Convex Optimization: Homework 2

Martín Prado

August 31, 2024
Universidad de los Andes − Bogotá Colombia

Exercise 1.

Calculate the minimum over $\mathbb{R}^2_{++}$ of the function $f(x,y) = \frac{1}{xy} + x + y$.

Solution: The function is differentiable on $\mathbb{R}^2 \backslash \{0\}$. Therefore, we have derivatives in $\mathbb{R}^2_{++}$.

$$Df = \begin{pmatrix} 1 - x^{-2}y^{-1} \\ 1 - x^{-1}y^{-2} \end{pmatrix}.$$

Critical points are found when

$$\begin{array}{rc} & x^{-2} = y \\ & y^{-2} = x \\ \hline \implies & x^4 = x \\ \iff & x^4 - x = 0 \end{array}$$

and since $x, y \in \mathbb{R}^2_{++}$, the only possible solution for this is when $x = y = 1$. Therefore, the only critical point is $(1,1)$. Now, the Hessian matrix is the following,

$$Hf(x,y) = \begin{bmatrix} \frac{2}{x^3 y} & \frac{1}{x^2 y^2} \\ \frac{1}{x^2 y^2} & \frac{2}{xy^3} \end{bmatrix}$$

Then,

$$Hf(1,1) = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix},$$

has positive eigenvalues $\lambda_1 = 3,\ \lambda_2 = 1$. Therefore, $(1,1)$ is a minimum on $\mathbb{R}^2_{++}$, which evaluates $f(1,1) = 3$.

Exercise 2.

Güler: Ex 2.9

Consider the function $f(x,y) = x^2 + \alpha xy^2 + 2y^4$. Show that all the parameter values except two, the origin $(0,0)$ is the only critical point of f.

a) Find the exceptional α's and show that f has infinitely many critical points for these α values. Determine the nature of these critical points.

b) Consider the values of α's for which the origin is the only critical point. For each α, determine the nature of the critical point. Show that in some cases, the origin is a local minimum, but in other cases, it is a saddle point.

c) Show that even when the origin is a saddle point, (0,0) is a local strict minimizer of f on every line passing through the origin. In fact, show that, except for one line, the function $g(t) := f(td)$ satisfies $g'(0) = 0$ and $g''(0) > 0$.

Solution Part (a)

The derivative of f is

$$Df(x,y) = \begin{pmatrix} 2x + \alpha y^2 \\ 2\alpha xy + 8y^3 \end{pmatrix}.$$

By solving α in $Df(x,y) = 0$ we can find where f has infinitely many critical points. Then, we get the following equations, for $x, y \neq 0$:

$$\begin{aligned} \alpha &= \frac{-2x}{y^2} \\ \alpha &= \frac{-8y^3}{2xy} \end{aligned} \iff \frac{-2x}{y^2} = \frac{-8y^3}{2xy}$$

$$\iff x^2 = 2y^4$$

$$\iff y^2 = \pm\frac{x}{\sqrt{2}}.$$

Therefore, the only parameter values where there are infinitely many solutions are:

$$\alpha = \frac{-2x}{y^2} = \frac{-2x}{\pm x \cdot \sqrt{1/2}} = \pm 2\sqrt{2}.$$

When α is different from these values, the chain of equivalences we made before tells us that $Df(x,y) \neq 0$ when $x, y \neq 0$. Since $Df(0,0) = 0$ for any α, it follows that $(0,0)$ is the only critical point whenever $\alpha \neq \pm 2\sqrt{2}$.

Solution Part (b)

The Hessian matrix is degenerate at (0,0), so I looked at Geogebra for a hint of the answer, it seems that when $\alpha \in [-2\sqrt{2}, 2\sqrt{2}]$, it is a minimum and a saddle point at $\alpha \in (-\infty, -2\sqrt{2}) \cup (2\sqrt{2}, \infty)$.

$$\begin{bmatrix} 2 & 2\alpha y \\ 2\alpha y & 2\alpha x + 24y^2 \end{bmatrix}\Big|_{(0,0)} = \begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix}$$

To justify that, we must complete a square in the previous expression to show that for every $(x, y) \neq (0, 0)$ the function doesn't decrease:

$$\begin{aligned} x^2 + \alpha x y^2 + 2y^4 &= \left(x^2 + \alpha x y^2 + \frac{1}{4}\alpha^2 y^4\right) - \frac{1}{4}\alpha^2 y^4 + 2y^4 \\ &= \left(x + \frac{\alpha y^2}{2}\right)^2 + y^4\left(2 - \frac{\alpha^2}{4}\right) \end{aligned}$$

If $|\alpha| \leq 2\sqrt{2}$, then $\frac{\alpha^2}{4} \leq 2$. Therefore,

$$y^4\left(2 - \frac{\alpha^2}{4}\right) \geq 0,$$

and if $x, y \neq 0$, then

$$\left(x + \frac{\alpha y^2}{2}\right)^2 \geq 0.$$

It follows that $(0, 0)$ is a global minimum, since every slight deviation from the origin will make f stay the same or increase. Now, for the case when $|\alpha| > 2\sqrt{2}$ the oposite happens when we consider the following curve,

$$x(t) = -\frac{\alpha t^2}{2}, \quad y(t) = t.$$

If $|\alpha| > 2\sqrt{2}$, then $\frac{\alpha^2}{4} > 2$. Therefore, when $t \neq 0$

$$t^4\left(2 - \frac{\alpha^2}{4}\right) < 0,$$

and

$$\left(x(t) + \frac{\alpha y(t)^2}{2}\right)^2 = \left(-\frac{\alpha t^2}{2} + \frac{\alpha t^2}{2}\right)^2 = 0.$$

Therefore, function evaluated on the curve $(x(t), y(t))$ decreases when $t \neq 0$.

Solution Part (c)

Let $d = (x_0, y_0)$. Then

$$g(t) = f(tx_0, ty_0) = t^2 x_0^2 + \alpha t^3 x_0 y_0^2 + 2t^4 y_0^4,$$

$$g'(t) = 2t x_0^2 + 3\alpha t^2 x_0 y_0^2 + 8t^3 y_0^4,$$

$$g''(t) = 2x_0^2 + 6\alpha t x_0 y_0^2 + 24 t^2 y_0^4.$$

It's clear that $g'(0) = 0$ for every $d \in \mathbb{R}^2 \backslash \{\vec{0}\}$ and $g''(0) = 2x_0^2 > 0$ whenever $x_0 \neq 0$. Therefore, we know for sure that $t = 0$ minimizes g in every line except for $(x(t), y(t)) = (0, t)$.

Exercise 3.

Nocedal & Wright: Ex 3.1

Program the steepest descent and Newton algorithms using the backtracking line search, Algorithm 3.1. Use them to minimize the Rosenbrock function (2.22). Set the initial step length $\alpha = 1$ and print the step length used by each method at each iteration. First try the initial point $x_0 = (1.2, 1.2)^T$ and then the more difficult starting point $x_0 = (-1.2, 1)^T$.

Solution:

The following packages are going to be used in this exercise:

```
import numpy as np
import sympy as sp
import matplotlib.pyplot as plt
from sympy.tensor.array import derive_by_array
from numpy.linalg import inv, norm
```

The package `numpy` is for the management of arrays, matrices and linear algebra. The package `sympy` is used in the calculation of the symbolic derivatives of f. The package `matplotlib.pyplot` is for the final comparison graphs. In the following steps I'm going to define the function and automatically calculate its derivatives.

$$f(x, y) = 100(y - x^2)^2 + (1 - x)^2$$

```
dim = 2
x = sp.symbols("".join([f"x_{i+1} " for i in range(dim) ])[:-1])
f = lambda x : 100*(x[1]-x[0]**2)**2 + (1-x[0])**2
D = lambda expr: derive_by_array(expr,x)
Df = sp.lambdify([x],D(f(x)))
Hf = sp.lambdify([x],D(D(f(x))))
```

Now, for the algorithm. We choose x_0 before starting the process. Then,

$$\begin{aligned} f_k &= f(x_k), & Df_k &= Df(x_k), \\ p_k &= -B_k^{-1} Df_k, & x_{k+1} &= x_k + \alpha_k p_k, \end{aligned}$$

where $B_k = Hf(x_k)$ in Newton's method and $B_k = I$ in the steepest descent method. α_k is chosen according to the backtracking method,

$$\alpha_k = \max \left\{ \rho^m \alpha \ : \ f(x_k + \rho^m \alpha p_k) < f(x_k) + \gamma \rho^m Df(x_k)^T p_k \right\}.$$

This is the code corresponding to the backtracking method,

```
def backtracking(alpha, gamma, rho, x_k, f_k, Df_k, p_k):
alpha_k = alpha
backtracking = f(x_k + alpha_k*p_k) - (f_k + gamma*alpha_k*Df_k.T @ p_k)
while backtracking > 0 :
    alpha_k = alpha_k * rho
    backtracking = f(x_k + alpha_k*p_k) - (f_k + gamma*alpha_k*Df_k.T @ p_k)
return alpha_k
```

Also, for the Newton's method and steepest descent, the code is the following

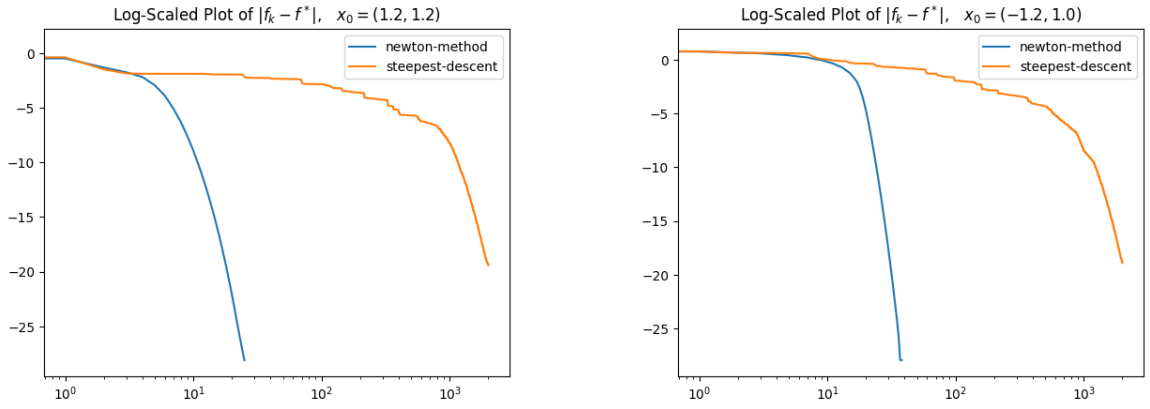
```
def newton(alpha, gamma, rho, n, x_0, f, Df, Hf):
    x_k = x_0
    x_list = [x_k]
    for i in range(n):
        f_k = f(x_k)
        Df_k = Df(x_k)
        B_k = Hf(x_k)
        p_k = -inv(B_k)@Df_k
        alpha_k = backtracking(alpha, gamma, rho, x_k, f_k, Df_k, p_k)
        x_k = x_k+alpha_k*p_k
        x_list.append(x_k)
        if norm(Df_k) < epsilon:
            break
    return x_list

def gradient_descent(alpha, gamma, rho, n, x_0, f, Df, Hf):
    x_k = x_0
    x_list = [x_k]
    for i in range(n):
        f_k = f(x_k)
        Df_k = Df(x_k)
        p_k = -Df_k
        alpha_k = backtracking(alpha, gamma, rho, x_k, f_k, Df_k, p_k)
        x_k = x_k+alpha_k*p_k
        x_list.append(x_k)
        if norm(Df_k) < epsilon:
            break

    return x_list
```

The parameters I'm going to use in the algorithm are defined as follows,

```
epsilon = 1e-12
n= 2000
alpha = 1
rho = 0.94
gamma = 0.6
x_0 = np.array([1.2,1.2])
```

$\varepsilon =$ `epsilon` is the termination criterion number for which the algorithm stops when $\|Df(x_k)\| < \varepsilon$. $n =$ `n` is the maximum number of iterations I want for the algorithm.

(α, ρ, γ) = `(alpha, rho, gamma)` are the parameters for the backtracking line search and x_0 = `x_0` is the initial position. The results are the following for $x_0 = (1.2, 1.2)$ and $x_0 = (-1.2, 1.0)$ respectively.



Finally, I the step sizes are located in the files `hw3-newton-stepsizes.txt` and

`hw3-steepest-stepsizes.txt` respectively.

Exercise 4.

Boyd & Vanderberghe: Ex 9.2

Minimizing a quadratic-over-linear fractional function. Consider the problem of minimizing the function $f : \mathbb{R}^n \to \mathbb{R}$ defined as

$$f(x) = \frac{\|Ax - b\|_2^2}{c^T x + d}, \qquad \text{dom}(f) = \{x \ : \ c^T x + d > 0\}.$$

We assume rank$A$

Exercise 5.

Consider the function

$$f(x) = \langle c, x \rangle - \sum_{j=1}^{m} \log(1 - \langle a_j, x \rangle) - \sum_{i=1}^{n} \log(1 - x_i^2).$$

Take $n = 5000$ and $m = 2000$, and vectors c, a_j chosen arbitrarily such that $\|a_j\| = 1$. For every method initiate the iterations in $x_0 = 0$ and the termination criterion in the form $\|\nabla f(x_k)\| \leq \varepsilon$. Graph $\log(f(x_k) - f^*)$ on each iteration (estimating the best possible f^*) and indicate the total time each method takes.

a) Use the gradient method with constant step size.

b) Use the gradient method with backtracking for different parameter values.

c) Use the gradient method with backtracking and some Goldstein of Wolfe conditions with different parameter values.

d) Use Newton's method

Solution:

With the objective of hasting the execution of the functions (especially the hessian), I had to vectorize them. We're going to define some elementwise functions for matrices. Some of them are known as the Hadamard products

$$\odot : \mathbb{R}^{m\times n} \times \mathbb{R}^{m\times n} \to \mathbb{R}^{m\times n}, \quad (A \odot B)_{ij} = A_{ij} \cdot B_{ij},$$

$$\oslash : \mathbb{R}^{m\times n} \times \mathbb{R}^{m\times n} \to \mathbb{R}^{m\times n}, \quad (A \oslash B)_{ij} = \frac{A_{ij}}{B_{ij}},$$

$$\square^{\circ k} : \mathbb{R}^{m\times n} \to \mathbb{R}^{m\times n}, \quad (A^{\circ k})_{ij} = A_{ij}^k,$$

$$\log_\circ : \mathbb{R}_+^{m\times n} \to \mathbb{R}^{m\times n}, \quad (\log_\circ(A))_{ij} = [\log(A_{ij})]_{ij}.$$

$$\operatorname{diag} : \mathbb{R}^n \to \mathbb{R}^{n\times n}, \quad \operatorname{diag}(x) = \begin{bmatrix} x_1 & & \\ & \ddots & \\ & & x_n \end{bmatrix}.$$

Now, we can express f in terms of the previous operations,

$$f(x) = f_1(x) + f_2(x) + f_3(x),$$

$$\begin{aligned} f_1(x) &= c^T x, \\ f_2(x) &= -\mathbb{1}_m^T \log_\circ(\mathbb{1}_m - A^T x), \\ f_3(x) &= -\mathbb{1}_n^T \log_\circ(\mathbb{1}_n - x^{\circ 2}). \end{aligned}$$

$$\begin{aligned} Df_1(x) &= c, & Hf_1(x) &= 0_{n\times n}, \\ Df_2(x) &= -A(\mathbb{1}_m - A^T x)^{\circ -1}, & Hf_2(x) &= -(A \odot (\mathbb{1}_m - A^T x)^{\circ -2})A^T, \\ Df_3(x) &= -(2x) \oslash (1 - x^{\circ 2}). & Hf_3(x) &= -\operatorname{diag}((2x^{\circ 2} + 2) \oslash (1 - x^{\circ 2})^{\circ 2}). \end{aligned}$$

For the previous operations, $A = [a_1|\cdots|a_m] \in \mathbb{R}^{n\times m}$ is the matrix with vectors a_j on the columns, $\mathbb{1}_m \in \mathbb{R}^m$, $\mathbb{1}_n \in \mathbb{R}^n$ are vectors with ones for entries. It can be shown that all the previous functions coincide with the functions required by the exercise, but it's not in the scope of this solution.

Now, regarding the code, I'm using the same framework I used for exercise 3. The functions are defined as follows,

```
n = 5000
m = 2000
c = np.random.normal(0,1,n)
a = np.random.normal(0,1,(n,m))
a = a/norm(a.T, axis = 1) # Uniformly distributed over S^{m-1}

f = lambda x: c.T@ x - sum(np.log(1-a.T@ x) - sum(np.log(1-x**2)))
Df = lambda x: c - a@(1/(1-a.T@ x))  - (2*x)/(1-x**2)
Hf = lambda x: 0 - a*(1/(1-a.T@ x)**2)@ a.T - np.diag(2*(x**2 + 1)/(1-x**2)**2)
```

However, I had to make a slight modification to the methods because f is not defined for every $x \in \mathbb{R}^n$. That is because

$$\mathrm{dom}(f_2) = \left\{ x \in \mathbb{R}^n \ : \ \underbrace{A^T x < \mathbb{1}_m}_{\text{simplex}} \right\}$$

$$\mathrm{dom}(f_3) = \left\{ x \in \mathbb{R}^n \ : \ \underbrace{\|x\|_\infty^2 = \max(x_i^2) < \mathbb{1}_m}_{\text{open square}} \right\}.$$

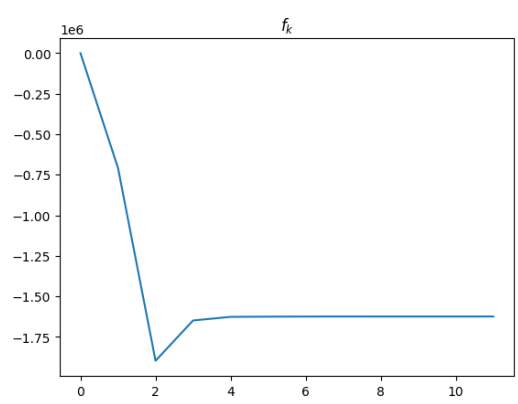
Furthermore, I believe the the further failure of the implementations presented here are due to the fact that the optimization here is constrained.

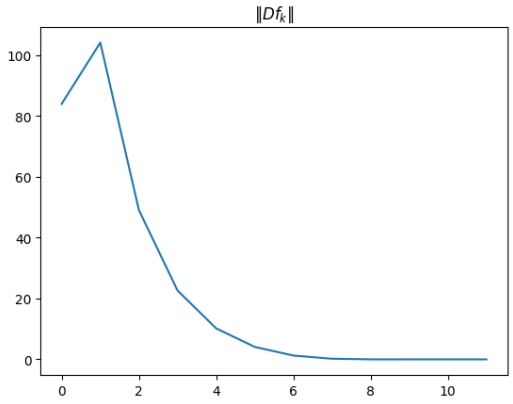
Newton's method

For the Newton's method I used almost the same implementation. But the initial α_k is chosen in such way that $f(x_k + \alpha_k p_k)$ is defined.

```
def newton(alpha, gamma, rho, iterations, x_0, f, Df, Hf):
    ...
    i = 0
    while not is_in_domain(x_k_temp) and i < 200:
        alpha_k *= rho
        x_k_temp = x_k+alpha_k*p_k
        i += 1
    print(is_in_domain(x_k_temp))

    if not is_in_domain(x_k_temp):
        break
    alpha_k = backtracking(alpha_k, gamma, rho, x_k, f_k, Df_k, p_k)
    ...
```

The results were dissapointing though. $\|Df_k\| < \varepsilon$ only happened when $\varepsilon = 4.2$ (I believe due to accumulations in computation errors), but the method converged eventually to some number (which isn't the minimum of the function). The following graphics show the results of Newton's method.





For the other methods the results weren't any better. For constant step size gradient descent, x_k falls of the domain every time with many combinations of parameters. For gradient descent with backtracking $\|Df_k\|$ increases too fast and also falls of the domain. I used all the time I had left trying to analice what went wrong with the implementation so I didn't implement Goldstein or Wolfe conditions.