Financial Computation: Unconstrained Optimization in Multiple Dimensions

Brian Clark

Fall 2018

Contents

1 Overview

This document describes several methods for solving optimizaiton problems in multiple dimensions. It provides some common algorithms for unconstrained multi-dimensional nonlinear optimization problems. The examples follow chapter 6 and Section 6.2 of the course textbook (Brandimarte (2013)).

2 Introduction to Nonlinear Optimization

This section describes the problem at hand. We will focus on unconstrained optimization in one dimension (i.e., choosing one variable to optimize a function). The general problem is as follows:

$$\min_{\mathbf{x}} \mathbf{F}(\mathbf{x}) \tag{1}$$

Note that we will always state the problem as a minimization problem. The solution $\mathbf{x}^*$ should satisfy the following:

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \tag{2}$$

where, $\mathbf{x}^*$ is said to be the global optimimum.

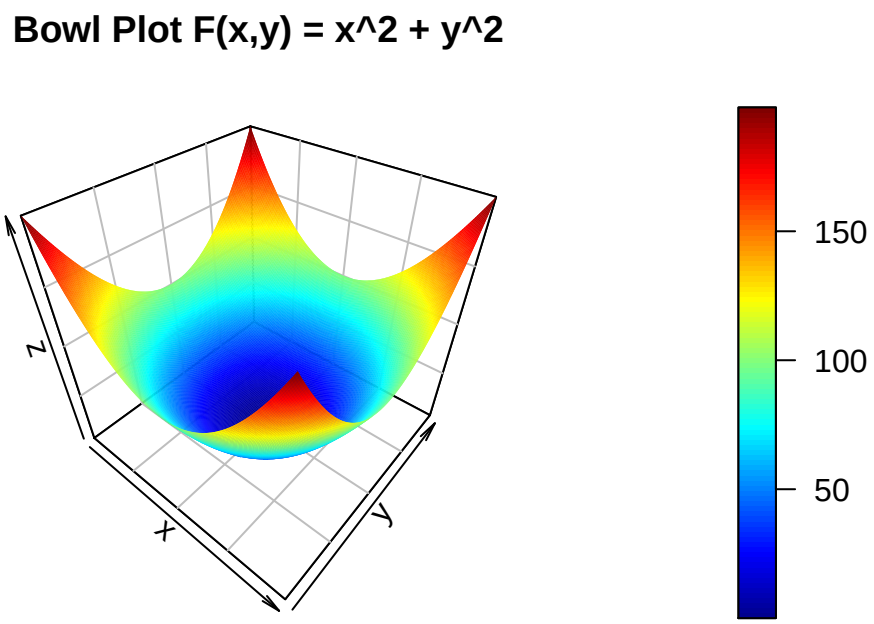
2.1 Example 1

The first example is to find the minimum of the following function:

$$F(x, y) = x^2 + y^2 \tag{3}$$

The following block of code plots the function, which is essentially a bowl shape:

```r
# Clear memory:
rm(list=ls())

# Set Working Directory:
setwd("C:/Users/CLARKB2/Documents/Classes/2018-Fall/Financial Computation/R-vignettes")
if (!require("plotly")) install.packages("plotly")
if (!require("plot3D")) install.packages("plot3D")
library("plotly")
library("plot3D")

N <- 201

Fxy <- function(x,y){
  fx <- x^2 + y^2
  return(fx)
}


xmat <- matrix(seq(-10,10,length=N), nrow = N, ncol = N, byrow=FALSE)
ymat <- matrix(seq(-10,10,length=N), nrow = N, ncol = N, byrow=TRUE)
zmat <- Fxy(xmat,ymat)

# Interactive Plot:
p <- plot_ly(x=xmat,y=ymat,z=zmat) %>% add_surface() %>%
  layout(
    title = "3D Bowl Plot",
    scene = list(
      xaxis = list(title = "x"),
      yaxis = list(title = "y"),
      zaxis = list(title = "F(x,y)")
    ))

print(p)

surf3D(x=xmat,y=ymat,z=zmat,
       bty="b2",
       main="Bowl Plot F(x,y) = x^2 + y^2")
```



Note that the goal here is to find the mimimum of the above surface, which will be $x^* = [0, 0]$. The solution is quite simple in this example as there is only one local minima that is also the global minimum. As such, you will always find the true value of x^*, regardless of your starting values.

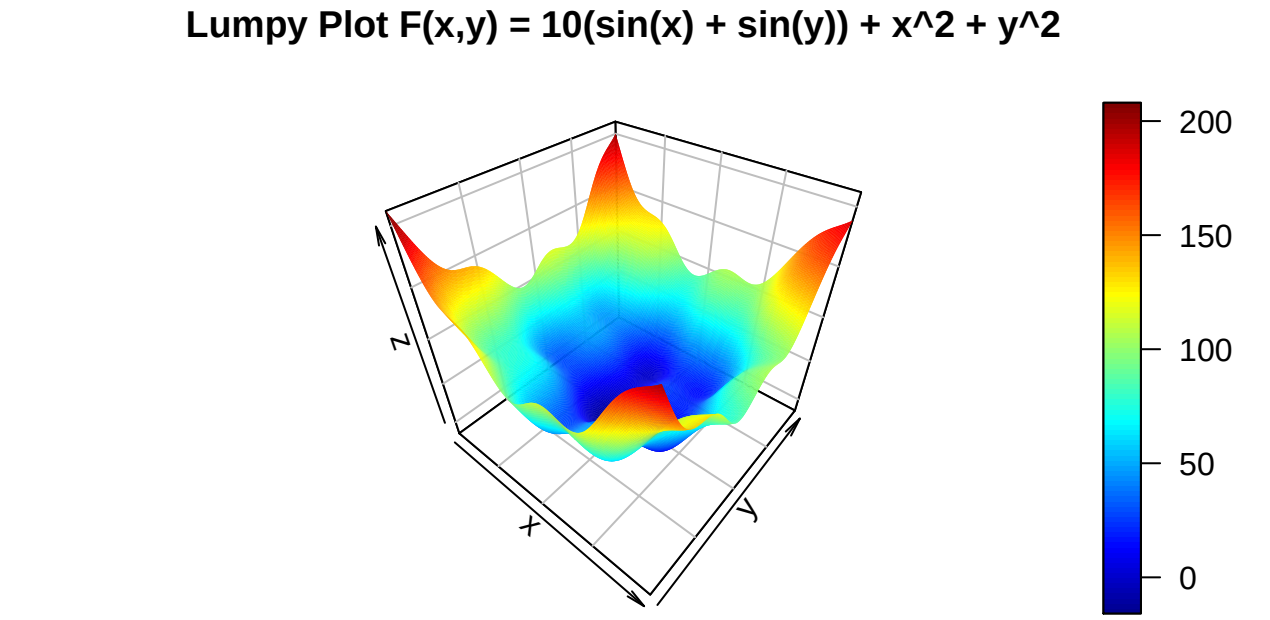
2.2 Exmaple 2

Now consider the following equation:

$$F(x, y) = 10(sin(x) + sin(y)) + x^2 + y^2 \tag{4}$$

```
N <- 201

Fxy <- function(x,y){
  fx <- 10*(sin(x) + sin(y)) + x^2 + y^2
  return(fx)
}

xmat <- matrix(seq(-10,10,length=N), nrow = N, ncol = N, byrow=FALSE)
ymat <- matrix(seq(-10,10,length=N), nrow = N, ncol = N, byrow=TRUE)
zmat <- Fxy(xmat,ymat)

# Interactive Plot:
p <- plot_ly(x=xmat,y=ymat,z=zmat) %>% add_surface() %>%
  layout(
    title = "3D Lumpy Plot",
```

```
    scene = list(
      xaxis = list(title = "x"),
      yaxis = list(title = "y"),
      zaxis = list(title = "F(x,y)")
    ))

print(p)

surf3D(x=xmat,y=ymat,z=zmat,
       bty="b2",
       main="Lumpy Plot F(x,y) = 10(sin(x) + sin(y)) + x^2 + y^2")
```



Now, in this example, there are several local minima that you may find even though there is only one gloabl minimum ($x^* = [0, 0]$).

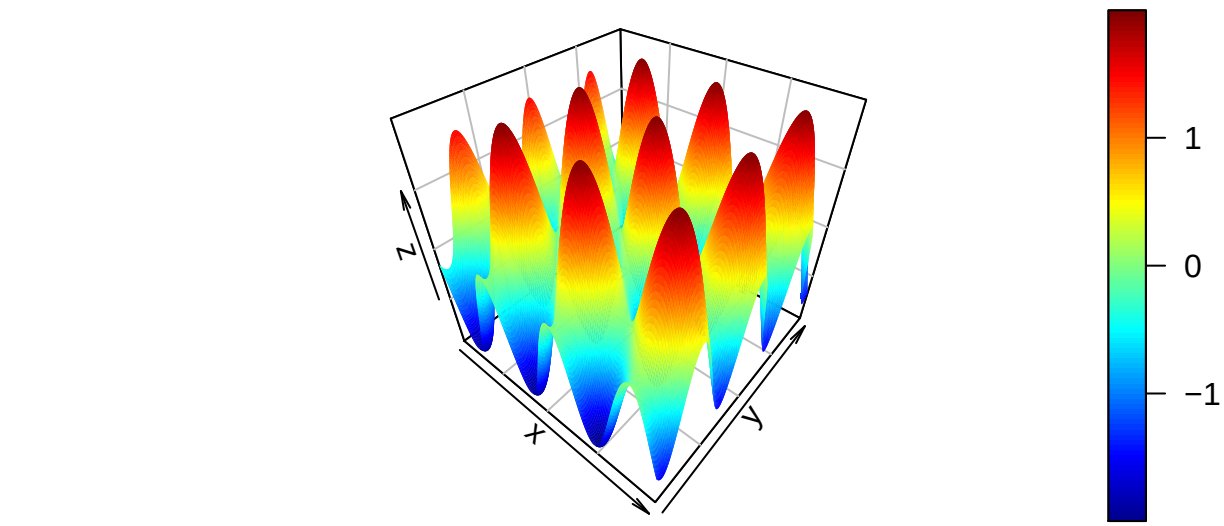
2.3 Example 3

Finally, consider a third function which has an infinite number of local minima that are all equal; meaning that there are an infinite number of solutions but no single global minumum.

$$F(x, y) = sin(x) + cos(y) \tag{5}$$

```
N <- 201

Fxy <- function(x,y){
```

```r
  fx <- (sin(x) + cos(y))
  return(fx)
}

xmat <- matrix(seq(-10,10,length=N), nrow = N, ncol = N, byrow=FALSE)
ymat <- matrix(seq(-10,10,length=N), nrow = N, ncol = N, byrow=TRUE)
zmat <- Fxy(xmat,ymat)

# Interactive Plot:
p <- plot_ly(x=xmat,y=ymat,z=zmat) %>% add_surface() %>%
  layout(
    title = "3D Egg Carton",
    scene = list(
      xaxis = list(title = "x"),
      yaxis = list(title = "y"),
      zaxis = list(title = "F(x,y)")
    ))

print(p)

surf3D(x=xmat,y=ymat,z=zmat,
       bty="b2",
       main="Egg Carton F(x,y) = sin(x) + cos(y)")
```



3 Basic Numerical Methods for Unconstrained Optimization

In principal, we can solve an unconstained optimization problem by finding a stationary point such that $\nabla f(\mathbf{x}^*) = \mathbf{0}$. Computation approaches to arrive at $\mathbf{x}^*$ are generally based on the generation of a sequence of points $\mathbf{x^{(k)}}$ that converge to $\mathbf{x}^*$.

The algorithms start at some initial point, $\mathbf{x^{(0)}}$ and search "downhill." In particular, for each $\mathbf{x^{(k)}}$, we want to move in a search direction, $\mathbf{x^{(k)}}$, such that:

$$f(\mathbf{x^{(k)}} + \alpha \mathbf{s^{(k)}}) < f(\mathbf{x^{(k)}}) \tag{6}$$

for some $\alpha > 0$.

Consider the function $h(\alpha) = f(\mathbf{x} + \alpha \mathbf{s})$ which is a descent direction characterized by the following:

$$\frac{dh}{d\alpha} = [\nabla f(\mathbf{x})]' s < 0 \tag{7}$$

which is to say that the slope is negative, i.e., we are moving toward the optimim (or at least "downhill").

A general algorithm for finding the minimum is then:

1. Find the descent direction $s^{(k)}$
2. Find a step length $\alpha^{(k)}$
3. Update $\mathbf{x^{(k+1)}} = \mathbf{x^{(k)}} + \alpha^{(k)} \mathbf{s^{(k)}}$
   Repeat steps (1-3) until some convergence criterion is met.

3.1 Steepest Descent Method:

The steepest descent method is one such method that follows the above algorithm. As the name suggests, first determine the search direction defined as follows:

$$\mathbf{s^{(k)}} = -\frac{\nabla f^(k)}{||\nabla f^(k)||} \tag{8}$$

where $\nabla f^{(k)}$ is the gradient of $f(\mathbf{x^{(k)}})$ and is defined as:

$$\nabla \mathbf{f}(\mathbf{x^{(k)}}) = \begin{bmatrix} \left(\frac{\partial f}{x_1}\right)_{\mathbf{x}=\mathbf{x^{(k)}}} \\ \left(\frac{\partial f}{x_2}\right)_{\mathbf{x}=\mathbf{x^{(k)}}} \\ \vdots \\ \left(\frac{\partial f}{x_n}\right)_{\mathbf{x}=\mathbf{x^{(k)}}} \end{bmatrix}$$

which is a vector of the partial derivaitves of $\nabla \mathbf{f}(\mathbf{x^{(k)}})$. And $||\nabla \mathbf{f}(\mathbf{x^{(k)}})||$ is the norm of the gradient.

With the search direction defined, we next define α by solving a one-dimensional optimization problem. Therefore, any of the line search algorithms discussed last week can be used to solve:

$$\min_{\alpha \geq 0} f(\mathbf{x^{(k)}} + \alpha \mathbf{s^{(k)}}) \tag{9}$$

3.2 Newton Method

A drawback of the steepest descent menthod is that it may be slow to converge because we are only using a first-order approximation. As such, we will also discuss Newton's method which is similar to the method we used to solve a system of non-linear equations. The benefit of Newton's method in this setting is that it can be implmented such that it uses a second-order approximation of the curvature of the search surface around $\mathbf{x^{(k)}}$.

The second order Taylor Series expansion can be written as follows:

$$f(\mathbf{x}+\delta) = f(\mathbf{x}) + [\nabla f(\mathbf{x})]'\delta + \frac{1}{2}\delta'\mathbf{H}(\mathbf{x})\delta \tag{10}$$

where δ is the step and $\mathbf{H}$ is the Hessian matrix of second derivatives:

$$\mathbf{H}(\mathbf{x^{(k)}}) = \begin{bmatrix} \left(\frac{\partial^2 f}{\partial x_1 \partial x_1}\right)_{\mathbf{x}=\mathbf{x^{(k)}}} & \left(\frac{\partial^2 f}{\partial x_1 \partial x_2}\right)_{\mathbf{x}=\mathbf{x^{(k)}}} & \cdots & \left(\frac{\partial^2 f}{\partial x_1 \partial x_n}\right)_{\mathbf{x}=\mathbf{x^{(k)}}} \\ \left(\frac{\partial^2 f}{\partial x_2 \partial x_1}\right)_{\mathbf{x}=\mathbf{x^{(k)}}} & \left(\frac{\partial^2 f}{\partial x_2 \partial x_2}\right)_{\mathbf{x}=\mathbf{x^{(k)}}} & \cdots & \left(\frac{\partial^2 f}{\partial x_2 \partial x_n}\right)_{\mathbf{x}=\mathbf{x^{(k)}}} \\ \vdots & \vdots & \ddots & \vdots \\ \left(\frac{\partial^2 f}{\partial x_n \partial x_1}\right)_{\mathbf{x}=\mathbf{x^{(k)}}} & \left(\frac{\partial^2 f}{\partial x_n \partial x_2}\right)_{\mathbf{x}=\mathbf{x^{(k)}}} & \cdots & \left(\frac{\partial^2 f}{\partial x_n \partial x_n}\right)_{\mathbf{x}=\mathbf{x^{(k)}}} \end{bmatrix}$$

By taking the derivative of Equation (10) and setting it equal to zero, we can solve for δ by solving the following equation:

$$\mathbf{H}(\mathbf{x})\delta = -\nabla\mathbf{f}(\mathbf{x}) \tag{11}$$

or,

$$\delta = -\mathbf{H}(\mathbf{x})^{-1}\nabla\mathbf{f}(\mathbf{x}). \tag{12}$$

And finally, the main iteration is as folows:

$$\mathbf{x^{(k+1)}} = \mathbf{x^{(k)}} + \delta \tag{13}$$

References

Brandimarte, Paolo. 2013. *Numerical Methods in Finance and Economics: A Matlab-Based Introduction.* John Wiley & Sons.