  [7,]  0.045195492  0.128951161 -0.084723218  0.074886709  0.01835192
##  [8,] -0.004020521 -0.084723218  0.200905363  0.055427904  0.04555786
##  [9,]  0.080954763  0.074886709  0.055427904  0.736863043  0.10461980
## [10,]  0.076017322  0.018351916  0.045557859  0.104619805  0.17420398
## [11,]  0.106118505  0.055238620  0.046597429  0.043904124  0.21909159
##              [,11]
##  [1,] -0.02193233
##  [2,] -0.01155043
##  [3,]  0.10611777
##  [4,] -0.09229901
##  [5,]  0.19011518
##  [6,]  0.10611850
##  [7,]  0.05523862
##  [8,]  0.04659743
##  [9,]  0.04390412
## [10,]  0.21909159
## [11,]  0.83643026
##
## $ineqx0
## [1] 12
##
## $nfuneval
## [1] 433
##
## $outer.iter
## [1] 2
##
## $elapsed
## Time difference of 0.1119869 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 probability of survival, obtained with those parameters. To finalize, let's make sure the computer actually did what we asked.

```r
# 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). 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.

Question

1. How do you calculate the total cost of the lots?
2. How do you calculate the conservation of performance?