INTRODUCTION TO ARTIFICIAL INTELLIGENCE

EXERCISE 1

Full names: Andreas Kalavas, Panagiota Chita

Emails: andreas.kalavas.stud@pw.edu.pl , panagiota.chita.stud@pw.edu.pl

In this report, we have been analyzing and applying two optimization techniques in order to minimize the given function. The first method is Newton's method and the second one is the Gradient Descent method. For the solution of the first exercise, we decided to use Python as the programming language.

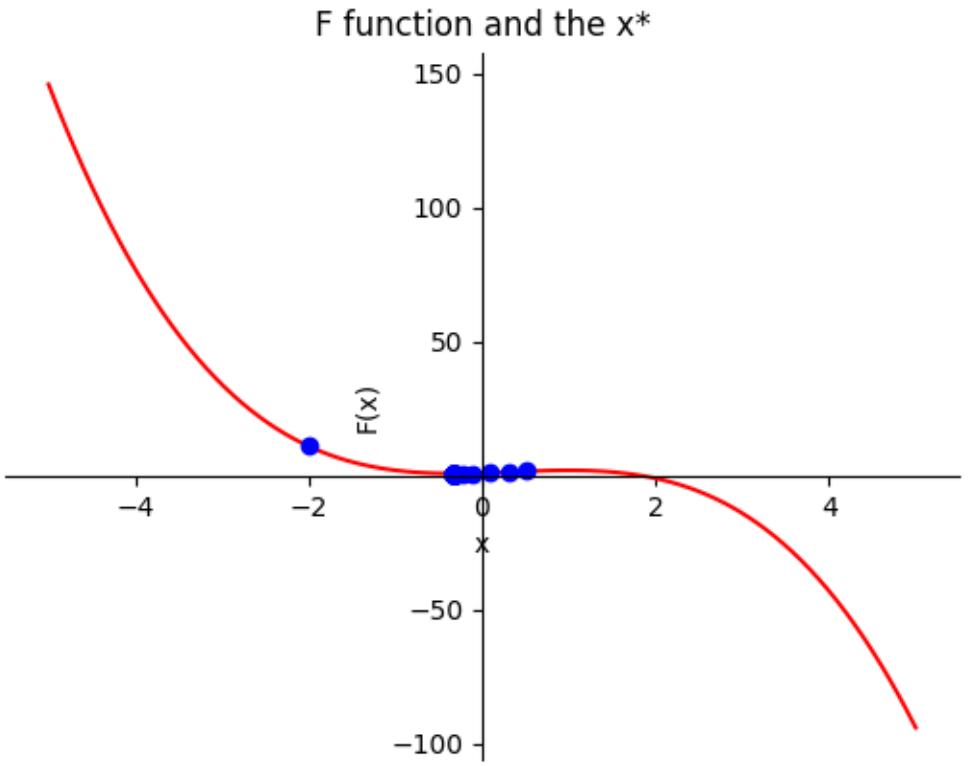
In the file *exe1.py* the user must make some decisions and give the corresponded inputs from the terminal. All the data must be integers in order to run the program. As far as the structure of the code, we have created four def (functions) for the function F: the newton_f and the gradient_f, while for the function G: te newton_g and the gradient_f.

Test1:

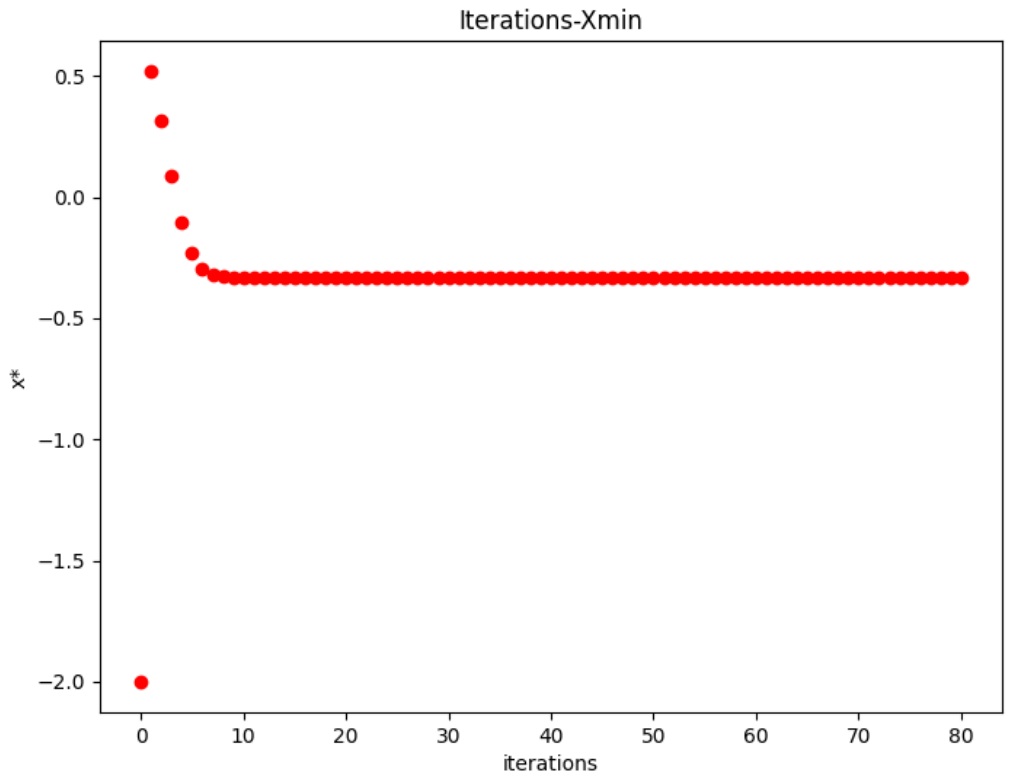
```
Choose function type (1 for F(x), 2 for G(x)): 1
Choose minimization method (1 for Gradient Descent, 2 for Newtons method): 1
Do you want batch/restart mode? (1 for yes, 0 for no): 0
Choose the way of definition of starting points (1 for directly, 2 from Uniform Distribution): 1
Choose the stopping condition (1 for number of iterations, 2 for desired value, 3 for computational time): 1
Declare the number of iterations: 80
declare the values (integers) of a, b, c and d: -1 1 1 1
Give starting (integer) point: -2
mean =  -0.3333333333333333  and the standard deviation =  0.8148148148148149

Process finished with exit code 0
```

Results:

The function's plot:



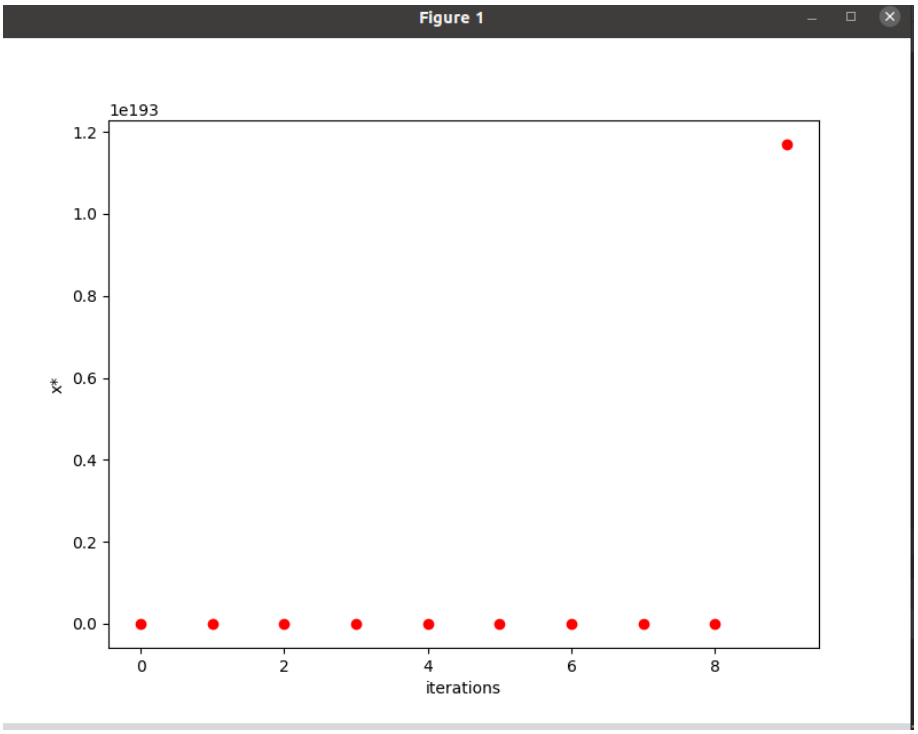
The x* in comparison with the iterations: (We notice that it reaches its minimum in very few iterations since the starting point was close to it)



It prints out the mean value of x* as well as the standard deviation of all the x*, our solutions:

```
mean =  -0.3333333333333333  and the standard deviation =  0.8148148148148149
```

If we begin from a different starting point :

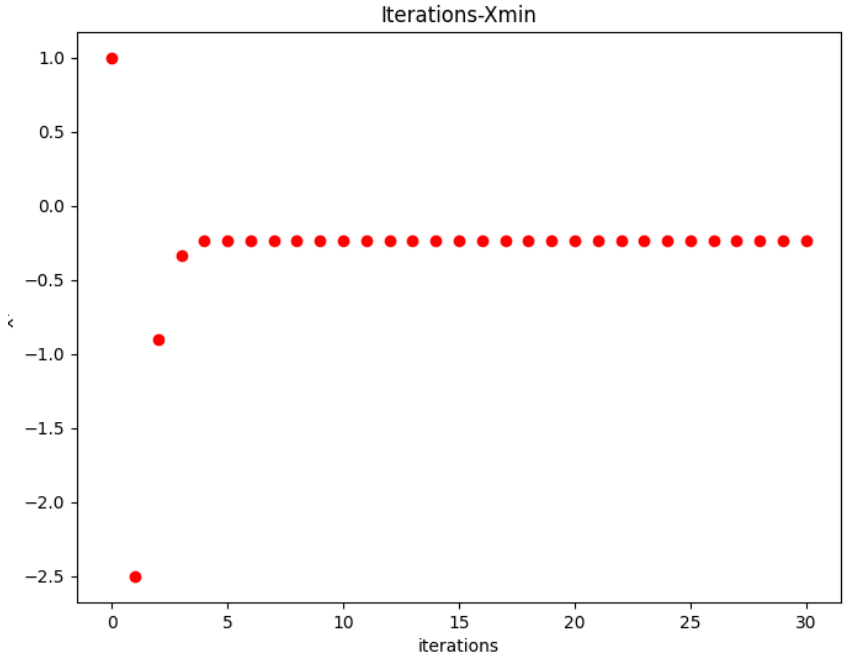


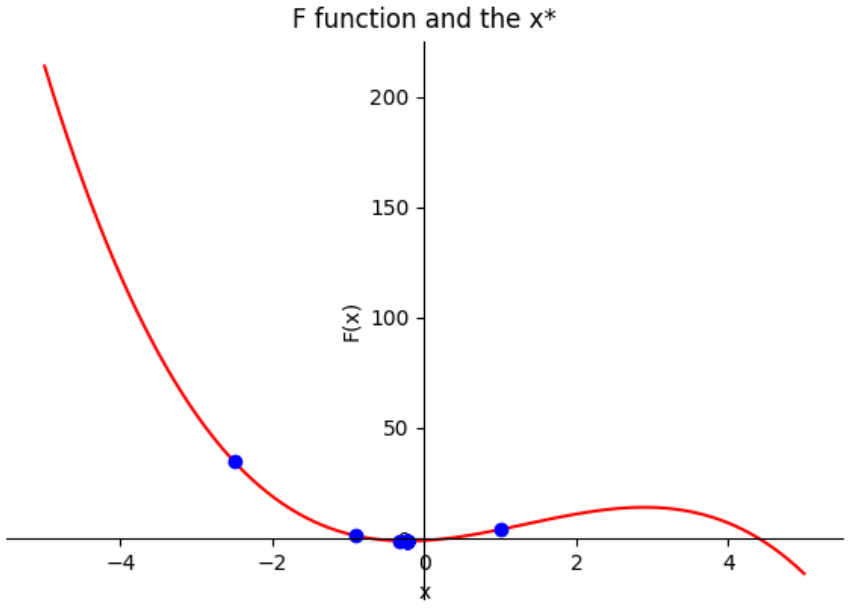
Note: Since the function is close to the x = 4 it decreases, finally, it reaches the inf. Above there is an array with the x* that this method finds:

```
10.552
62.956258816
2039.2341800038419
2097225.808107077
2216772862013.7217
2.476697288568767e+24
3.091550847438757e+48
4.817074067718846e+96
1.1694918097240253e+193
inf
nan
nan
```

Test2:

```
Choose function type (1 for F(x), 2 for G(x)): 1
Choose minimization method (1 for Gradient Descent, 2 for Newtons method): 2
Do you want batch/restart mode? (1 for yes, 0 for no): 0
Choose the way of definition of starting points (1 for directly, 2 from Uniform Distribution): 1
Choose the stopping condition (1 for number of iterations, 2 for desired value, 3 for computational time): 1
Declare the number of iterations: 30
declare the (int) values of a, b, c and d: -1 4 2 -1
Give starting point: 1
mean =  -0.23013858660780986  and the standard deviation =  -1.236233090082626
```



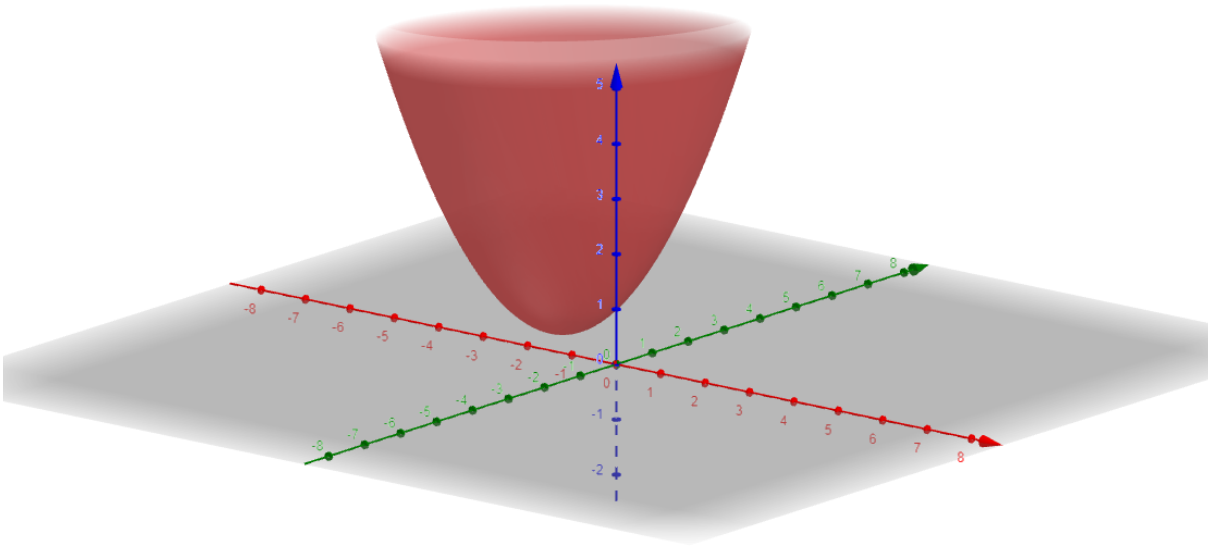


Test3:

```
Choose function type (1 for F(x), 2 for G(x)): 2
Choose minimization method (1 for Gradient Descent, 2 for Newtons method): 1
Do you want batch/restart mode? (1 for yes, 0 for no): 1
How many times to do the operation? 3
Choose the way of definition of starting points (1 for directly, 2 from Uniform Distribution): 1
Choose the stopping condition (1 for number of iterations, 2 for desired value, 3 for computational time): 3
Enter value c: 2
Enter d-dimensional vector b: 2 1
Enter matrix A (each row you add press Enter) :
2 4
-1 3
Give (integer) starting point: 1 3
Give computational time (in seconds): 4
Give (integer) starting point: 2 4
Give computational time (in seconds): 2
Give (integer) starting point: -1 2
Give computational time (in seconds): 3
mean =  [-0.6         0.13333333]  and the standard deviation =  [0. 0.]

Process finished with exit code 0
```

Test4:

G Function:



Algorithm

```
Choose function type (1 for F(x), 2 for G(x)): 2
Choose minimization method (1 for Gradient Descent, 2 for Newtons method): 2
Do you want batch/restart mode? (1 for yes, 0 for no): 1
How many times to do the operation? 3
Choose the way of definition of starting points (1 for directly, 2 from Uniform Distribution): 2
Define the range (give two int numbers: low high): -100 100
Choose the stopping condition (1 for number of iterations, 2 for desired value, 3 for computational time): 2
Enter value c: 7
Enter d-dimensional vector b: 5 -4
Enter matrix A:
1 -1
1 1
Give desired value: 1
Give desired value: 2
Give desired value: 1
mean =  [-2.5  2. ]  and the standard deviation =  [0. 0.]

Process finished with exit code 0
```

Notes:

With the gradient descent method the results are deeply connected and affected by the starting point and the solution is not always optimum. It also requires more iterations until it reaches the x* than Newton's and the route it follows is not always the best possible. It is also affected by the learning rate(LR), particularly high values of LR provide a bad estimation while the values closer to 1 approaches the best solution. Furthermore, if LR takes a high value then we could be led to oscillations and even divergence.

Points that need attention in the newton method are the risk of entrapment in local minimums. And we need to pay even more attention when the Hessian of f function is not a positively defined array. In these cases, the newton method does not apply and we use another method the Levenberg. Although the method of gradient descent guarantees almost certain convergence in contrast to Newton, it is not preferred because of its computational disadvantages, one of which is the calculation of step LR, where it takes considerable computational time to calculate if we do not take it as a constant number and we try to calculate it. That is why an alternative is the newton method which can save time but is not always applied.

In addition, in Newton's method, we notice that if the function is quadratic as in the case of the F, the iterations needed to reach the minimum x* are reduced to 1 iteration as it is shown below :

$$G(x) = c + b^T x + x^T A x$$

$$b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}, \qquad A = \begin{bmatrix} a_{11} & \cdots & a_{1d} \\ \vdots & \ddots & \vdots \\ a_{d1} & \cdots & a_{dd} \end{bmatrix}, \qquad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$$G(x) = c + b_1 x_1 + b_2 x_2 + \cdots + b_d x_d + a_{11} {x_1}^2 + a_{12} x_1 x_2 + a_{21} x_2 x_1 + \cdots + a_{dd} {x_d}^2$$

$$\nabla G(x) = \begin{bmatrix} b_1 + 2a_{11}x_1 + (a_{12} + a_{21})x_2 + \cdots + (a_{1d} + a_{d1})x_d \\ b_2 + (a_{12} + a_{21})x_1 + 2a_{22}x_2 + \cdots + (a_{2d} + a_{d2})x_d \\ \vdots \\ b_d + (a_{1d} + a_{d1})x_d + (a_{2d} + a_{d2})x_2 + \cdots + 2a_{dd}x_d \end{bmatrix}$$

$$\nabla^2 G(x) = \begin{bmatrix} 2a_{11} & (a_{12} + a_{21}) & \cdots & (a_{1d} + a_{d1}) \\ (a_{12} + a_{21}) & 2a_{22} & \cdots & (a_{2d} + a_{d2}) \\ \vdots & \vdots & & \vdots \\ (a_{1d} + a_{d1}) & (a_{2d} + a_{d2}) & \cdots & 2a_{dd} \end{bmatrix}$$

$$\nabla G(x) = b + \nabla^2 G(x) * x$$

$$\nabla^2 G(x) = A + A^T$$

$$\begin{aligned} x_{k+1} &= x_k - \nabla^2 G(x_k)^{-1} * \nabla G(x_k) = x_k - \nabla^2 G(x_k)^{-1} * (b + \nabla^2 G(x_k) * x_k) \\ &= x_k - \nabla^2 G(x_k)^{-1} * b - \nabla^2 G(x_k)^{-1} * \nabla^2 G(x_k) * x_k = x_k - \nabla^2 G(x_k)^{-1} * b - x_k \\ &= -\nabla^2 G(x_k)^{-1} * b = -(A + A^T)^{-1} * b \end{aligned}$$

$$x_{k+1} = -(A + A^T)^{-1} * b$$