Numerical Linear Algebra Homework Project 4: Unconstrained Optimization

Catalano Giuseppe, Cerrato Nunzia

In this project we want to perform unconstrained optimization using the Newton method both in its standard form and by considering its variants which make use of the backtracking and the trust region approach. The first function that we report is called `Newton` and it performs the standard Newton algorithm, with the possibility of using backtracking for the choice of α if `backtracking=True` is passed as keyword argument to the function. The second function is called `Newton_trust_region` and it performs the Newton algorithm with the trust region approach. These two functions are presented with their own documentation and they can be found in the library `Project_4.py` on GitHub. We will apply these methods to four test functions and, in some cases, we will compare these approaches to see how convergence changes.

1 Algorithms

```
def Newton(func, grad, hess, tol, maxit, x_0, sol_x, sol_f, alpha=1,
↪  sigma=0.0001, rho=0.5, backtracking=False):
    r''' This function implements the standard Newton method for
    ↪  unconstrained optimization.

     Parameters
     ----------
     func : function
         Function to be minimized. It must be :math:`f : \mathbb{R}^{n}
         ↪  \rightarrow \mathbb{R}`
     grad : function
         Gradient of the function. It returns a 1d-array (vector)
     hess : function
         Hessian of the function. It returns a 2d-array (matrix)
     tol : float
         Tolerance parameter for the stopping criterion
     maxit : int
         Maximum number of iterations
     x_0 : ndarray
         Starting point
     sol_x : ndarray
         Exact solution (x) to the minimization problem
     sol_f : float
         Exact minimum value of the function
     alpha : float
         Step lenght. Default value alpha=1
     sigma : float
```

```
25          Constant parameter in :math:`(0,1)`. Used if backtracking = True.
            ↪ Default value sigma=0.0001
26      rho : float
27          Reduction parameter in :math:`(0,1)`. Used if backtracking = True.
            ↪ Default value rho=0.5
28
29      Results
30      -------
31      results : dict
32          Dictonary of the results given by the function. It contains the
            ↪ following items:
33            - 'convergence' : (bool) True if the algorithm converges, False
              ↪ if it doesn't converge
34            - 'k' : (int) final iteration at which convergence is reached
35            - 'min_point' : (ndarray) computed point at which the minimum of
              ↪ the function is reached
36            - 'min_value' : (float) computed minimum value of the function
37            - 'interm_point' : (list) list of the intermediate points
38            - 'error_x' : (list) list that contains the 2-norm of the
              ↪ difference between each intermediate point and the exact
              ↪ solution (min_point).
39            - 'error_f' : (list) list that contains the difference between
              ↪ the function evaluated at each intermediate point and its
              ↪ value in the exact minimum point
40            - 'scalar_product' : (list) list that contains the scalar product
              ↪ between the descent direction and the gradient
41
42      '''
43      old_point = x_0
44      alpha_0 = alpha
45
46      # Create a list to save intermediate points
47      interm_points = [old_point]
48      # Create a list to save the scalar product between the gradient and the
        ↪ descent direction
49      scalar_prod = []
50
51      new_point = x_0
52      norm_diff_x = np_lin.norm(new_point - old_point)
53      # Cycle on the number of iterations
54      for k in range(maxit):
55          gradient = grad(old_point)
56          hessian = hess(old_point)
57
58          # Compute norms for the stopping criterion
59          norm_grad = np_lin.norm(gradient)
60
61          # Check if the stopping criterion is satisfied
62          if norm_grad <= tol and norm_diff_x <= tol*(1 +
            ↪ np_lin.norm(old_point)):
63              min_value = func(new_point)
```

```
            conv = True
            break

        # Compute the descent direction by solving a linear system
        p = np_lin.solve(hessian, -gradient)
        scalar_prod.append(gradient@p)

        # Implement backtracking if backtracking == True
        if backtracking == True:
            alpha = alpha_0
            iteraz_backtracking = 0
            while func(old_point + alpha*p) > func(old_point) +
            ↪  (sigma*alpha)*(p @ gradient) and iteraz_backtracking < 100:
                alpha = rho*alpha
                iteraz_backtracking += 1

        # Compute the new point and add it to the list of intermediate
        ↪  points
        new_point = old_point + alpha*p
        interm_points.append(new_point)
        norm_diff_x = np_lin.norm(new_point - old_point)

        old_point = new_point

    if k == maxit-1:
        min_value = func(new_point)
        conv = False

    gradient = grad(new_point)
    hessian = hess(new_point)
    p = np_lin.solve(hessian, -gradient)
    scalar_prod.append(gradient@p)

    # Compute the 2norm of the difference between each intermediate point
    ↪  and the exact solution
    error_x = [np_lin.norm(interm_x - sol_x) for interm_x in interm_points]

    # Compute the difference between the function evaluated in each
    ↪  intermediate point and its value in the exact minimum point
    error_f = [func(interm_x) - sol_f for interm_x in interm_points]

    results = {'convergence': conv, 'k' : k, 'min_point' : new_point,
    ↪  'min_value' : min_value , 'interm_point' : interm_points, 'error_x'
    ↪  : error_x, 'error_f' : error_f, 'scalar_product' : scalar_prod}

    return results

```

Below the code of the trust region Newton algorithm.

```
def Newton_trust_region(func, grad, hess, tol, maxit, x_0, sol_x, sol_f,
↪  alpha=1, eta=0.01):
```

```
    ''' This function implements the Newton method with the trust region approach for unconstrained optimization.

    Parameters
    ----------
    func : function
        Function to be minimized. It must be :math:`f : \mathbb{R}^{n} \rightarrow \mathbb{R}`.
    grad : function
        Gradient of the function. It returns a 1d-array (vector)
    hess : function
        Hessian of the function. It returns a 2d-array (matrix)
    tol : float
        Tolerance parameter for the stopping criterion
    maxit : int
        Maximum number of iterations
    x_0 : ndarray
        Starting point
    sol_x : ndarray
        Exact solution (x) to the minimization problem
    sol_f : float
        Exact minimum value of the function
    alpha : float
        Step lenght. Default value alpha=1
    eta : float
        Constant parameter in :math:`(0,0.25)`. Default value eta=0.01

    Results
    -------
    results : dict
        Dictonary of the results given by the function. It contains the following items:
          - 'convergence' : (bool) True if the algorithm converges, False if it doesn't converge
          - 'k' : (int) final iteration at which convergence is reached
          - 'min_point' : (ndarray) computed point at which the minimum of the function is reached
          - 'min_value' : (float) computed minimum value of the function
          - 'interm_point' : (list) list of the intermediate points
          - 'error_x' : (list) list that contains the 2-norm of the difference between each intermediate point and the exact solution (min_point).
          - 'error_f' : (list) list that contains the difference between the function evaluated at each intermediate point and its value in the exact minimum point
          - 'scalar_product' : (list) list that contains the scalar product between the descent direction and the gradient
    '''

    # Evaluate the gradient and the hessian in the starting point
    gradient = grad(x_0)
```

```
    hessian = hess(x_0)

    # Compute the fist descent direction and the first delta value
    p = np_lin.solve(hessian, -gradient)
    delta = np_lin.norm(p)
    interm_radius = [delta]

    interm_points = [x_0]
    scalar_prod = []
    old_point = x_0

    # Cycle on the number of iterations
    for k in range(maxit):

        # Evaluate the gradient and the hessian at the current point
        gradient = grad(old_point)
        hessian = hess(old_point)

        # Diagonalize the hessian computed at the current point
        eigval, eigvect = np_lin.eigh(hessian)
        # Choose an initial mu value
        mu = abs(min(min(eigval), 0)) + 1e-12
        coeff_vect = eigvect.T @ gradient/(eigval + mu)

        # Choose the optimal mu value which respects the condition on the
        ↪  2-norm
        while sum([coeff**2 for coeff in coeff_vect]) > delta**2:
            mu = mu*2
            coeff_vect = eigvect.T @ gradient/(eigval + mu)

        # Compute the descent direction
        p = - eigvect @ coeff_vect
        scalar_prod.append(- gradient @ eigvect @ np.diag(1/eigval) @
        ↪  eigvect.T @ gradient)

        new_point = old_point + alpha*p

        # Choose the value of delta for the successive iteration
        rho = (func(new_point)-func(old_point))/(p @ gradient + 0.5*p @
        ↪  hessian @ p)

        if 0 < rho < 0.25:
            delta = delta/4
        elif rho > 0.75:
            delta = 2*delta
        elif 0 < rho < eta:
            new_point = old_point

        interm_points.append(new_point)

        # Compute norms for the stopping criterion
```

```
        norm_grad = np_lin.norm(gradient)
        norm_diff_x = np_lin.norm(new_point - old_point)

        # Check if the stopping criterion is satisfied
        if norm_grad <= tol and norm_diff_x <=tol*(1 +
        ↪  np_lin.norm(new_point)):
            min_value = func(new_point)
            conv = True
            break

        old_point = new_point

    if k == maxit-1:
        min_value = func(new_point)
        conv = False

    gradient = grad(new_point)
    hessian = hess(new_point)
    p = np_lin.solve(hessian, -gradient)
    scalar_prod.append(- gradient @ eigvect @ np.diag(1/eigval) @ eigvect.T
    ↪  @ gradient)

    # Compute the 2norm of the difference between each intermediate point
    ↪  and the exact solution
    error_x = [np_lin.norm(interm_x - sol_x) for interm_x in interm_points]

    # Compute the difference between the function evaluated in each
    ↪  intermediate point and
    # its value in the exact minimum point
    error_f = [func(interm_x) - sol_f for interm_x in interm_points]

    results = {'convergence': conv, 'k' : k, 'min_point' : new_point,
    ↪  'min_value' : min_value, 'interm_point' : interm_points, 'error_x' :
    ↪  error_x, 'error_f' : error_f, 'scalar_product' : scalar_prod}

    return results
```

2 Test Functions

We will consider four test functions, which will be distinguished by using an apex (a), (b), (c), (d), respectively. For each of these functions, we will try to minimize it first by using the standard Newton algorithm and, if it does not converge, we will try to use backtracking or, as a last attempt, the trust region approach.

In all of these cases, we have chosen as tolerance parameter `tol=1e-12` and as maximum number of iterations allowed `maxit=100`. Moreover, we have chosen `alpha=1` as default α value and `sigma=0.0001` and `rho=0.5` as default parameters for backtracking. Finally, in the case of the Newton trust region method, we have chosen `eta=0.01`. However, these constants can be passed as keyword argument to the appropriate functions, therefore one can change their value in the main program, where these functions are effectively used.

In each case we will report a table with the required values in correspondence with the intermediate points (note that $k = 0$ will correspond to the starting point) and a contour plot and a 3D plot to show how the minimum point is reached.

Test function (a)

Consider the function $f^{(a)} : \mathbb{R}^2 \to \mathbb{R}$, defined as

$$f^{(a)}(x_1, x_2) = (x_1 - 2)^4 + (x_1 - 2)^2 x_2^2 + (x_2 + 1)^2. \tag{1}$$

It is possible to note that this function assumes a minimum value $f^{(a)}(\mathbf{x}^*) = 0$ in correspondence with the point $\mathbf{x}^* = (2, -1)$. We would like to obtain this minimum value and the corresponding point $\mathbf{x}^*$ by using the standard Newton algorithm implemented by the function `Newton`. We report below the gradient and the Hessian of $f^{(a)}(\mathbf{x})$, which must be passed as arguments to the Python function which performs the minimization.

$$\nabla f^{(a)}(x_1, x_2) = \begin{bmatrix} 4(x_1 - 2)^3 + 2x_2^2(x_1 - 2) \\ 2x_2(x_1 - 2)^2 + 2x_2(x_2 - 2) \end{bmatrix}, \tag{2}$$

$$\nabla^2 f^{(a)}(x_1, x_2) = \begin{bmatrix} 12(x_1 - 2)^2 + 2x_2^2 & 4x_2(x_1 - 2) \\ 4x_2(x_1 - 2) & 2(x_1 - 2)^2 + 2 \end{bmatrix}. \tag{3}$$

We consider first the starting point $x_0 = (1, 1)^T$. The results that we obtained are the following:

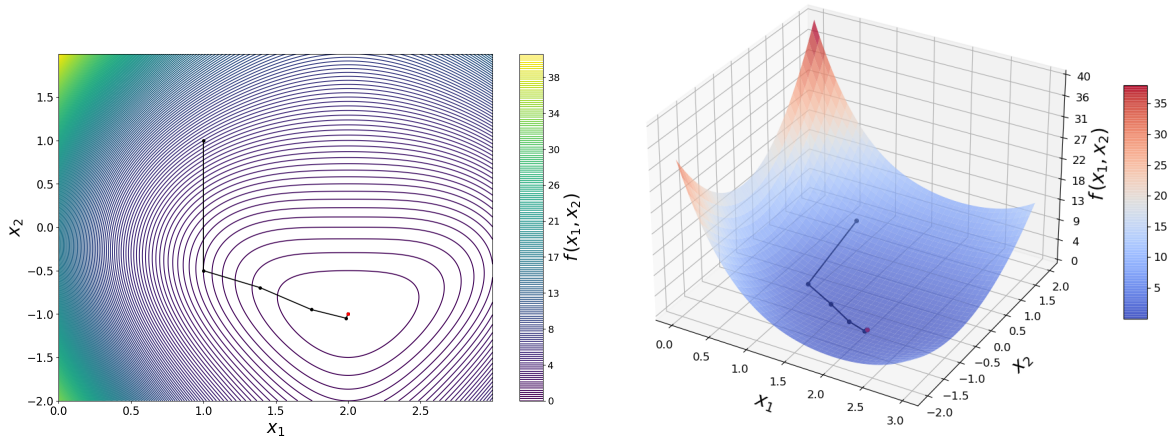
```
convergence = True, with 8 steps
min point = [ 2. -1.]
min value = 0.0
```

As we can see, the algorithm converges in 8 steps and it returns the right minimum point and the corresponding minimum value of the function. We report in Tab. 1 the required quantities computed at each step.

k	$\|\mathbf{x}_k - \mathbf{x}^*\|_2$	$f^{(a)}(\mathbf{x}_k) - f^{(a)}(\mathbf{x}^*)$	$-\nabla f^{(a)}(\mathbf{x}_k)^T [\nabla^2 f^{(a)}(\mathbf{x}_k)]^{-1} \nabla f^{(a)}(\mathbf{x}_k)$
0	2.236	6.000	-9.000
1	1.118	1.500	-1.761
2	6.805×10^{-1}	4.092×10^{-1}	-5.552×10^{-1}
3	2.592×10^{-1}	6.489×10^{-2}	-1.237×10^{-1}
4	5.012×10^{-2}	2.531×10^{-3}	-5.026×10^{-3}
5	1.277×10^{-3}	1.632×10^{-6}	-3.262×10^{-6}
6	1.659×10^{-6}	2.754×10^{-12}	-5.508×10^{-12}
7	1.404×10^{-12}	1.971×10^{-24}	-3.943×10^{-24}
8	0	0	0

Table 1: Table of data obtained using the standard Newton method to minimize the function $f^{(a)}(x_1, x_2)$ by using $\alpha = 1$ and by considering as starting point $\mathbf{x}_0 = (1, 1)^T$.

We report in Fig. 1 the contour plot and the 3D plot, where it is possible to see how the minimum point (in red) is reached.



(a) (b)

Figure 1: Contour plot (Panel (a)) and 3D plot (Panel (b)) of the function $f^{(a)}(x_1, x_2)$ where the intermediate points and the minimum point (in red) are obtained by using the standard Newton algorithm starting from the point $x_0 = (1, 1)^T$.

The second starting point which we consider is $x_0 = (2, -1)^T$. This is exactly the minimum point of the function, therefore we expect the algorithm to converge in 0 steps. The results that we obtained are the following:

```
convergence = True, with 0 steps
min point = [ 2 -1]
min value = 0
```

2.1 (b)

Consider the function $f^{(b)} : \mathbb{R}^4 \to \mathbb{R}$, defined as $f^{(b)}(\mathbf{x}) = \mathbf{b}^T + \frac{1}{2}\mathbf{x}^T H\mathbf{x}$, where:

$$\mathbf{b} = (5.04, -59.4, 146.4, -96.6)^T, \tag{4}$$

$$H = \begin{bmatrix} 0.16 & -1.2 & 2.4 & -1.4 \\ -1.2 & 12.0 & -27.0 & 16.8 \\ 2.4 & -27.0 & 64.8 & -42.0 \\ -1.4 & 16.8 & -42.0 & 28.0 \end{bmatrix}. \tag{5}$$

We know that the minimum point is $\mathbf{x}^* = (1, 0, -1, 2)^T$ and the minimum value of the function is $f^{(b)}(\mathbf{x}^*) = -167.28$. Moreover, in this case we have that $\nabla f^{(b)}(\mathbf{x}) = H\mathbf{x} + \mathbf{b}$ and $\nabla^2 f^{(b)}(\mathbf{x}) = H$. Since this is a quadratic function, we expect the standard Newton algorithm to converge in 1 step, regardless the starting point. In this case, we consider as starting point $x_0 = (-1, 3, 3, 0)^T$ and the results that we obtained are the following:

```
convergence = True, with 2 steps
min point = [ 1.000e+00 -4.730e-13 -1.000e+00  2.000e+00]
min value = -167.28
```

Below is the table with the required quantities at each step.

k	$\|\mathbf{x}_k - \mathbf{x}^*\|_2$	$f^{(b)}(\mathbf{x}_k) - f^{(b)}(\mathbf{x}^*)$	$-\nabla f^{(b)}(\mathbf{x}_k)^T[\nabla^2 f^{(b)}(\mathbf{x}_k)]^{-1}\nabla f^{(b)}(\mathbf{x}_k)$
0	5.745	5.223×10^{2}	-1.045×10^{3}
1	9.769×10^{-13}	2.842×10^{-14}	-9.948×10^{-27}
2	1.688×10^{-13}	0.000	-2.005×10^{-26}

Table 2: Table of data obtained using the standard Newton method to minimize the function $f^{(b)}(\mathbf{x})$ by using $\alpha = 1$ and by considering as starting point $x_0 = (-1, 3, 3, 0)^T$.

The second, in principle unexpected, step is due to the convergence criterion. Recall that we have chosen as tolerance parameter `tol=1e-12` and we have requested that two successive points must be close *enough*, where enough means that it must be satisfied the condition $\|\mathbf{x}_k - \mathbf{x}_{k-1}\|_2 \leq tol(1 + \|\mathbf{x}_k\|_2)$.

2.2 (c)

Consider the function $f : \mathbb{R}^2 \to \mathbb{R}$, defined as

$$f^{(c)}(x_1, x_2) = (1.5 - x_1(1 - x_2))^2 + (2.25 - x_1(1 - x_2^2))^2 + (2.625 - x_1(1 - x_2^3))^2. \quad (6)$$

This function assumes a minimum value $f^{(c)}(\mathbf{x}^*) = 0$ in correspondence with the point $\mathbf{x}^* = (3, 0.5)^T$. The gradient and the Hessian of this function can be obtained, by linearity, by computing the gradient and the Hessian of each of the three terms in the expression of $f^{(c)}(\mathbf{x})$ and then summing them up.

We would like to obtain the minimum point and the minimum value of the function by using, as a first attempt, the standard Newton algorithm.

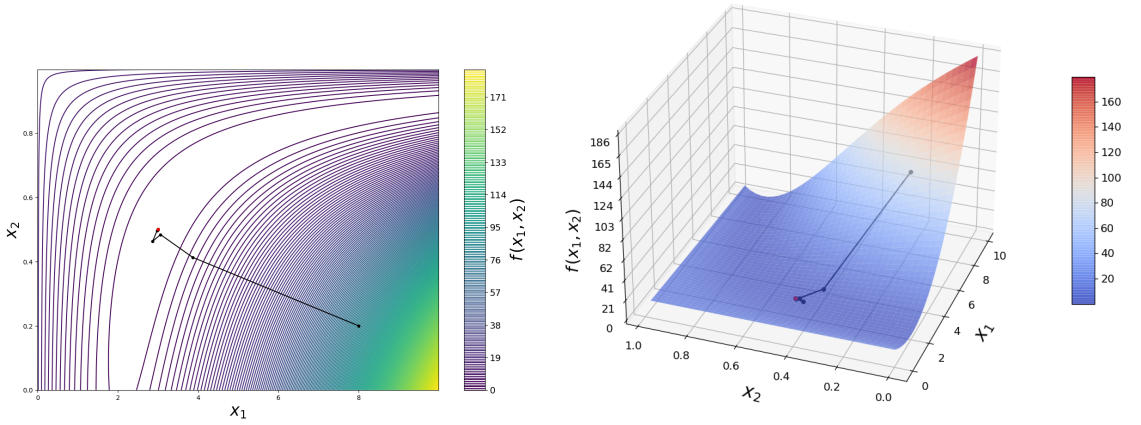
The first point that we consider is $\mathbf{x}_0 = (8, 0.2)^T$. The results that we obtained are the following:

```
convergence = True, with 9 steps
min point = [3.  0.5]
min value = 0.0
```

We report in Tab 3 the required quantities at each step of the algorithm.

k	$\|\mathbf{x}_k - \mathbf{x}^*\|_2$	$f^{(c)}(\mathbf{x}_k) - f^{(c)}(\mathbf{x}^*)$	$-\nabla f^{(c)}(\mathbf{x}_k)^T[\nabla^2 f^{(c)}(\mathbf{x}_k)]^{-1}\nabla f^{(c)}(\mathbf{x}_k)$
0	5.009	8.17×10^{1}	-1.455×10^{2}
1	8.66×10^{-1}	2.423	-4.527
2	6.494×10^{-2}	2.407×10^{-2}	-4.643×10^{-2}
3	1.393×10^{-1}	3.45×10^{-3}	-6.32×10^{-3}
4	2.103×10^{-2}	1.383×10^{-4}	-2.704×10^{-4}
5	1.377×10^{-3}	2.863×10^{-7}	-5.717×10^{-7}
6	3.033×10^{-6}	2.186×10^{-12}	-4.372×10^{-12}
7	2.836×10^{-11}	1.233×10^{-22}	-2.466×10^{-22}
8	4.441×10^{-16}	4.437×10^{-31}	-8.79×10^{-31}
9	0.000	0.000	0.000

Table 3: Table of data obtained using the standard Newton algorithm to minimize the function $f^{(c)}(x_1, x_2)$ by using $\alpha = 1$ and by considering as starting point $x_0 = (8, 0.2)^T$.



(a) (b)

Figure 2: Contour plot (Panel (a)) and 3D plot (Panel (b)) of the function $f^{(c)}(x_1, x_2)$ where the intermediate points and the minimum point (in red) are obtained by using the standard Newton algorithm starting from the point $x_0 = (8, 0.2)^T$.

As can be seen from the obtained results, the algorithm reaches convergence in 9 step. This can be also seen by considering the quantities reported in Tab. 3 and the contour plot and the 3D plot reported in Fig. 2.

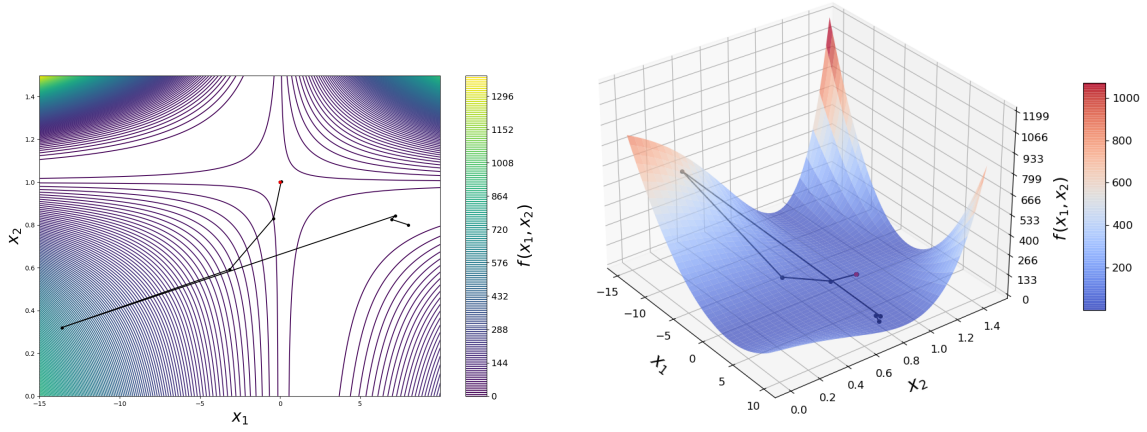
We now consider as starting point $x_0 = (8, 0.8)^T$. In this case we obtained the following results:

```
convergence = True, with 10 steps
min point = [0. 1.]
min value = 14.203125
```

As we can see, the algorithm converges in 10 step to the point $\tilde{\mathbf{x}} = (0, 1)^T$ which, however, is not the minimum point of the function (which we know to be equal to $\mathbf{x}^* = (3, 0.5)^T$). This emerges also from the data reported in Tab. 4, where one can see that the 2-norm of the difference between the computed minimum point at step k and the exact minimum point stabilizes to a non-zero value, as well as the difference between the value of the function at the point $\mathbf{x}_k$ and the minimum value $f(\mathbf{x}^*)$. What happens in this case is that the algorithm converges to a saddle point. In fact, in this point the gradient of the function vanishes, i.e. $\nabla f^{(c)}(\tilde{\mathbf{x}}) = \mathbf{0}$ (and this also reflects on the last points in Tab. 4, since the scalar product between the gradient and the descent direction becomes equal to zero), while the Hessian is indefinite.

k	$\|\mathbf{x}_k - \mathbf{x}^*\|_2$	$f^{(c)}(\mathbf{x}_k) - f^{(c)}(\mathbf{x}^*)$	$-\nabla f^{(c)}(\mathbf{x}_k)^T[\nabla^2 f^{(c)}(\mathbf{x}_k)]^{-1}\nabla f^{(c)}(\mathbf{x}_k)$
0	5.009	2.043	-3.291
1	3.963	2.553×10^{-1}	-5.885×10^{-2}
2	4.218	2.328×10^{-1}	-6.421×10^{-1}
3	1.661×10^{1}	5.743×10^{2}	-9.291×10^{2}
4	6.137	5.226×10^{1}	-6.327×10^{1}
5	3.426	1.596×10^{1}	-3.709
6	2.974	1.421×10^{1}	-8.445×10^{-3}
7	3.041	1.42×10^{1}	-5.688×10^{-8}
8	3.041	1.42×10^{1}	-1.083×10^{-18}
9	3.041	1.42×10^{1}	0.000
10	3.041	1.42×10^{1}	0.000

Table 4: Table of data obtained using the standard Newton algorithm to minimize the function $f^{(c)}(x_1, x_2)$ by using $\alpha = 1$ and by considering as starting point $x_0 = (8, 0.8)^T$.



(a) (b)

Figure 3: Contour plot (Panel (a)) and 3D plot (Panel (b)) of the function $f^{(c)}(x_1, x_2)$ where the intermediate points and the minimum point (in red) are obtained by using the standard Newton algorithm starting from the point $x_0 = (8, 0.2)^T$.

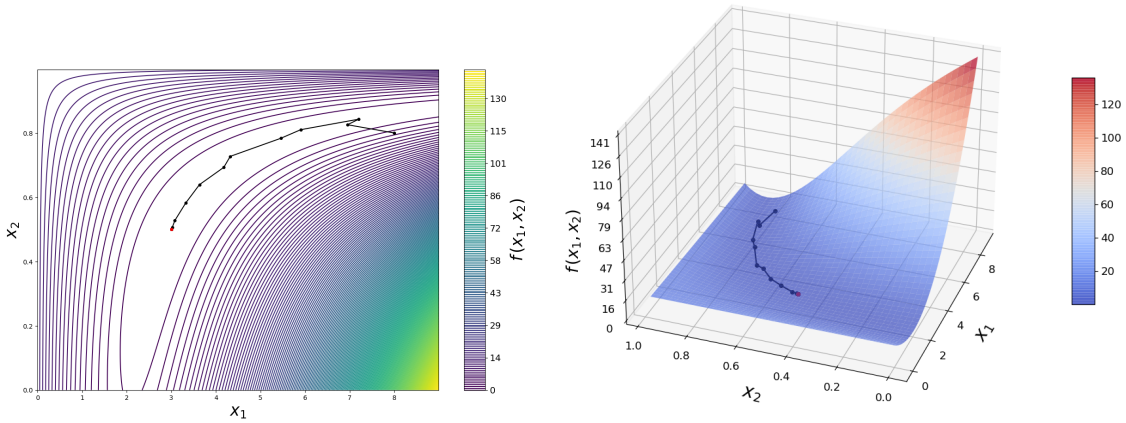
To solve this problem, we tried to minimize the function $f^{(c)}(x_1, x_2)$ by using the Newton algorithm with backtracking. In this case, we obtained:

```
convergence = True, with 14 steps
min point = [3.  0.5]
min value = 0.0
```

We can see that the algorithm converges to the expected minimum point in 14 steps. We report in Tab. 5 the required quantities at each step and in Fig. 4 the obtained contour plot and the 3D plot.

k	$\|\mathbf{x}_k - \mathbf{x}^*\|_2$	$f^{(c)}(\mathbf{x}_k) - f^{(c)}(\mathbf{x}^*)$	$-\nabla f^{(c)}(\mathbf{x}_k)^T[\nabla^2 f^{(c)}(\mathbf{x}_k)]^{-1}\nabla f^{(c)}(\mathbf{x}_k)$
0	5.009	2.043	-3.291
1	3.963	2.553×10^{-1}	-5.885×10^{-2}
2	4.218	2.328×10^{-1}	-6.421×10^{-1}
3	2.920	2.131×10^{-1}	-7.337×10^{-2}
4	2.471	1.649×10^{-1}	-1.224×10^{-1}
5	1.340	1.502×10^{-1}	-1.22×10^{-1}
6	1.192	7.989×10^{-2}	-1.697×10^{-1}
7	6.453×10^{-1}	5.012×10^{-2}	-5.027×10^{-2}
8	3.335×10^{-1}	1.73×10^{-2}	-2.418×10^{-2}
9	8.357×10^{-2}	3.009×10^{-3}	-5.269×10^{-3}
10	2.128×10^{-2}	1.004×10^{-4}	-1.967×10^{-4}
11	2.767×10^{-4}	1.503×10^{-7}	-3.005×10^{-7}
12	1.103×10^{-6}	2.293×10^{-13}	-4.585×10^{-13}
13	2.404×10^{-13}	8.166×10^{-25}	-1.633×10^{-24}
14	0.000	0.000	0.000

Table 5: Table of data obtained using the Newton algorithm with backtracking to minimize the function $f^{(c)}(x_1, x_2)$ by using $\alpha = 1$ and by considering as starting point $x_0 = (8, 0.8)^T$.



(a) (b)

Figure 4: Contour plot (Panel (a)) and 3D plot (Panel (b)) of the function $f^{(c)}(x_1, x_2)$ where the intermediate points and the minimum point (in red) are obtained by using the Newton algorithm with backtracking, starting from the point $x_0 = (8, 0.8)^T$.

2.3 (d)

Consider the function $f : \mathbb{R}^2 \to \mathbb{R}$, defined as $f(x_1, x_2) = x_1^4 + x_1 x_2 + (1 + x_2)^2$. This function assumes a minimum value $f^{(d)}(\mathbf{x}^*) \simeq -0.582445174$ in correspondence with the point $\mathbf{x}^* \simeq (0.695884386, -1.34794219)^T$. We report below the gradient and the Hessian of $f^{(d)}(\mathbf{x})$, which must be passed as arguments to the Python function which performs the minimization.

$$\nabla f^{(d)}(\mathbf{x}) = \begin{bmatrix} 4x_1^3 + x_2 \\ x_1 + 2(1 + x_2) \end{bmatrix}, \qquad \nabla^2 f^{(d)}(\mathbf{x}) = \begin{bmatrix} 12x_1^2 & 1 \\ 1 & 2 \end{bmatrix}. \tag{7}$$

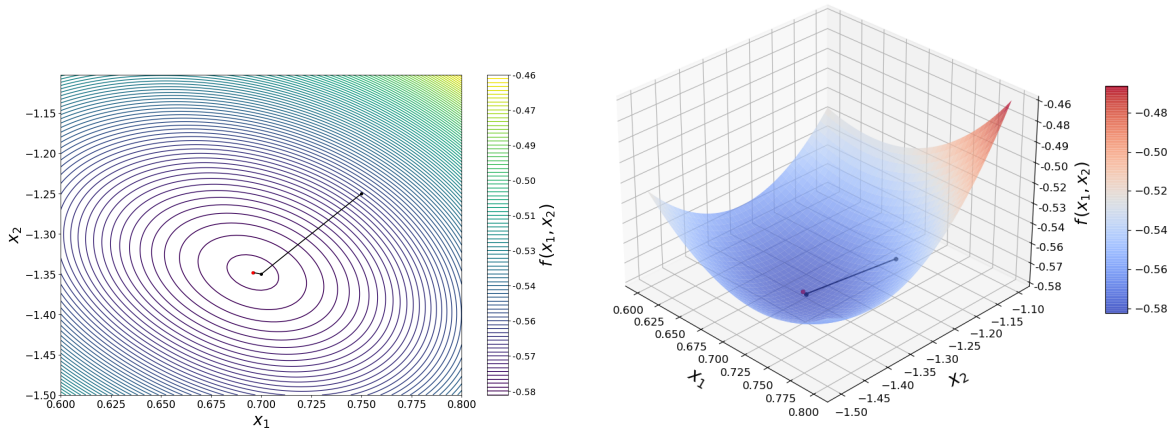
We consider first the starting point $x_0 = (0.75, -1.25)^T$ and we try to see if the standard Newton algorithm converges, obtaining the following results:

```
convergence = True, with 5 steps
min point = [ 0.69588439 -1.34794219]
min value = -0.5824451744436351
```

In this case, the algorithm converges in 5 steps. We report in Tab. 6 the required quantities and in Fig. 5 the contour plot and the 3D plot.

k	$\|\mathbf{x}_k - \mathbf{x}^*\|_2$	$f^{(d)}(\mathbf{x}_k) - f^{(d)}(\mathbf{x}^*)$	$-\nabla f^{(d)}(\mathbf{x}_k)^T[\nabla^2 f^{(d)}(\mathbf{x}_k)]^{-1}\nabla f^{(d)}(\mathbf{x}_k)$
0	1.119×10^{-1}	2.385×10^{-2}	-4.688×10^{-2}
1	4.601×10^{-3}	4.517×10^{-5}	-8.996×10^{-5}
2	2.951×10^{-5}	1.406×10^{-9}	-3.7×10^{-9}
3	3.805×10^{-9}	-4.436×10^{-10}	-6.372×10^{-18}
4	3.061×10^{-9}	-4.436×10^{-10}	0.000
5	3.061×10^{-9}	-4.436×10^{-10}	0.000

Table 6: Table of data obtained using the standard Newton algorithm to minimize the function $f^{(d)}(x_1, x_2)$ by using $\alpha = 1$ and by considering as starting point $x_0 = (0.75, -1.25)^T$.



(a) (b)

Figure 5: Contour plot (Panel (a)) and 3D plot (Panel (b)) of the function $f^{(c)}(x_1, x_2)$ where the intermediate points and the minimum point (in red) are obtained by using the standard Newton algorithm with $\alpha = 1$, starting from the point $x_0 = (8, 0.2)^T$.

We now consider as starting point $x_0 = (0, 0)^T$. In this case we can see that the standard Newton algorithm fails to converge, indeed

```
convergence = False, with 100 steps
min point = [ 0.00169883 -1.00084941]
min value = -0.0016995470778935944
```

As emerges from this results, after 100 iterations the algorithm does not reach the minimum point of the considered function. We report in Tab.7 the first and the last values of the required quantities. From these results we can see that the 2-norm of the difference between the computed minimum point at step k and the exact minimum point has a sort of periodic pattern, since it decreases every five iterations and then increases again, and this also

k	$\|\mathbf{x}_k - \mathbf{x}^*\|_2$	$f^{(d)}(\mathbf{x}_k) - f^{(d)}(\mathbf{x}^*)$	$-\nabla f^{(d)}(\mathbf{x}_k)^T[\nabla^2 f^{(d)}(\mathbf{x}_k)]^{-1}\nabla f^{(d)}(\mathbf{x}_k)$
0	1.517	1.582	0.000
1	3.014	1.758×10^1	-2.156×10^1
2	2.261	4.563	-4.537
3	1.736	1.796	-1.148
4	1.321	1.065	-6.339×10^{-1}
5	7.376×10^{-1}	5.459×10^{-1}	2.140
6	3.088	1.979×10^1	-2.447×10^1
7	2.311	5.017	-5.116
8	1.772	1.900	-1.262
9	1.353	1.101	-6.193×10^{-1}
10	8.159×10^{-1}	6.16×10^{-1}	1.988
11	3.077	1.945×10^1	-2.402×10^1
⋮	⋮	⋮	⋮
90	7.939×10^{-1}	5.966×10^{-1}	1.981
91	3.025	1.789×10^1	-2.197×10^1
92	2.268	4.627	-4.618
93	1.742	1.810	-1.164
94	1.326	1.070	-6.309×10^{-1}
95	7.501×10^{-1}	5.573×10^{-1}	2.081
96	3.048	1.859×10^1	-2.288×10^1
97	2.284	4.770	-4.800
98	1.753	1.843	-1.200
99	1.336	1.082	-6.254×10^{-1}
100	7.761×10^{-1}	5.807×10^{-1}	2.004

Table 7: Table of data obtained using the standard Newton algorithm to minimize the function $f^{(d)}(x_1, x_2)$ by using $\alpha = 1$ and by considering as starting point $x_0 = (0, 0)^T$.

happens when considering the difference between $f^{(d)}(\mathbf{x}_k)$ and $f^{(d)}(\mathbf{x}^*)$. This suggests that the computed point tries to slowly approach the exact minimum point but then moves away again, showing a sort of zig-zag pattern. In fact, if we see the scalar product between the gradient of the function and the descent direction we can observe that at the fifth iteration, and every five iterations, it becomes positive, suggesting that in the next step the new point will move away from the minimum point.

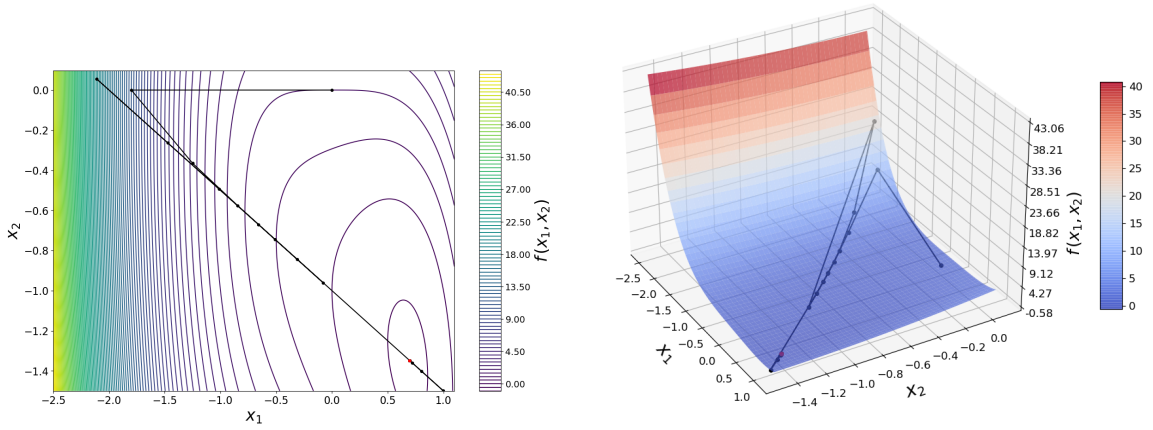
To reach convergence we first tried to use the standard Newton algorithm considering a different value for α. In particular, we observed that it is possible to reach convergence by choosing $\alpha = 0.9$ instead of $\alpha = 1$.

```
convergence = True, with 25 steps
min point = [ 0.69588439 -1.34794219]
min value = -0.5824451744436351
```

From these results we can see that the computed minimum point and the computed minimum value of the function coincide with the exact ones. However, in this case, the number of iterations required to reach convergence is 25, which is a much higher number if compared to the previous case.

k	$\|\mathbf{x}_k - \mathbf{x}^*\|_2$	$f^{(d)}(\mathbf{x}_k) - f^{(d)}(\mathbf{x}^*)$	$-\nabla f^{(d)}(\mathbf{x}_k)^T[\nabla^2 f^{(d)}(\mathbf{x}_k)]^{-1}\nabla f^{(d)}(\mathbf{x}_k)$
0	1.517	1.582	0.000
1	2.837	1.208×10^{1}	-1.432×10^{1}
2	2.181	3.888	-3.680
3	1.725	1.764	-1.114
4	1.352	1.099	-6.198×10^{-1}
5	8.656×10^{-1}	6.593×10^{-1}	2.174
6	3.138	2.143×10^{1}	-2.663×10^{1}
7	2.424	6.213	-6.657
8	1.910	2.391	-1.830
9	1.514	1.320	-6.987×10^{-1}
10	1.126	8.791×10^{-1}	-1.402
11	3.352×10^{-1}	3.219×10^{-1}	-5.271×10^{-1}
12	1.188×10^{-1}	3.348×10^{-2}	-6.1×10^{-2}
13	2.637×10^{-2}	1.514×10^{-3}	-2.957×10^{-3}
14	3.474×10^{-3}	2.572×10^{-5}	-5.128×10^{-5}
15	3.626×10^{-4}	2.789×10^{-7}	-5.586×10^{-7}
16	3.643×10^{-5}	2.375×10^{-9}	-5.637×10^{-9}
17	3.646×10^{-6}	-4.154×10^{-10}	-5.642×10^{-11}
18	3.659×10^{-7}	-4.434×10^{-10}	-5.643×10^{-13}
19	3.801×10^{-8}	-4.436×10^{-10}	-5.643×10^{-15}
20	5.778×10^{-9}	-4.436×10^{-10}	-5.643×10^{-17}
21	3.252×10^{-9}	-4.436×10^{-10}	-5.643×10^{-19}
22	3.079×10^{-9}	-4.436×10^{-10}	-5.643×10^{-21}
23	3.063×10^{-9}	-4.436×10^{-10}	-5.643×10^{-23}
24	3.061×10^{-9}	-4.436×10^{-10}	-5.644×10^{-25}
25	3.061×10^{-9}	-4.436×10^{-10}	-5.63×10^{-27}

Table 8: Table of data obtained using the standard Newton algorithm to minimize the function $f^{(d)}(x_1, x_2)$ by using $\alpha = 0.9$ and by considering as starting point $x_0 = (0, 0)^T$.



(a) (b)

Figure 6: Contour plot and 3D plot for the function $f^{(d)}(x_1, x_2)$ where the intermediate points and the minimum point (in red) are obtained by using the standard Newton method with $\alpha = 0.9$. The staring point considered in this case is $x_0 = (0, 0)^T$.

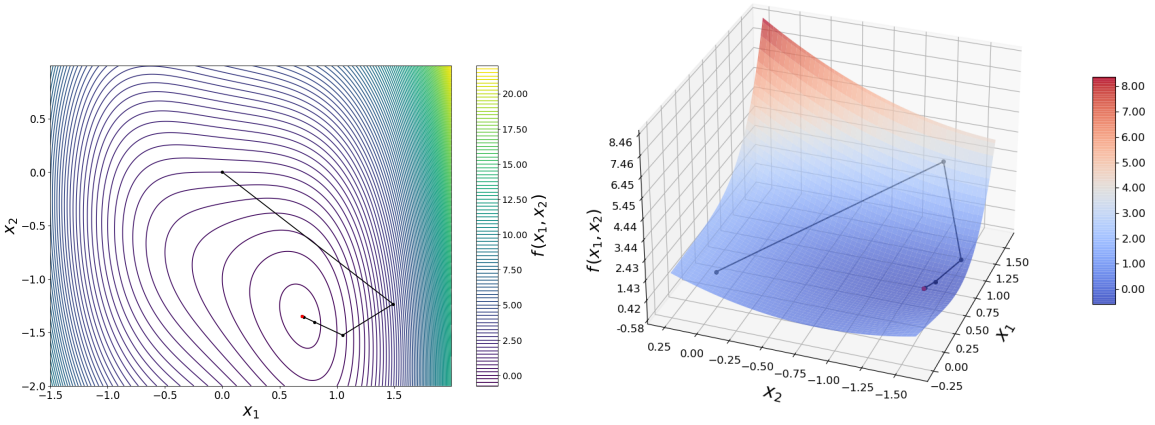
Finally, we implemented the trust region Newton algorithm to see if it is possible to reach convergence in fewer steps. In this case, we obtained the following results:

```
convergence = True, with 8 steps
min point = [ 0.69588439 -1.34794219]
min value = -0.5824451744436351
```

We can see that in this case the algorithm converges in 8 steps, returning the expected minimum point and the corresponding minimum value of the function. We report in Tab. 9 the required quantities at each step of the algorithm and in Fig. 7 the obtained contour plot and the 3D plot.

k	$\|\mathbf{x}_k - \mathbf{x}^*\|_2$	$f^{(d)}(\mathbf{x}_k) - f^{(d)}(\mathbf{x}^*)$	$-\nabla f^{(d)}(\mathbf{x}_k)^T[\nabla^2 f^{(d)}(\mathbf{x}_k)]^{-1}\nabla f^{(d)}(\mathbf{x}_k)$
0	1.517	1.582	0.000
1	8.014×10^{-1}	3.716	-5.554
2	3.959×10^{-1}	4.724×10^{-1}	-7.576×10^{-1}
3	1.232×10^{-1}	3.61×10^{-2}	-6.559×10^{-2}
4	1.717×10^{-2}	6.364×10^{-4}	-1.253×10^{-3}
5	4.011×10^{-4}	3.414×10^{-7}	-6.834×10^{-7}
6	2.275×10^{-7}	-4.435×10^{-10}	-2.171×10^{-13}
7	3.061×10^{-9}	-4.436×10^{-10}	-2.197×10^{-26}
8	3.061×10^{-9}	-4.436×10^{-10}	0.000

Table 9: Table of data obtained using the trust region Newton algorithm to minimize the function $f^{(d)}(x_1, x_2)$ by considering as starting point $x_0 = (0, 0)^T$.



(a) (b)

Figure 7: Contour plot and 3D plot of the function $f^{(d)}(x_1, x_2)$ where the intermediate points and the minimum point (in red) are obtained by using the Newton method with the trust region approach. The starting point considered in this case is $x_0 = (0, 0)^T$.