TOML - First project

Oriol Martínez Acón

March 2022

1 Introduction

The idea of this project is to identify the convexity of the problems and how to solve them using the programming language Python.

To solve the exercises we need to use the packages as **scipy** and **cvxpy**. This packages provides the minimum functions to solve convex/concave problems.

All the code developed for each of the exercises can be found in the following GitHub repository: `https://github.com/oriolmartinezac/TOML-Labs/tree/main/project-1`.

Before trying to solve any problem first is necessary to analyze it. It is essential to know what are the variables, functions and constraints to verify if the problem we are trying to solve is either convex or concave.

Before entering in more detail with a problem is convex or not we first need to know what are **Hessian matrix**, **Gradient**, and **Jacobian matrix**.

The **Hessian matrix** ($H(f)$) is the matrix that contains the second derivatives (f'') of a function given, The way to represent the Hessian matrix is the following one.

$$H(f) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

The **Gradient** (∇f) is the vector field whose values at a point p is the vector whose components are the partial derivatives of f (f').

$$\nabla f(x_1, x_2, \ldots, x_n) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(x_1, x_2, \ldots, x_n) \\ \frac{\partial f}{\partial x_2}(x_1, x_2, \ldots, x_n) \\ \vdots \\ \frac{\partial f}{\partial x_n}(x_1, x_2, \ldots, x_n) \end{bmatrix}$$

The **Jacobian matrix** is the matrix of all its first-order partial derivatives (f').

$$\mathbb{J} = \begin{bmatrix} \dfrac{\partial \mathbf{f}(\mathbf{x})}{\partial x_1} & \cdots & \dfrac{\partial \mathbf{f}(\mathbf{x})}{\partial x_n} \end{bmatrix}$$

So we can say also that the Jacobian is equal to the Gradient of the function.

$$\mathbb{J} = \begin{bmatrix} \dfrac{\partial \mathbf{f}(\mathbf{x})}{\partial x_1} & \cdots & \dfrac{\partial \mathbf{f}(\mathbf{x})}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \nabla^T f_1(\mathbf{x}) \\ \vdots \\ \nabla^T f_m(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f_1(\mathbf{x})}{\partial x_1} & \cdots & \dfrac{\partial f_1(\mathbf{x})}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m(\mathbf{x})}{\partial x_1} & \cdots & \dfrac{\partial f_m(\mathbf{x})}{\partial x_n} \end{bmatrix}$$

Now that **Hessian matrix**, **Gradient** and **Jacobian matrix** is shown, we can continue by verifying if a problem is convex. The way to do it is by fulfilling some features/conditions that are shown below:

1. **Domain of the function**: The way to verify a domain is convex, is to check if any linear combination of two points inside the function could have a representative value in the existing set.

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda) f(x_2)$$

2. **First-order condition**:

$$f(y) \geq f(x) + \nabla f(x)(y-x)$$

for $x \in \mathbb{R}^n, b \in \mathbb{R}^m, A \in \mathbb{R}^{mxn}$.

3. **Second-order condition**: To fulfill the second-order condition, the Hessian matrix of the functions has to be positive semi-definite ($H \geq 0$). If this condition is fulfilled we can say the functions is convex. It is also important to remark that when the Hessian is equal to 0, it means that we have a flat plane (convex but not strictly convex).

So, one function will be convex if all the supporting hyper-planes are below the function given if it is the opposite is concave and if above/below could be convex and concave. Is important to highlight that there are more ways to verify a function is convex, but the ones that are explaining more or are more easy to understand are those.

Another important concept is the **Lagrangian** of an optimization problem. The **Lagrangian** can be defined mathematically as for an optimization problem that is not necessary restricted as a convex problem:

$$L(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{m} \nu_i h_i(x)$$

The parameter λ_i refers to the Lagrange multiplier from the inequality constraints and the ν_i the Lagrange multiplier from the equality constraints. The **Lagrange**

Dual Function is defined as the minimum of the Lagrangian over $\lambda \in \mathbb{R}^m$ and $\nu \in \mathbb{R}^p$.

$$q(\lambda, \nu) = inf_{x \in X} L(x, \lambda, \nu) = inf_{x \in X}\{f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{m} \nu_i h_i(x)\}$$

Since the $q(\lambda, \nu)$ is an **infimum** of a family of affine functions, then it is a **concave** function. The **Lagrange Dual** function has lower bounds on optimal values (P^*).

For $\lambda \geq 0$ and any $\nu -> q(\lambda) \leq p^*$. The idea is to find the best **lower bound** that can be obtained with the Lagrange dual function.

$$\begin{aligned} &\text{maximize} \quad q(\lambda, \nu) \\ &\text{subject to} \quad \lambda \geq 0 \end{aligned} \tag{1}$$

Where d^* is the best solution of the Lagrange Dual Problem.

$$d* = sup\{q(\lambda, \nu) | \lambda \geq 0\} < \infty$$

The weak duality theorem says that for the general problem, the optimal value of the Lagrange dual problem (d^*) and the optimal value of the primal minimization problem (p^*) are related by:

$$d^* \leq p^*$$

This means that the dual problem provides the lower bound for the primal problem. The opposite holds true for a primal maximization problem. The difference between the two optimal values is called the optimal duality gap. The strong duality theorem says that for convex problems that satisfy certain conditions, the optimal duality gap is zero, meaning that the optimal values of the primal and dual problems are the same. For convex problems to guarantee the strong duality condition, Slater's constraint qualifications must be met, i.e. the convex problem must be strictly feasible. If $d^* \leq p^*$, then there is **weak duality**, while if $d^* = p^*$ there is **strong duality**.

The **Karush-Kuhn-Tucker** (KKT) conditions are first derivative tests (a.k.a. first-order necessary conditions) for a solution in non-linear programming to be optimal, provided that some regularity conditions are satisfied. If we assume that x^* is the optimal point of the primal problem and (λ^*, ν^*) the optimal points of the dual problem. The conditions are the following ones:

1. **Primal constraints:**

$$f_i(x^*) \leq 0 \quad i = 1, ..., m$$

2. **Primal constraints:**

$$h_i(x*) = 0 \quad i = 1, ..., p$$

3. **Dual constraints:**

$$\lambda_i \geq 0 \quad i = 1, ..., m$$

4. **Complementary slackness:**

$$\lambda_i * f_i(x*) = 0 \quad i = 1, ..., m$$

5. **Gradient of Lagrangian vanishes:**

$$\nabla_x L(x, \lambda, \nu) = \nabla f_0(x*) + \sum_{i=1}^{m} \lambda_i^* \nabla f_i(x*) + \sum_{i=1}^{p} \nu_i^* \nabla h_i(x*) = 0$$

2 Exercise 1

Given the following **objective function**:

$$\text{minimize} \quad e^{x_1}(4x_1^2 + 2x_2^2 + 4x_1x_2 + 2x_2 + 1)$$

With the following **constraints** and values for the variables:

$$x_1 * x_2 - x_1 - x_2 \leq -1.5$$

$$-x_1 * x_2 \leq -10$$

And the values for the variables as:

$$\text{var} \quad x_1, x_2$$

2.1 Identify whether is convex or not.

With the definitions explained before we can say that the domain of the problem is convex, as the variables in the set are real, i.e. all the linear combinations between the two points in the existing set (the representation of the function line). Once the domain is checked, we have to analyze the objective function to see if it is whether convex or not. The easy way to check is by checking the **second-order condition**. The Hessian matrix is:

$$H(f) = \begin{bmatrix} e^{x_1}(4x_1^2 + 4x_1(x_2 + 4) + 2x_2^2 + 10x_2 + 9) & e^{x_1}(4x_1 + 4x_2 + 6) \\ e^{x_1}(4x_1 + 4x_2 + 6) & 4e^{x_1} \end{bmatrix}$$

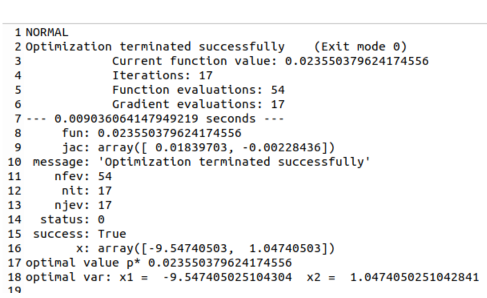
Now once the Hessian matrix is calculated we have to calculate the determinant of the Hessian matrix to know if it is positive semi-definite.

$$Det(H(f)) = -8e^{2x_1}(2x_1(x_2 - 1) + x_2(x_2 + 1)) < 0$$

As the determinant of the Hessian matrix is less than 0 we can say that the Hessian is not positive semi-definite and then the objective function to minimize is not convex. Summarizing we can say that the problem is not convex due to that the objective function is not convex.

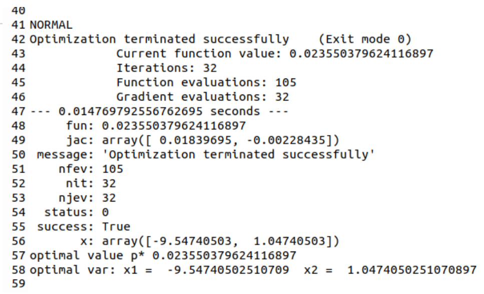
2.2 Find the minimum, e.g. use scipy.optimize.minimize (SLSQP as method) of python. Use as initial points x0 [0,0], [10,20], [-10,1], [-30,-30] and explain the difference in the solutions if any. Choose which ones give an optimal point/value and give how long take to converge to a result. Plot the objective curve and see whether the plot helps you to understand the optimization problem an results.

The program used could solve all the problems, i.e. find the points that minimizes the objective function, with the different initial guesses.



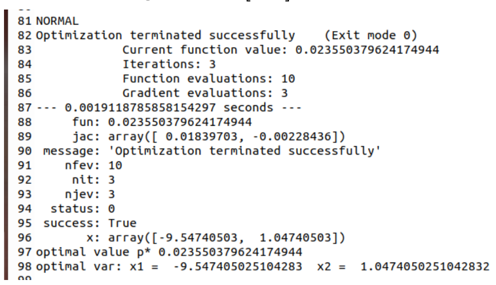
```
NORMAL
Optimization terminated successfully    (Exit mode 0)
            Current function value: 0.023550379624174556
            Iterations: 17
            Function evaluations: 54
            Gradient evaluations: 17
--- 0.009036064147949219 seconds ---
     fun: 0.023550379624174556
     jac: array([ 0.01839703, -0.00228436])
 message: 'Optimization terminated successfully'
    nfev: 54
     nit: 17
    njev: 17
  status: 0
 success: True
       x: array([-9.54740503,  1.04740503])
optimal value p* 0.023550379624174556
optimal var: x1 =  -9.547405025104304  x2 =  1.0474050251042841
```

Figure 1: Output of the program with initial guess as [0,0].



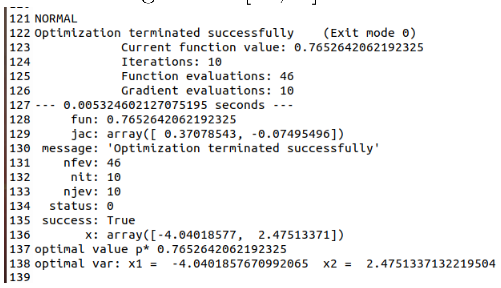
```
NORMAL
Optimization terminated successfully    (Exit mode 0)
            Current function value: 0.023550379624116897
            Iterations: 32
            Function evaluations: 105
            Gradient evaluations: 32
--- 0.014769792556762695 seconds ---
     fun: 0.023550379624116897
     jac: array([ 0.01839695, -0.00228435])
 message: 'Optimization terminated successfully'
    nfev: 105
     nit: 32
    njev: 32
  status: 0
 success: True
       x: array([-9.54740503,  1.04740503])
optimal value p* 0.023550379624116897
optimal var: x1 =  -9.54740502510709  x2 =  1.0474050251070897
```

Figure 2: Output of the program with initial guess as [10,20].



```
NORMAL
Optimization terminated successfully    (Exit mode 0)
            Current function value: 0.023550379624174944
            Iterations: 3
            Function evaluations: 10
            Gradient evaluations: 3
--- 0.0019118785858154297 seconds ---
     fun: 0.023550379624174944
     jac: array([ 0.01839703, -0.00228436])
 message: 'Optimization terminated successfully'
    nfev: 10
     nit: 3
    njev: 3
  status: 0
 success: True
       x: array([-9.54740503,  1.04740503])
optimal value p* 0.023550379624174944
optimal var: x1 =  -9.547405025104283  x2 =  1.0474050251042832
```

Figure 3: Output of the program with initial guess as [-10,1].



```
NORMAL
Optimization terminated successfully    (Exit mode 0)
            Current function value: 0.7652642062192325
            Iterations: 10
            Function evaluations: 46
            Gradient evaluations: 10
--- 0.005324602127075195 seconds ---
     fun: 0.7652642062192325
     jac: array([ 0.37078543, -0.07495496])
 message: 'Optimization terminated successfully'
    nfev: 46
     nit: 10
    njev: 10
  status: 0
 success: True
       x: array([-4.04018577,  2.47513371])
optimal value p* 0.7652642062192325
optimal var: x1 =  -4.0401857670992065  x2 =  2.4751337132219504
```

Figure 4: Output of the program with initial guess as [-30,-30].

In the following table we can see all the results from the solver using different initial guesses for x_1 and x_2.

Initial guess	**Optimal value** (p^*)	**Optimal variable** x_1	**Optimal variable** x_2
[0, 0]	0.02355	-9.5474	1.0474
[10, 20]	0.02355	-9.5474	1.0474
[-10, 1]	0.02355	-9.5474	1.0474
[-30, -30]	0.7653	-4.0402	2.4751

Table 1: Results of the solver with the different initial guesses.

Finally, the best way to understand the problem and how it is solved is by looking the graphic representation, e.g. plots. Below there are the plots of the objective function and the optimal values (minimal values) found in the optimization problem. One plot is from the matplotlib package and the other from GeoGebra. I put GeoGebra plot in order to see better the representation of the graph.

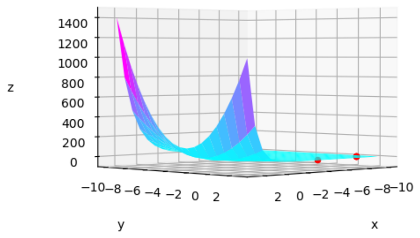


Figure 5: Matplotlib's plot of the objective function and optimal values.

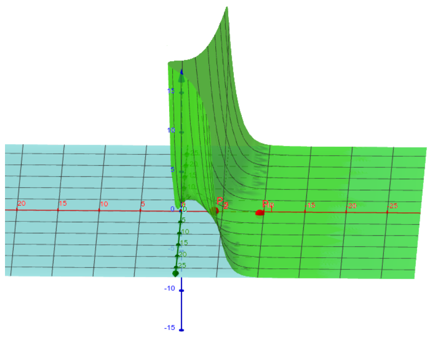


Figure 6: GeoGebra's plot of the objective function and optimal values.

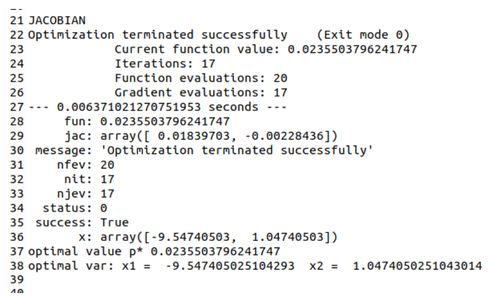
As we can see in the table and in the plot, the objective function is not convex because it has different local minimums, found by the solver given different initial guesses, and one global minimum. That confirms the affirming said before, the objective function to minimize is not convex because there is not a unique minimum, global minimum.

2.3 Give as input the Jacobian (exact gradient) to the method, and repeat and check whether the method speeds up.

If we want to find the Jacobian matrix we need to make the gradient of the objective function, as the Jacobian matrix is the same as the Gradient of the function. The Jacobian matrix contains then, all the partial derivatives.

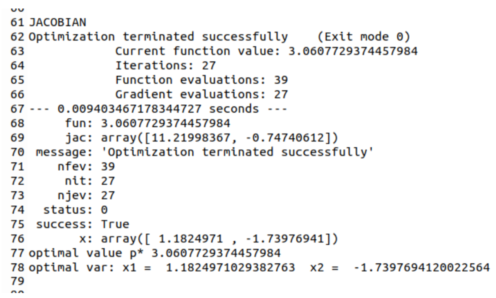
$$\nabla_f = \begin{bmatrix} e_1^x(4x_1^2 + 4x_1(x_2+2) + 2x_2^2 + 6x_2 + 1) \\ e_1^x(4x_1 + 4x_2 + 2) \end{bmatrix}$$

The outputs given by sending the Jacobian matrix to solver are shown below.



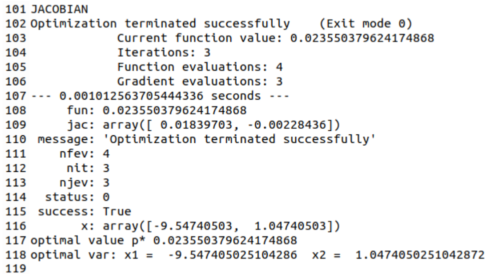
```
JACOBIAN
Optimization terminated successfully    (Exit mode 0)
            Current function value: 0.0235503796241747
            Iterations: 17
            Function evaluations: 20
            Gradient evaluations: 17
--- 0.006371021270751953 seconds ---
     fun: 0.0235503796241747
     jac: array([ 0.01839703, -0.00228436])
 message: 'Optimization terminated successfully'
    nfev: 20
     nit: 17
    njev: 17
  status: 0
 success: True
       x: array([-9.54740503,  1.04740503])
optimal value p* 0.0235503796241747
optimal var: x1 =  -9.547405025104293  x2 =  1.0474050251043014
```

Figure 7: Output of the program with initial guess as [0,0] using Jacobian matrix.



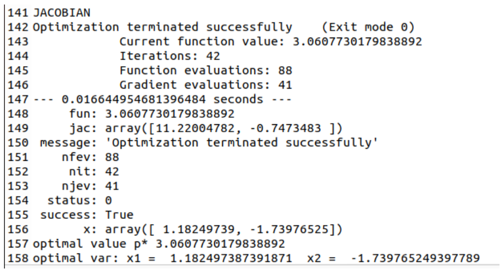
```
JACOBIAN
Optimization terminated successfully    (Exit mode 0)
            Current function value: 3.0607729374457984
            Iterations: 27
            Function evaluations: 39
            Gradient evaluations: 27
--- 0.009403467178344727 seconds ---
     fun: 3.0607729374457984
     jac: array([11.21998367, -0.74740612])
 message: 'Optimization terminated successfully'
    nfev: 39
     nit: 27
    njev: 27
  status: 0
 success: True
       x: array([ 1.1824971 , -1.73976941])
optimal value p* 3.0607729374457984
optimal var: x1 =  1.1824971029382763  x2 =  -1.7397694120022564
```

Figure 8: Output of the program with initial guess as [10,20] using Jacobian matrix.



```
JACOBIAN
Optimization terminated successfully    (Exit mode 0)
            Current function value: 0.02355037962417486 8
            Iterations: 3
            Function evaluations: 4
            Gradient evaluations: 3
--- 0.001012563705444336 seconds ---
     fun: 0.023550379624174868
     jac: array([ 0.01839703, -0.00228436])
 message: 'Optimization terminated successfully'
    nfev: 4
     nit: 3
    njev: 3
  status: 0
 success: True
       x: array([-9.54740503,  1.04740503])
optimal value p* 0.023550379624174868
optimal var: x1 =  -9.547405025104286  x2 =  1.0474050251042872
```

Figure 9: Output of the program with initial guess as [-10,1] using Jacobian matrix.



```
JACOBIAN
Optimization terminated successfully    (Exit mode 0)
            Current function value: 3.0607730179838892
            Iterations: 42
            Function evaluations: 88
            Gradient evaluations: 41
--- 0.016644954681396484 seconds ---
     fun: 3.0607730179838892
     jac: array([11.22004782, -0.7473483 ])
 message: 'Optimization terminated successfully'
    nfev: 88
     nit: 42
    njev: 41
  status: 0
 success: True
       x: array([ 1.18249739, -1.73976525])
optimal value p* 3.0607730179838892
optimal var: x1 =  1.182497387391871  x2 =  -1.739765249397789
```

Figure 10: Output of the program with initial guess as [-30,-30] using Jacobian matrix.

In the following table there are the number of iterations, function evaluations and gradient evaluations as the output of the solver program by using the different initial guesses.

Initial guesses	**Normal** (Iterations, function evaluations, gradient evaluations)	**Jacobian** (Iterations, function evaluations, gradient evaluations)
$x_1 = 0$, $x_2 = 0$	17, 54, 17	17, 20, 17
$x_1 = 10$, $x_2 = 20$	32, 105, 32	27, 39, 27
$x_1 = -10$, $x_2 = 1$	3, 10, 3	3, 4, 3
$x_1 = -30$, $x_2 = -30$	10, 46, 10	42, 88, 41

We can see that the **function evaluations** decrease in all the different cases and the iterations in some cases as in $x_1 = 10, x_2 = 20$. However, is important to remark that in the last initial guess ($x_1 = -30, x_2 = -30$) the Jacobian is not giving good iterations and not even good results. So, summarizing, we can say that Jacobian matrix improve in the performance of the solver in finding the solution of the problem. That is because if we do not give the Jacobian, the solver needs to calculate how much it has to descend to find the minimum (calculating the gradient), but if we give to the solver the Jacobian we are facilitating the calculation of the descent.

3 Exercise 2

Given the following **objective function**:

$$\text{minimize } x_1^2 + x_2^2$$

With the following **constraints**:

$$0.5 \leq x_1$$

$$-x_1 - x_2 + 1 \leq 0$$
$$-x_1^2 - x_2^2 + 1 \leq 0$$
$$-9x_1^2 - x_2^2 + 9 \leq 0$$
$$-x_1^2 + x_2 \leq 0$$
$$-x_2^2 + x_1 \leq 0$$

3.1 Identify whether is convex or not.

To know if the problem to solve is convex or not we have to check for the **domain**. As we can see in the problem statement, x_1 and x_2 can take all the existing values ($\mathbb{R}$). Now we have to check the **Hessian matrix** and see if it is positive semi-definite.

$$H(f) = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

The last step is to check for the **determinant** of the Hessian matrix.

$$Det(H(f)) = 4 > 0$$

The determinant is bigger than 0, so we can say that the Hessian matrix is **positive semi-definite** and then, the objective function is convex.

3.2 Propose an initial point that is not feasible and an initial point that is feasible and check what happens with SLSQP as method.

For this section two different initial guesses have been choosing. These are [(10, -10), (10, 10)]. The first one is not in a feasible set of the function, i.e. it does not fulfill all the constraints, and the second one is in a feasible set. That means only the second initial guess finds a valid solution for the problem (minimize the objective function by specific constraints). For example, the first guess, $x_1 = 10$ and $x_2 = -10$, does not fulfill all the constraints and then is not in the feasible set of the problem.

$$-(10) - (-10) + 1 = 9 \nleq 0$$

On the other hand, the second guess, $x_1 = 10$ and $x_2 = 10$, fulfills all the constraints given in the problem.

$$0.5 \leq 10$$

$$-10 - 10 + 1 \leq 0$$

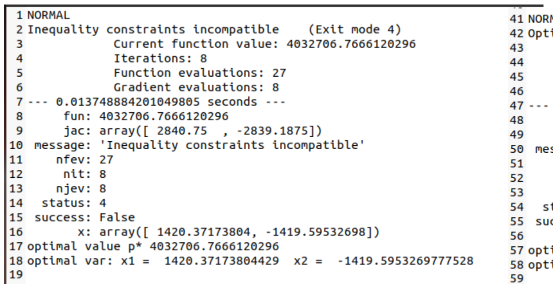
$$-(10^2) - (10^2) + 1 \leq 0$$

$$-9 * (10^2) - 10^2 + 9 \leq 0$$

$$-(10^2) + 10 \leq 0$$

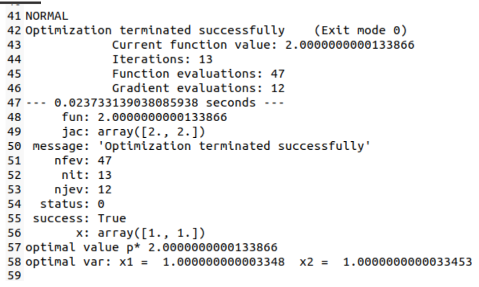
$$-(10^2) + 10 \leq 0$$

In the following images we can see the outputs of the solver with both initial guesses.



```
NORMAL
Inequality constraints incompatible    (Exit mode 4)
            Current function value: 4032706.7666120296
            Iterations: 8
            Function evaluations: 27
            Gradient evaluations: 8
--- 0.013748884201049805 seconds ---
     fun: 4032706.7666120296
     jac: array([ 2840.75  , -2839.1875])
 message: 'Inequality constraints incompatible'
    nfev: 27
     nit: 8
    njev: 8
  status: 4
 success: False
       x: array([ 1420.37173804, -1419.59532698])
optimal value p* 4032706.7666120296
optimal var: x1 =  1420.37173804429  x2 =  -1419.5953269777528
```

Figure 11: Output of the program with initial guess as [10,-10].



```
NORMAL
Optimization terminated successfully    (Exit mode 0)
            Current function value: 2.0000000000133866
            Iterations: 13
            Function evaluations: 47
            Gradient evaluations: 12
--- 0.023733139038085938 seconds ---
     fun: 2.0000000000133866
     jac: array([2., 2.])
 message: 'Optimization terminated successfully'
    nfev: 47
     nit: 13
    njev: 12
  status: 0
 success: True
       x: array([1., 1.])
optimal value p* 2.0000000000133866
optimal var: x1 =  1.000000000003348  x2 =  1.0000000000033453
```

Figure 12: Output of the program with initial guess as [10,10].

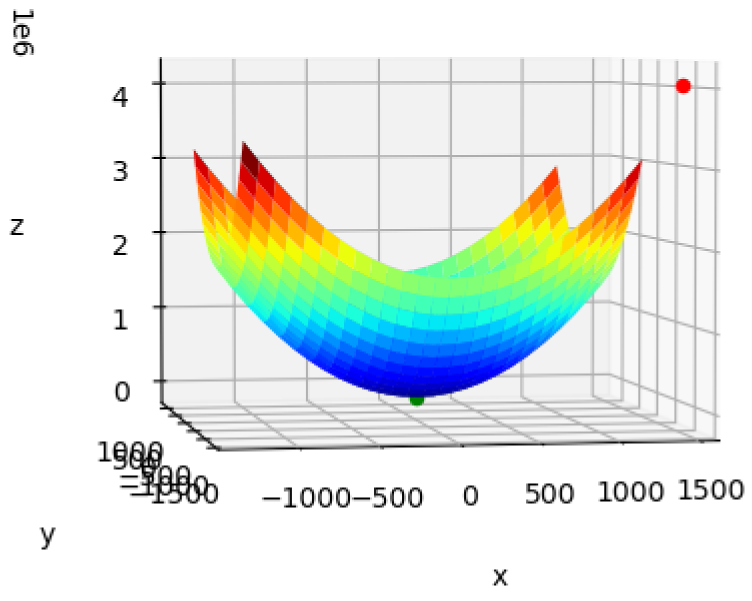


Figure 13: Plot of objective function and optimal values.

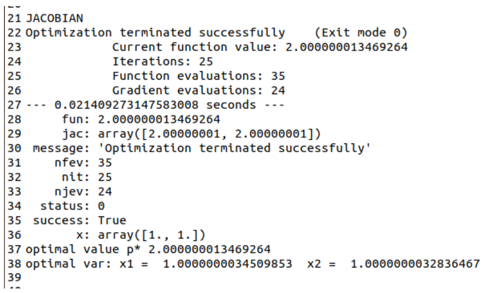
In the plot we can see that the objective function is convex; in other words, it has only an unique global minimum that fulfill all the constraints (feasible point). In addition, we can also see two points. The green dot represents the minimum found by the solver that is in the feasible set and, the red one, represents the minimum found by the solver that is in the not feasible set.

3.3 Repeat giving as input the Jacobian. Check the convergence time (or number of steps).

To find the Jacobian matrix we need to find the Gradient of the function.

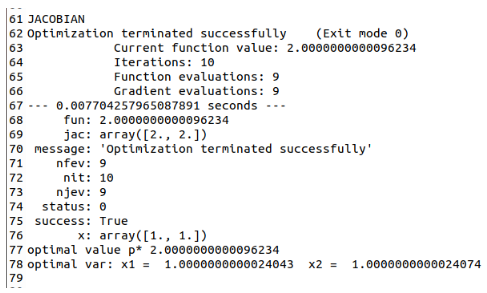
$$\nabla_f = \begin{bmatrix} 2x_1 \\ 2x_2 \end{bmatrix}$$

The outputs given by sending the Jacobian matrix to solver are shown below.



```
JACOBIAN
Optimization terminated successfully    (Exit mode 0)
            Current function value: 2.000000013469264
            Iterations: 25
            Function evaluations: 35
            Gradient evaluations: 24
--- 0.021409273147583008 seconds ---
     fun: 2.000000013469264
     jac: array([2.00000001, 2.00000001])
 message: 'Optimization terminated successfully'
    nfev: 35
     nit: 25
    njev: 24
  status: 0
 success: True
       x: array([1., 1.])
optimal value p* 2.000000013469264
optimal var: x1 =  1.0000000034509853  x2 =  1.000000032836467
```

Figure 14: Output of the program with initial guess as [10,-10] using Jacobian matrix.



```
JACOBIAN
Optimization terminated successfully    (Exit mode 0)
            Current function value: 2.0000000000096234
            Iterations: 10
            Function evaluations: 9
            Gradient evaluations: 9
--- 0.007704257965087891 seconds ---
     fun: 2.0000000000096234
     jac: array([2., 2.])
 message: 'Optimization terminated successfully'
    nfev: 9
     nit: 10
    njev: 9
  status: 0
 success: True
       x: array([1., 1.])
optimal value p* 2.0000000000096234
optimal var: x1 =  1.0000000000024043  x2 =  1.0000000000024074
```

Figure 15: Output of the program with initial guess as [10,10] using Jacobian matrix.

In the following table there are the number iterations, function evaluations and gradient evaluations as the output of the solver program by using the different initial guesses.

Initial guesses	**Normal** (Iterations, function evaluations, gradient evaluations)	**Jacobian** (Iterations, function evaluations, gradient evaluations)
$x_1 = 10$, $x_2 = -10$ **(Not feasible)**	8, 27, 8	9, 8, 8
$x_1 = 10$, $x_2 = 10$ **(Feasible)**	13, 47, 12	13, 17, 12

In the table above we can see the same thing that we saw in the previous exercise, giving the Jacobian matrix improves the performance of the solver in terms of finding the solution.

4 Exercise 3

Given the following **objective function** to:

$$\text{minimize } x_1^2 + x_2^2$$

With the following **constraints**:

$$x_1^2 + x_1x_2 + x_2^2 \leq 3$$

$$3x_1 + 2x_2 \geq 3$$

And the values for the variables as:

$$\text{var} \quad x_1, x_2$$

4.1 Identify whether is convex or not.

To know if the problem to solve is convex or not we have to check for the **domain**. As we can see in the problem statement, x_1 and x_2 can take all the existing values ($\mathbb{R}$). Now we have to check the **Hessian matrix** and see if it is positive semi-definite.

$$H(f) = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

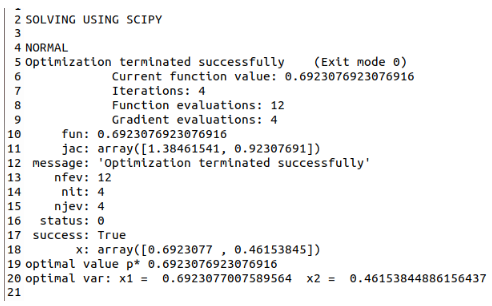
The last step is to check for the **determinant** of the Hessian matrix.

$$Det(H(f)) = 4 > 0$$

The determinant is bigger than 0, so we can say that the Hessian matrix is **positive semi-definite** and then, the objective function is convex.

4.2 Consider the following non-linear problem. Say whether is a COP and solve it using the scipy library. Check convergence. Learn how to use the CVX toolbox to program it.

The initial guess used to solve the problem has been $x_1 = 10$ and $x_2 = 10$. In the following images there are the outputs given by the different solvers and implementations.

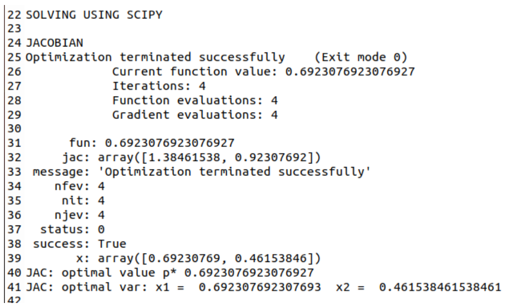


```
SOLVING USING SCIPY

NORMAL
Optimization terminated successfully    (Exit mode 0)
            Current function value: 0.6923076923076916
            Iterations: 4
            Function evaluations: 12
            Gradient evaluations: 4
     fun: 0.6923076923076916
     jac: array([1.38461541, 0.92307691])
 message: 'Optimization terminated successfully'
    nfev: 12
     nit: 4
    njev: 4
  status: 0
 success: True
       x: array([0.6923077 , 0.46153845])
optimal value p* 0.6923076923076916
optimal var: x1 =  0.6923077007589564  x2 =  0.46153844886156437

```

Figure 16: Output of the program with initial guess as [10,10] and SCIPY package.

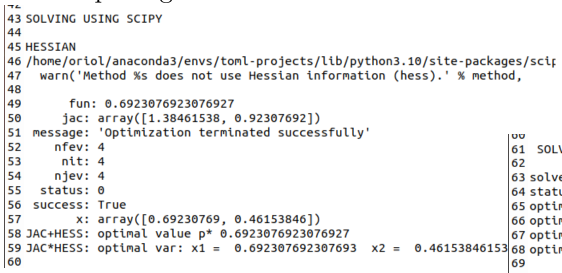


```
SOLVING USING SCIPY

JACOBIAN
Optimization terminated successfully    (Exit mode 0)
            Current function value: 0.6923076923076927
            Iterations: 4
            Function evaluations: 4
            Gradient evaluations: 4

     fun: 0.6923076923076927
     jac: array([1.38461538, 0.92307692])
 message: 'Optimization terminated successfully'
    nfev: 4
     nit: 4
    njev: 4
  status: 0
 success: True
       x: array([0.69230769, 0.46153846])
JAC: optimal value p* 0.6923076923076927
JAC: optimal var: x1 =  0.692307692307693  x2 =  0.461538461538461
```

Figure 17: Output of the program with initial guess as [10,10] and SCIPY package using the Jacobian matrix.



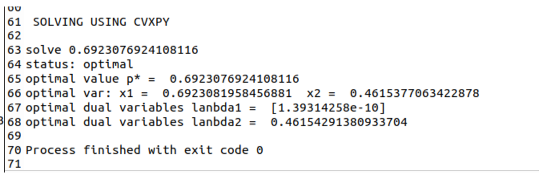
```
SOLVING USING SCIPY

HESSIAN
/home/oriol/anaconda3/envs/toml-projects/lib/python3.10/site-packages/scip
  warn('Method %s does not use Hessian information (hess).' % method,

     fun: 0.6923076923076927
     jac: array([1.38461538, 0.92307692])
 message: 'Optimization terminated successfully'
    nfev: 4
     nit: 4
    njev: 4
  status: 0
 success: True
       x: array([0.69230769, 0.46153846])
JAC+HESS: optimal value p* 0.6923076923076927
JAC*HESS: optimal var: x1 =  0.692307692307693  x2 =  0.46153846153

```

Figure 18: Output of the program with initial guess as [10,10] and SCIPY package using the Jacobian matrix and the Hessian matrix.



```
 SOLVING USING CVXPY

solve 0.6923076924108116
status: optimal
optimal value p* =  0.6923076924108116
optimal var: x1 =  0.6923081958456881  x2 =  0.4615377063422878
optimal dual variables lanbda1 =  [1.39314258e-10]
optimal dual variables lanbda2 =  0.46154291380933704

Process finished with exit code 0

```

Figure 19: Output of the program with initial guess as [10,10] and CVXPY package.

As we can see the solution for all the problems is the same. The optimal value is $p^* = 0.6923$, the optimal variables are $x_1 = 0.6923, x_2 = 0.4615$ and finally the lambda values are $\lambda_1 = 1.39314258e-10, \lambda_2 = 0.4615$.

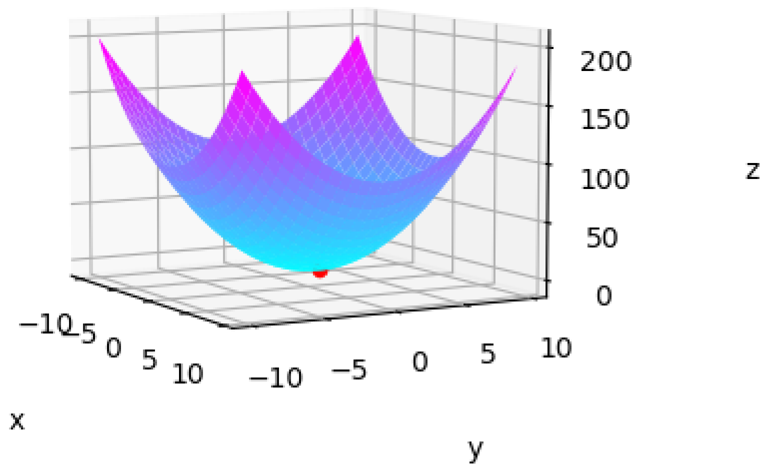


Figure 20: Plot of objective function and optimal value.

The plot help us to visualize the problem and to check if the solutions found makes sense. As it is shown in the plot the objective function is convex and all the solvers find the same solution at the same point, the unique global minimum.

5 Exercise 4

Given the following **objective function**:

$$\text{minimize } x^2 + 1$$

With the following **constraint**:

$$(x-2)(x-4) \leq 0$$

5.1 Identify whether is convex or not.

To verify the convexity of the problem we have to verify the domain is convex. As there is no restrictions of the values that can take the variable x the domain are all the existing values and then is a convex domain. Finally to know that some a objective function we have to fulfill the second-order condition. That means we have to calculate the Hessian matrix of the objective function.

$$H(f) = \begin{bmatrix} 2 \end{bmatrix}$$

Once we have the Hessian matrix is time to see the determinant of it.

$$Det(H(f)) = 2 > 0$$

As the determinant of the Hessian matrix is bigger than 0, the Hessian matrix is positive semi-definite and then the second-order condition has been fulfilled.

5.2 Use the CVX toolbox to program it.

To solve the problem the solver from CVXPY package has been used. In this solver we have to define in a different way all the parameters, constraints, objective function and bounds.

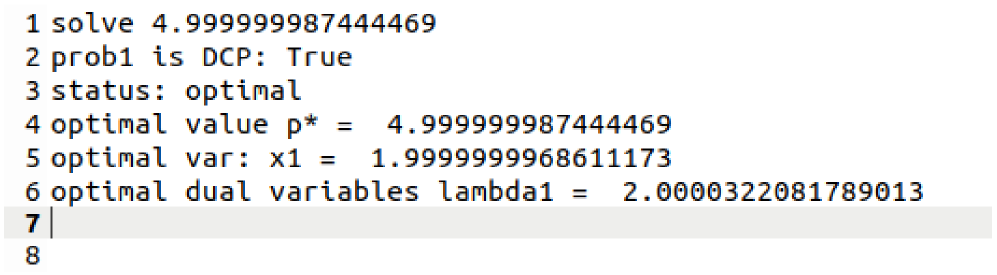
Is important to highlight that the minimum that the solver will find is not the global minimum. That is because of the constraints that does not allow a solution feasible in the global minimum of the function. The way to solve this problem is by using the **Lagrangian Dual Problem**.

$$\begin{aligned} &\text{maximize} && q(\lambda) = inf_{x \in X}\{x^2 + 1 + \lambda_1(x-2)(x-4) \\ &\text{subject to} && \lambda_1 \geq 0 \end{aligned} \tag{2}$$

Once the problem is formalized the way to solve is by making the respective derivative with x and set an equation equals to 0 to verify for the KKT conditions. After using the slackness conditions the result is the following one.

$$\lambda_1 = -\frac{x}{(x-3)}$$

Below there is the output of the solver when executing the optimization problem.


```
solve 4.999999987444469
prob1 is DCP: True
status: optimal
optimal value p* =  4.999999987444469
optimal var: x1 =  1.9999999968611173
optimal dual variables lambda1 =  2.0000322081789013


```

Figure 21: Output of the solver using CVXPY.

As we can see the solutions of the solver are:

- $p* = 4.9999 \approx 5$ (**optimal value**).
- $x = 1.9999 \approx 2$ (**optimal variable**).
- $\lambda = 2$ (**dual value of the constraints**).

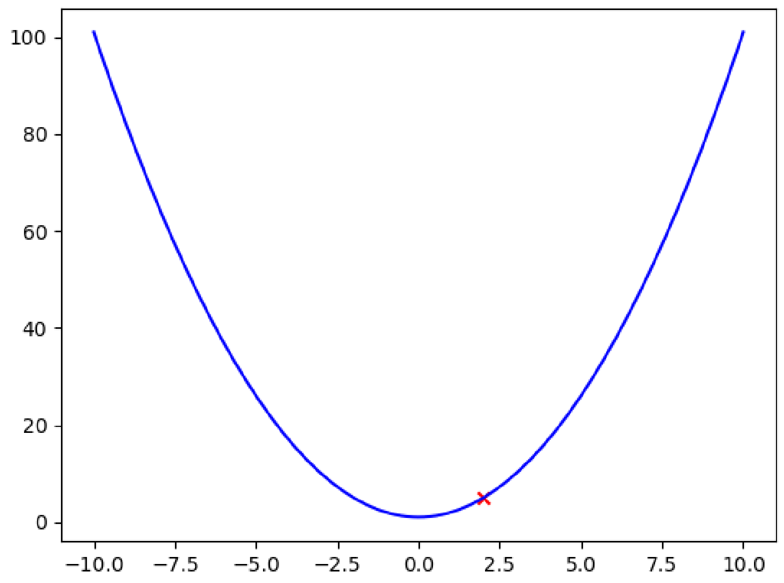


Figure 22: Plot of objective function and optimal values.

Finally in the plot we can see the minimum that fulfills all the constraints.

6 Exercise 5

Given the following **objective function**:

$$\text{minimize } x_1^2 + x_2^2$$

With the following **constraints**:

$$(x_1 - 1)^2 + (x_2 - 1)^2 \leq 1$$

$$(x_1 - 1)^2 + (x_2 + 1)^2 \leq 1$$

6.1 Identify whether is convex or not.

To identify if the problem is convex or not we have to look to the domain of the objective function given and determine if it is convex or not. As all the x_1 and x_2 can take all the existing values we can say there is no restrictions and then the domain of the function is totally convex. Then, once we know the domain of the function is convex we have to check for the **second-order condition** if it is fulfilled. To do it, first we have to calculate the Hessian matrix of the objective function.

$$H(f) = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

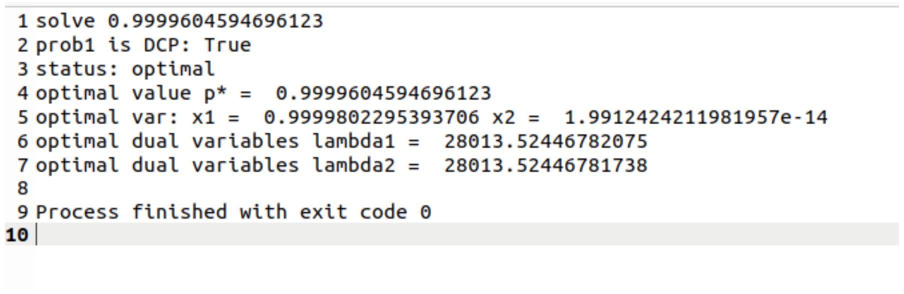
After finding the Hessian matrix we have to check for the **determinant** of the matrix.

$$Det(H(f)) = 4 > 0$$

The determinant of the matrix is bigger than 0, that means the Hessian matrix is positive semi-definite and then the **second-order condition** has been fulfilled, i.e. the objective function is convex.

6.2 Use the CVX toolbox to program it.

To solve the problem it has been used the solver from CVXPY package. The output of the program execution is shown below.



```
solve 0.9999604594696123
prob1 is DCP: True
status: optimal
optimal value p* =  0.9999604594696123
optimal var: x1 =  0.9999802295393706 x2 =  1.9912424211981957e-14
optimal dual variables lambda1 =  28013.52446782075
optimal dual variables lambda2 =  28013.52446781738

Process finished with exit code 0

```

Figure 23: Output of the solver from CVXPY package.

As we can see the solutions of the solver are:

- $p* = 1$ (**optimal value**).
- $x_1 = 1$ and $x_2 = 1.9912e - 14 \approx 0$ (**optimal variables**).
- $\lambda_1 = 28013.5245$ and $\lambda_2 = 28013.5245$ (**dual value of the constraints**).

Finally, in the following plot we can see the objective function is convex as we said in the previous section and the solver it has found the minimum that fulfills all the constraints (red point).

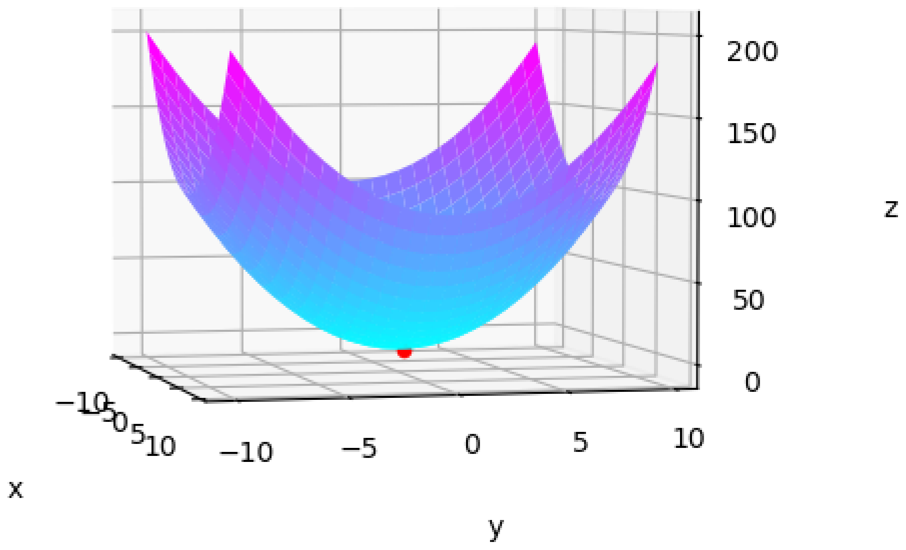


Figure 24: Plot of objective function and optimal value.

7 Exercise 6

In this exercise we have deployed the **Gradient Descent method** and the **Newton's method** by hand to solve different optimization problems. The Gradient Descent method is an iterative optimization algorithm that finds the local minimum of a differentiable function. The Gradient Descent method follows the equation shown below.

$$x^{(k+1)} = x^{(k)} + t\triangle x = x^{(k)} - t\nabla f(x^{(k)})$$

The parameter k represents the current step of the algorithm, and the parameter *k+1* the step to be calculated. The parameter t (a.k.a. **learning rate** in Machine Learning definitions) is the value that increase or decrease the size of the $\nabla f(x^k)$, making it larger or smaller depending on a value. This value can be constant or variable, large values make the space step by step bigger and low values the opposite. Is important to remark that if we are descending we have to put a minus in front of the t parameter. The Gradient of the function ($\nabla f(x^k)$) is a parameter which depicts the direction that the Gradient Descent method is moving to. The reason of this is because the Gradient is equivalent to the tangent line of the function in the current point. On the other hand, Newton's method is a root-finding algorithm which makes better approximations to the roots of a real-valued function. The Newton's method follows the equation shown below.

$$x^{(k+1} = x^{(k)} + \frac{f'(x^k)}{f''(x^k)}$$

As we can see the equation is pretty similar to the Gradient Descent method's one. The only thing that has changed are the new parameters, the inverse of the Hessian matrix and the Jacobian matrix of the value calculated. the Newton's method is really useful to solve some optimization problems that are hard to solve only with the local, Gradient. In the following image there is a graphic explanation to understand better the Newton's method.

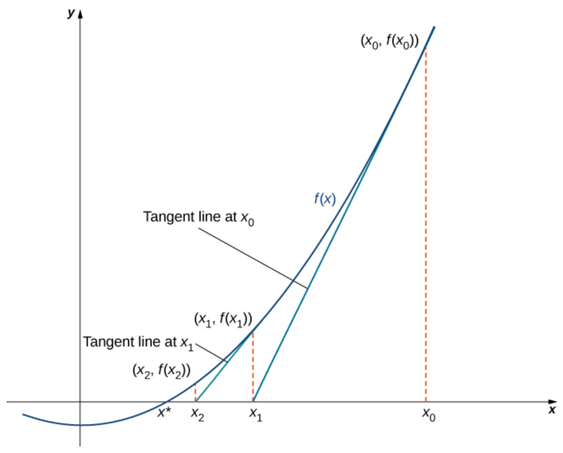


Figure 25: Tangents calculated in the steps of the Newton's method.

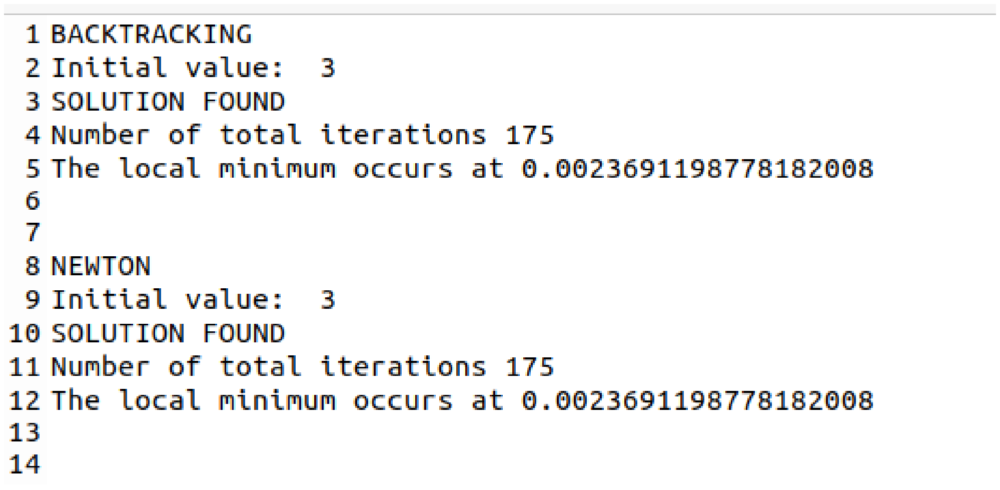
The biggest difference between both methods is that the Gradient Descent method maximizes/minimizes a function by the knowledge of its derivative. On the other hand, Newton's method maximizes/minimizes the function by the knowledge of its second derivative. The second derivative translates into faster way to get to the solution. However, sometimes, the analytic expression for the second derivative is often complicated or intractable, i.e. requires a lot of computation. Numerical methods for computing the second derivative also requires a lot of computation, in other words, if N values are required to compute first derivative, N^2 are required for the secon derivative

7.1 First optimization problem.

Given the following function:

$$f(x) = 2x^2 - 0.5$$

Which the initial point is $x^{(0)} = 3$ and accuracy (stop criterion) $\eta = 10^{-4}$. I applied both algorithms to search for the minimum of the function. Is important to highlight that the learning rate of both algorithms has been set in 0.01 ($t = 0.01$). The value taken it has been really small because we want to be sure that we can reach the solution without any problem and because the complexity of the function is simple (it does not take so long to reach a minimum) that we can take one small number to move from step to step. In addition to remark, I have used *np.linalg.cond()* function from numpy instead of the *np.linalg()* because we want to compute the condition number of a matrix. The output of the program using the Gradient Descent method and Newton's method is shown below.



```
BACKTRACKING
Initial value:  3
SOLUTION FOUND
Number of total iterations 175
The local minimum occurs at 0.0023691198778182008


NEWTON
Initial value:  3
SOLUTION FOUND
Number of total iterations 175
The local minimum occurs at 0.0023691198778182008


```

Figure 26: Output of the program using Gradient Descent method and Newton's method.

In the following plot we can see how the both methods reach the minimum of the function given.

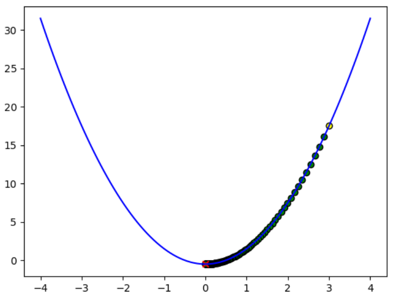


Figure 27: Output of the program with initial guess as [3] using Gradient Descent method.

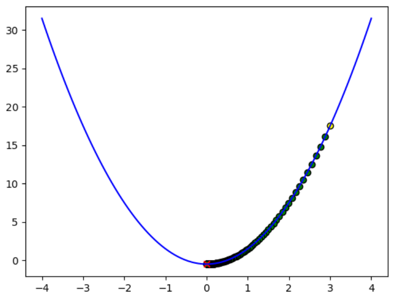


Figure 28: Output of the program with initial guess as [3] using Newton's method.

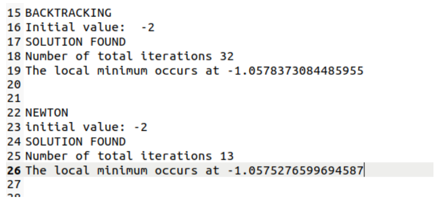
In the plots we can see how the Gradient Descent method is descending "*faster*" at the beginning and "*slower*" when it is close to the points. And how the Newton's method is going to the center of the minimum in the function.

7.2 Second optimization problem.

Given the following function:

$$f(x) = 2x^4 - 4x^2 + x - 0.5$$

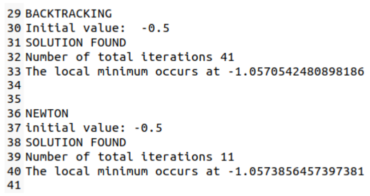
Which these initial points, $x^{(0)} = -2$, $x^{(0)} = -0.5$, $x^{(0)} = 0.5$ and $x^{(0)} = 2$. The accuracy (stop criterion) $\eta = 10^{-4}$. Is important to highlight that the learning rate of both algorithms has been set in 0.01 ($t = 0.01$). The value taken it has been really small because we want to be sure that we can reach the solution without any problem and because the complexity of the function is simple (it does not take so long to reach a minimum) that we can take one small number to move from step to step. In addition to remark, I have used *np.linalg.cond()* function from numpy instead of the *np.linalg()* because we want to compute the condition number of a matrix. The output of the program using the Gradient Descent method and Newton's method is shown below.



```
BACKTRACKING
Initial value:  -2
SOLUTION FOUND
Number of total iterations 32
The local minimum occurs at -1.0578373084485955


NEWTON
initial value: -2
SOLUTION FOUND
Number of total iterations 13
The local minimum occurs at -1.0575276599694587
```

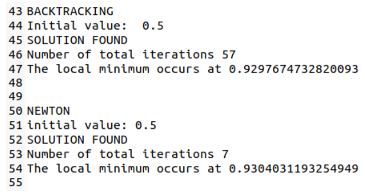
Figure 29: Output of the program with initial guess as [-2] using Gradient Descent method and Newton's method.



```
BACKTRACKING
Initial value:  -0.5
SOLUTION FOUND
Number of total iterations 41
The local minimum occurs at -1.0570542480898186


NEWTON
initial value: -0.5
SOLUTION FOUND
Number of total iterations 11
The local minimum occurs at -1.0573856457397381
```

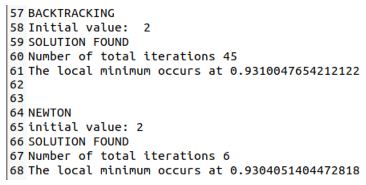
Figure 30: Output of the program with initial guess as [-0.5] using Gradient Descent method and Newton's method.



```
BACKTRACKING
Initial value:  0.5
SOLUTION FOUND
Number of total iterations 57
The local minimum occurs at 0.9297674732820093


NEWTON
initial value: 0.5
SOLUTION FOUND
Number of total iterations 7
The local minimum occurs at 0.9304031193254949
```

Figure 31: Output of the program with initial guess as [0.5] using Gradient Descent method and Newton's method.



```
BACKTRACKING
Initial value:  2
SOLUTION FOUND
Number of total iterations 45
The local minimum occurs at 0.9310047654212122


NEWTON
initial value: 2
SOLUTION FOUND
Number of total iterations 6
The local minimum occurs at 0.9304051404472818
```

Figure 32: Output of the program with initial guess as [2] using Gradient Descent method and Newton's method.

As we can see, all the results given for Newton's method have less iterations than the Gradient Descent method, i.e. Newton's method arrives faster to the local minimum. In the plots below, we can see how both methods reach a local minimum of the function given with the different initial guesses.

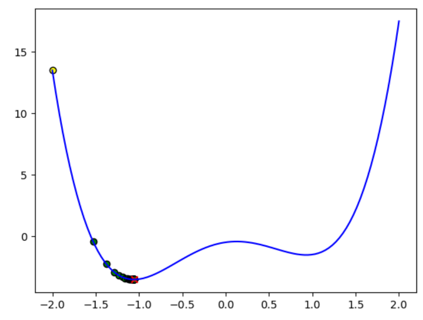


Figure 33: Plot of the function and the steps using the Gradient Descent method with the [-2] as initial guess.

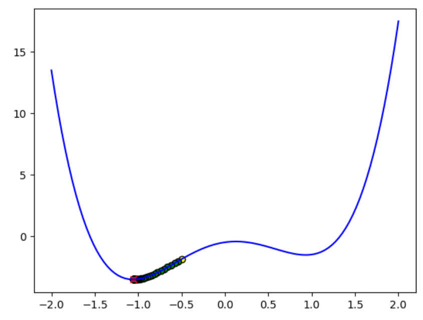


Figure 34: Plot of the function and the steps using the Gradient Descent method with the [-0.5] as initial guess.

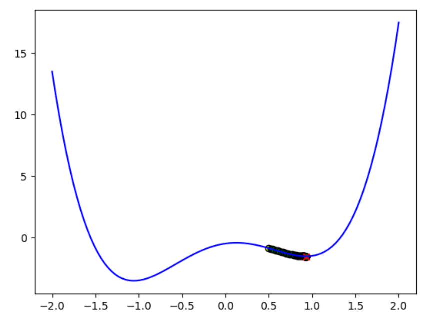


Figure 35: Plot of the function and the steps using the Gradient Descent method with the [0.5] as initial guess.

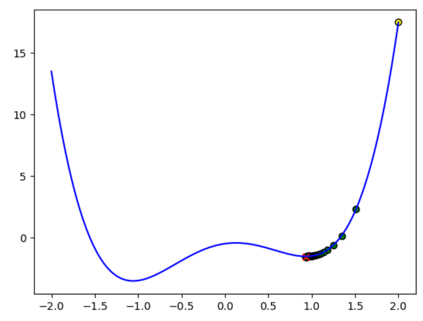


Figure 36: Plot of the function and the steps using the Gradient Descent method with the [2] as initial guess.

In the plots are shown how the Gradient Descent method is descending "*faster*" at the beginning and "*slower*" when it is close to the points.

8 Exercise 7

Consider a Network Utility problem such as the one given at class, slide 12 of Topic 12, in which 3 sources and 5 links with capacity $(C_1, C_2, C_3, C_4, C_5)$ =(1,2,1,2,1) respectively are considered and solve it using CVX. Source 1 traverses link 1 and 2, source 2 traverses link 2 and source 3 traverses link 1 and 5. Obtain also the Lagrange multipliers. From the problem statement we can state the following objective function:

$$\text{maximize } log(x_1) + log(x_2) + log(x_3)$$

With the following **constraints**:

$$x_1 + x_3 \leq 1$$

$$x_1 + x_2 \leq 2$$

$$x_3 \leq 1$$

$$x_1 \geq 0$$

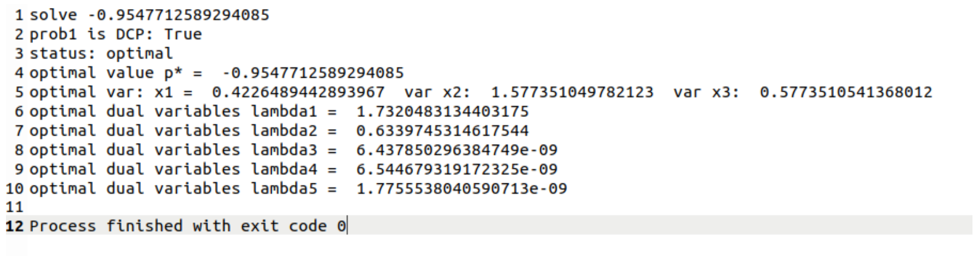
$$x_2 \geq 0$$

$$x_3 \geq 0$$

And the values for the variables as:

$$\text{var} \quad x_1, x_2, x_3$$

In the following image there is the output of the program using the solver from the CVXPY package.



```
solve -0.9547712589294085
prob1 is DCP: True
status: optimal
optimal value p* =  -0.9547712589294085
optimal var: x1 =  0.4226489442893967  var x2:  1.577351049782123  var x3:  0.5773510541368012
optimal dual variables lambda1 =  1.7320483134403175
optimal dual variables lambda2 =  0.6339745314617544
optimal dual variables lambda3 =  6.437850296384749e-09
optimal dual variables lambda4 =  6.544679319172325e-09
optimal dual variables lambda5 =  1.7755538040590713e-09

Process finished with exit code 0
```

Figure 37: Output of the program using the solver from CVXPY.

As we can see the solutions of the solver are:

- $p* = -0.9548$ (**optimal value**).
- $x_1 = 0.4226$, $x_2 = 1.5774$ and $x_3 = 0.5774$ (**optimal variables**).
- $\lambda_1 = 1.7320$, $\lambda_2 = 0.6340$, $\lambda_3 = 6.4379$, $\lambda_4 = 0.6.5447$ and $\lambda_5 = 1.7756$ (**dual value of the constraints**).

Is important to highlight that with this kind of problems with high complexity we can not plot anything due to the high dimensionality.

9 Exercise 8

Consider a Resource Allocation problem such as the one given at class, similar to slide 12 of Topic 13 but with a wireless network. Give the traffic allocated to each user x_i and the percentage of time each link R_ij is used. Give also the dual variables. The variables R_ij represent a slot in which node i is scheduled to transmit to node j. That translates to the following **objective function**:

$$\text{maximize} \sum_{i}^{N} \log(x_i)$$

With the following **constraints**:

$$x_1 + x_2 \leq R_{12}$$

$$x_1 \leq R_{23}$$

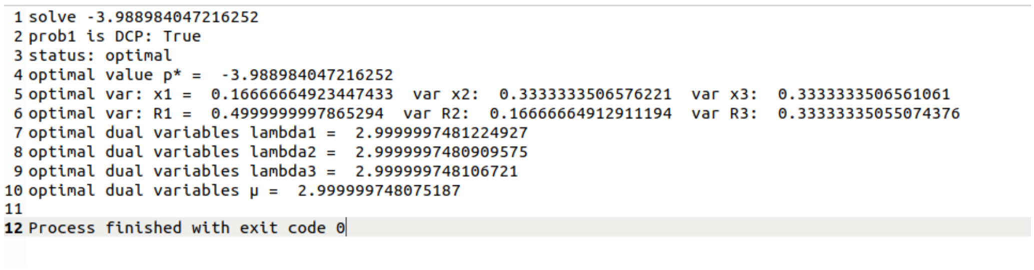
$$x_3 \leq R_{32}$$

$$R_{12} + R_{23} + R_{32} \leq 1$$

And the values for the variables as:

$$\text{var} \quad x_i, R_jk$$

In the following image there is the output of the program using the solver from the CVXPY package.



```
solve -3.988984047216252
prob1 is DCP: True
status: optimal
optimal value p* =  -3.988984047216252
optimal var: x1 =  0.16666664923447433  var x2:  0.3333333506576221  var x3:  0.3333333506561061
optimal var: R1 =  0.4999999997865294  var R2:  0.16666664912911194  var R3:  0.33333335055074376
optimal dual variables lambda1 =  2.9999997481224927
optimal dual variables lambda2 =  2.9999997480909575
optimal dual variables lambda3 =  2.999999748106721
optimal dual variables μ =  2.999999748075187

Process finished with exit code 0
```

Figure 38: Output of the program using the solver from CVXPY.

As we can see the solutions of the solver are:

- $p* = -3.9890$ (**optimal value**).
- $x_1 = 0.1667$, $x_2 = 0.3333$, $x_3 = 0.3333$, $R_12 = 0.5000$, $R_23 = 0.1667$ and $R_32 = 0.3333$ (**optimal variables**).
- $\lambda_1 = 3.0000$, $\lambda_2 = 3.0000$, $\lambda_3 = 3.0000$ and $\mu = 3.0000$ (**dual value of the constraints**).

Is important to highlight that with this kind of problems with high complexity we can not plot anything due to the high dimensionality.

10 References & links

In this section are referenced all the documentation (definitions, software programs for computation, etc.) that has been used in order to do all the exercises from the first laboratory project.

References

[1] Theory from the slides of subject Topics on Optimization and Machine Learning. José M. Barceló Ordinas, Departament d'Arquitectura de Computadors (UPC).

[2] Mk. Wolfram. Sci Nat [Internet]. 1913 [cited 2022 Apr 2];1(21):511–511. Available from: https://www.wolframalpha.com/

[3] Wikipedia contributors. Jacobian matrix and determinant [Internet]. Wikipedia, The Free Encyclopedia. 2022. Available from: https://en.wikipedia.org/wiki/Jacobian_matrix_and_determinant

[4] Wikipedia contributors. Gradient [Internet]. Wikipedia, The Free Encyclopedia. 2022. Available from: https://en.wikipedia.org/wiki/Gradient

[5] Wikipedia contributors. Hessian matrix [Internet]. Wikipedia, The Free Encyclopedia. 2022. Available from: https://en.wikipedia.org/wiki/Hessian_matrix

[6] Profit Maximization [Internet]. Github.io. [cited 2022 Apr 2]. Available from: https://saylordotorg.github.io/text_introduction-to-economic-analysis/s10-03-profit-maximization.html

[7] Welcome to CVXPY 1.2 — CVXPY 1.2 documentation [Internet]. Cvxpy.org. [cited 2022 Apr 3]. Available from: https://www.cvxpy.org/

[8] scipy.optimize.minimize — SciPy v1.8.0 Manual [Internet]. Scipy.org. [cited 2022 Apr 3]. Available from: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html

[9] Wikipedia contributors. Gradient descent [Internet]. Wikipedia, The Free Encyclopedia. 2022. Available from: https://en.wikipedia.org/wiki/Gradient_descent

[10] Baeldung.com. [cited 9th of May of 2022]. Available in: https://www.baeldung.com/cs/gradient-descent-vs-newtons-gradient-descent

[11] Lagrangean duality [Internet]. Northwestern.edu. [cited 9th of May of 2022]. Available in: https://optimization.mccormick.northwestern.edu/index.php/Lagrangean_duality