

# 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  $\alpha$  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

```
1 def Newton(func, grad, hess, tol, maxit, x_0, sol_x, sol_f, alpha=1,
2             sigma=0.0001, rho=0.5, backtracking=False):
3     r''' This function implements the standard Newton method for
4         unconstrained optimization.
5
6     Parameters
7     -----
8     func : function
9         Function to be minimized. It must be :math:`f : \mathbb{R}^n \rightarrow \mathbb{R}`
10    grad : function
11        Gradient of the function. It returns a 1d-array (vector)
12    hess : function
13        Hessian of the function. It returns a 2d-array (matrix)
14    tol : float
15        Tolerance parameter for the stopping criterion
16    maxit : int
17        Maximum number of iterations
18    x_0 : ndarray
19        Starting point
20    sol_x : ndarray
21        Exact solution (x) to the minimization problem
22    sol_f : float
23        Exact minimum value of the function
24    alpha : float
25        Step lenght. Default value alpha=1
26    sigma : float
```

```

25      Constant parameter in :math:`(0,1)`. Used if backtracking = True.
26          ↳ Default value sigma=0.0001
27  rho : float
28      Reduction parameter in :math:`(0,1)`. Used if backtracking = True.
29          ↳ Default value rho=0.5
30
31  Results
32  -----
33  results : dict
34      Dictionary of the results given by the function. It contains the
35          following items:
36          - 'convergence' : (bool) True if the algorithm converges, False
37              ↳ if it doesn't converge
38          - 'k' : (int) final iteration at which convergence is reached
39          - 'min_point' : (ndarray) computed point at which the minimum of
40              ↳ the function is reached
41          - 'min_value' : (float) computed minimum value of the function
42          - 'interm_point' : (list) list of the intermediate points
43          - 'error_x' : (list) list that contains the 2-norm of the
44              ↳ difference between each intermediate point and the exact
45              ↳ solution (min_point).
46          - 'error_f' : (list) list that contains the difference between
47              ↳ the function evaluated at each intermediate point and its
48              ↳ value in the exact minimum point
49          - 'scalar_product' : (list) list that contains the scalar product
50              ↳ between the descent direction and the gradient
51
52      ...
53
54  old_point = x_0
55  alpha_0 = alpha
56
57  # Create a list to save intermediate points
58  interm_points = [old_point]
59
60  # Create a list to save the scalar product between the gradient and the
61      ↳ descent direction
62  scalar_prod = []
63
64
65  new_point = x_0
66  norm_diff_x = np_lin.norm(new_point - old_point)
67  # Cycle on the number of iterations
68  for k in range(maxit):
69      gradient = grad(old_point)
70      hessian = hess(old_point)
71
72      # Compute norms for the stopping criterion
73      norm_grad = np_lin.norm(gradient)
74
75      # Check if the stopping criterion is satisfied
76      if norm_grad <= tol and norm_diff_x <= tol*(1 +
77          ↳ np_lin.norm(old_point)):
78          min_value = func(new_point)

```

```

64         conv = True
65         break
66
67     # Compute the descent direction by solving a linear system
68     p = np_lin.solve(hessian, -gradient)
69     scalar_prod.append(gradient @ p)
70
71     # Implement backtracking if backtracking == True
72     if backtracking == True:
73         alpha = alpha_0
74         iteraz_backtracking = 0
75         while func(old_point + alpha*p) > func(old_point) +
76             → (sigma*alpha)*(p @ gradient) and iteraz_backtracking < 100:
77             alpha = rho*alpha
78             iteraz_backtracking += 1
79
80     # Compute the new point and add it to the list of intermediate
81     → points
82     new_point = old_point + alpha*p
83     interm_points.append(new_point)
84     norm_diff_x = np_lin.norm(new_point - old_point)
85
86     old_point = new_point
87
88     if k == maxit-1:
89         min_value = func(new_point)
90         conv = False
91
92     gradient = grad(new_point)
93     hessian = hess(new_point)
94     p = np_lin.solve(hessian, -gradient)
95     scalar_prod.append(gradient @ p)
96
97     # Compute the 2norm of the difference between each intermediate point
98     → and the exact solution
99     error_x = [np_lin.norm(interm_x - sol_x) for interm_x in interm_points]
100
101    # Compute the difference between the function evaluated in each
102    → intermediate point and its value in the exact minimum point
103    error_f = [func(interm_x) - sol_f for interm_x in interm_points]
104
105    results = {'convergence': conv, 'k' : k, 'min_point' : new_point,
106    → 'min_value' : min_value , 'interm_point' : interm_points, 'error_x'
107    → : error_x, 'error_f' : error_f, 'scalar_product' : scalar_prod}
108
109    return results

```

Below the code of the trust region Newton algorithm.

```

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

```

```

2      ''' This function implements the Newton method with the trust region
3      approach for unconstrained optimization.
4
5      Parameters
6      -----
7      func : function
8          Function to be minimized. It must be :math:`f : \mathbb{R}^n \rightarrow \mathbb{R}`.
9      grad : function
10         Gradient of the function. It returns a 1d-array (vector)
11     hess : function
12         Hessian of the function. It returns a 2d-array (matrix)
13     tol : float
14         Tolerance parameter for the stopping criterion
15     maxit : int
16         Maximum number of iterations
17     x_0 : ndarray
18         Starting point
19     sol_x : ndarray
20         Exact solution ( $x$ ) to the minimization problem
21     sol_f : float
22         Exact minimum value of the function
23     alpha : float
24         Step lenght. Default value alpha=1
25     eta : float
26         Constant parameter in :math:`(0, 0.25)` . Default value eta=0.01
27
28      Results
29      -----
30      results : dict
31          Dictionary of the results given by the function. It contains the
32          following items:
33          - 'convergence' : (bool) True if the algorithm converges, False if
34              it doesn't converge
35          - 'k' : (int) final iteration at which convergence is reached
36          - 'min_point' : (ndarray) computed point at which the minimum of
37              the function is reached
38          - 'min_value' : (float) computed minimum value of the function
39          - 'interm_point' : (list) list of the intermediate points
40          - 'error_x' : (list) list that contains the 2-norm of the
41              difference between each intermediate point and the exact
42              solution (min_point).
43          - 'error_f' : (list) list that contains the difference between the
44              function evaluated at each intermediate point and its value in
45              the exact minimum point
46          - 'scalar_product' : (list) list that contains the scalar product
47              between the descent direction and the gradient
48
49      '''
50
51      # Evaluate the gradient and the hessian in the starting point
52      gradient = grad(x_0)

```

```

43     hessian = hess(x_0)
44
45     # Compute the fist descent direction and the first delta value
46     p = np_lin.solve(hessian, -gradient)
47     delta = np_lin.norm(p)
48     interm_radius = [delta]
49
50     interm_points = [x_0]
51     scalar_prod = []
52     old_point = x_0
53
54     # Cycle on the number of iterations
55     for k in range(maxit):
56
57         # Evaluate the gradient and the hessian at the current point
58         gradient = grad(old_point)
59         hessian = hess(old_point)
60
61         # Diagonalize the hessian computed at the current point
62         eigval, eigvect = np_lin.eigh(hessian)
63         # Choose an initial mu value
64         mu = abs(min(min(eigval), 0)) + 1e-12
65         coeff_vect = eigvect.T @ gradient/(eigval + mu)
66
67         # Choose the optimal mu value which respects the condition on the
68         # 2-norm
69         while sum([coeff**2 for coeff in coeff_vect]) > delta**2:
70             mu = mu*2
71             coeff_vect = eigvect.T @ gradient/(eigval + mu)
72
73         # Compute the descent direction
74         p = - eigvect @ coeff_vect
75         scalar_prod.append(- gradient @ eigvect @ np.diag(1/eigval) @
76                             eigvect.T @ gradient)
77
78         new_point = old_point + alpha*p
79
80         # Choose the value of delta for the successive iteration
81         rho = (func(new_point)-func(old_point))/(p @ gradient + 0.5*p @
82                                         hessian @ p)
83
84         if 0 < rho < 0.25:
85             delta = delta/4
86         elif rho > 0.75:
87             delta = 2*delta
88         elif 0 < rho < eta:
89             new_point = old_point
90
91         interm_points.append(new_point)
92
93         # Compute norms for the stopping criterion

```

```

91     norm_grad = np_lin.norm(gradient)
92     norm_diff_x = np_lin.norm(new_point - old_point)
93
94     # Check if the stopping criterion is satisfied
95     if norm_grad <= tol and norm_diff_x <= tol*(1 +
96         → np_lin.norm(new_point)):
97         min_value = func(new_point)
98         conv = True
99         break
100
101     old_point = new_point
102
103     if k == maxit-1:
104         min_value = func(new_point)
105         conv = False
106
107     gradient = grad(new_point)
108     hessian = hess(new_point)
109     p = np_lin.solve(hessian, -gradient)
110     scalar_prod.append(- gradient @ eigvect @ np.diag(1/eigval) @ eigvect.T
111         → @ gradient)
112
113     # Compute the 2norm of the difference between each intermediate point
114     → and the exact solution
115     error_x = [np_lin.norm(interm_x - sol_x) for interm_x in interm_points]
116
117     # Compute the difference between the function evaluated in each
118     → intermediate point and
119     # its value in the exact minimum point
120     error_f = [func(interm_x) - sol_f for interm_x in interm_points]
121
122     results = {'convergence': conv, 'k' : k, 'min_point' : new_point,
123         → 'min_value' : min_value, 'interm_point' : interm_points, 'error_x' :
124             → error_x, 'error_f' : error_f, 'scalar_product' : scalar_prod}
125
126     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  $\alpha$  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 of 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.

**(a)**

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

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

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  $f^{(a)}$ , which must be passed as arguments to the Python function which performs the minimization.

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

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

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 value of the function. In the following we report the table with the required quantities computed at each step.

k	$\ \mathbf{x}_k - \mathbf{x}^*\ _2$	$f_1(\mathbf{x}_k) - f_1(\mathbf{x}^*)$	$-\nabla f_1(\mathbf{x}_k)^T [\nabla^2 f_1(\mathbf{x}_k)]^{-1} \nabla f_1(\mathbf{x}_k)$
0	2.236	6.000	-9.000
1	1.118	1.500	-1.761
2	$6.805 \times 10^{-1}$	$4.092 \times 10^{-1}$	$-5.552 \times 10^{-1}$
3	$2.592 \times 10^{-1}$	$6.489 \times 10^{-2}$	$-1.237 \times 10^{-1}$
4	$5.012 \times 10^{-2}$	$2.531 \times 10^{-3}$	$-5.026 \times 10^{-3}$
5	$1.277 \times 10^{-3}$	$1.632 \times 10^{-6}$	$-3.262 \times 10^{-6}$
6	$1.659 \times 10^{-6}$	$2.754 \times 10^{-12}$	$-5.508 \times 10^{-12}$
7	$1.404 \times 10^{-12}$	$1.971 \times 10^{-24}$	$-3.943 \times 10^{-24}$
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  $x_0 = (1, 1)^T$ .

Below are the contour plot and the 3D plot, where it is possible to see how the minimum point (in red) is reached.

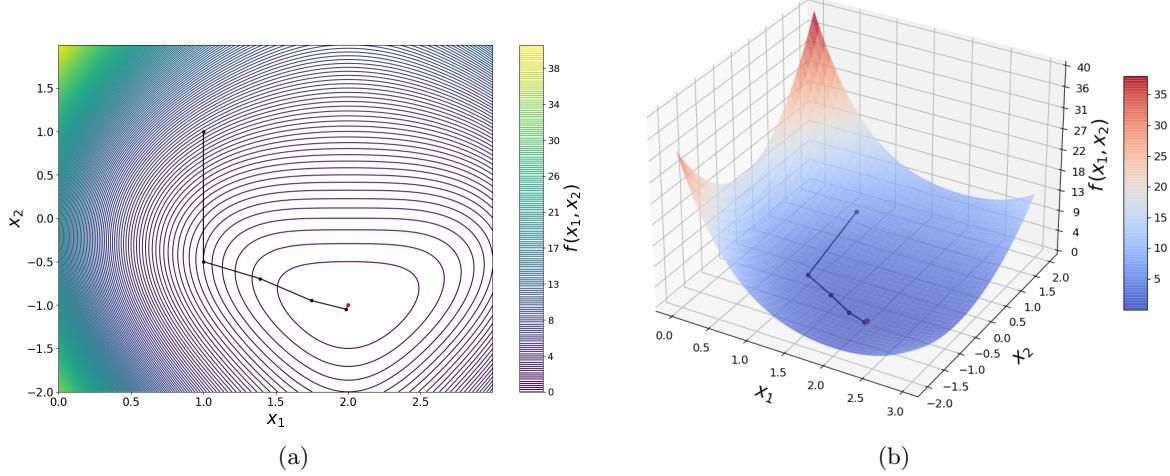


Figure 1: Contour plot (Panel (a)) and 3D plot (Panel (b)) of the function  $f^{(a)}(x_1, x_2)$  where the intermediate points and the minimum point (in red) are obtained by using the standard Newton algorithm starting from the point  $x_0 = (1, 1)^T$ .

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

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

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

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	$\ \mathbf{x}_k - \mathbf{x}^*\ _2$	$f_1(\mathbf{x}_k) - f_1(\mathbf{x}^*)$	$-\nabla f_1(\mathbf{x}_k)^T [\nabla^2 f_1(\mathbf{x}_k)]^{-1} \nabla f_1(\mathbf{x}_k)$
0	5.745	$5.223 \times 10^2$	$-1.045 \times 10^3$
1	$9.769 \times 10^{-13}$	$2.842 \times 10^{-14}$	$-9.948 \times 10^{-27}$
2	$1.688 \times 10^{-13}$	0.000	$-2.005 \times 10^{-26}$

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  $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ , 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	$\ \mathbf{x}_k - \mathbf{x}^*\ _2$	$f_1(\mathbf{x}_k) - f_1(\mathbf{x}^*)$	$-\nabla f_1(\mathbf{x}_k)^T [\nabla^2 f_1(\mathbf{x}_k)]^{-1} \nabla f_1(\mathbf{x}_k)$
0	5.009	$8.17 \times 10^1$	$-1.455 \times 10^2$
1	$8.66 \times 10^{-1}$	2.423	-4.527
2	$6.494 \times 10^{-2}$	$2.407 \times 10^{-2}$	$-4.643 \times 10^{-2}$
3	$1.393 \times 10^{-1}$	$3.45 \times 10^{-3}$	$-6.32 \times 10^{-3}$
4	$2.103 \times 10^{-2}$	$1.383 \times 10^{-4}$	$-2.704 \times 10^{-4}$
5	$1.377 \times 10^{-3}$	$2.863 \times 10^{-7}$	$-5.717 \times 10^{-7}$
6	$3.033 \times 10^{-6}$	$2.186 \times 10^{-12}$	$-4.372 \times 10^{-12}$
7	$2.836 \times 10^{-11}$	$1.233 \times 10^{-22}$	$-2.466 \times 10^{-22}$
8	$4.441 \times 10^{-16}$	$4.437 \times 10^{-31}$	$-8.79 \times 10^{-31}$
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  $x_0 = (8, 0.2)^T$ .

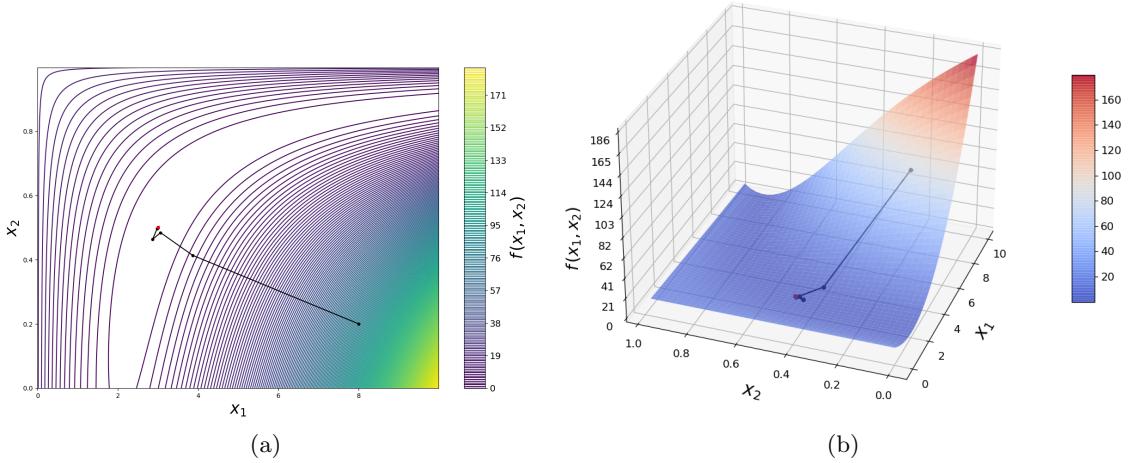


Figure 2: Contour plot (Panel (a)) and 3D plot (Panel (b)) of the function  $f^{(c)}(x_1, x_2)$  where the intermediate points and the minimum point (in red) are obtained by using the standard Newton algorithm starting from the point  $x_0 = (8, 0.2)^T$ .

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	$\ \mathbf{x}_k - \mathbf{x}^*\ _2$	$f_1(\mathbf{x}_k) - f_1(\mathbf{x}^*)$	$-\nabla f_1(\mathbf{x}_k)^T [\nabla^2 f_1(\mathbf{x}_k)]^{-1} \nabla f_1(\mathbf{x}_k)$
0	5.009	2.043	-3.291
1	3.963	$2.553 \times 10^{-1}$	$-5.885 \times 10^{-2}$
2	4.218	$2.328 \times 10^{-1}$	$-6.421 \times 10^{-1}$
3	$1.661 \times 10^1$	$5.743 \times 10^2$	$-9.291 \times 10^2$
4	6.137	$5.226 \times 10^1$	$-6.327 \times 10^1$
5	3.426	$1.596 \times 10^1$	-3.709
6	2.974	$1.421 \times 10^1$	$-8.445 \times 10^{-3}$
7	3.041	$1.42 \times 10^1$	$-5.688 \times 10^{-8}$
8	3.041	$1.42 \times 10^1$	$-1.083 \times 10^{-18}$
9	3.041	$1.42 \times 10^1$	0.000
10	3.041	$1.42 \times 10^1$	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  $x_0 = (8, 0.8)^T$ .

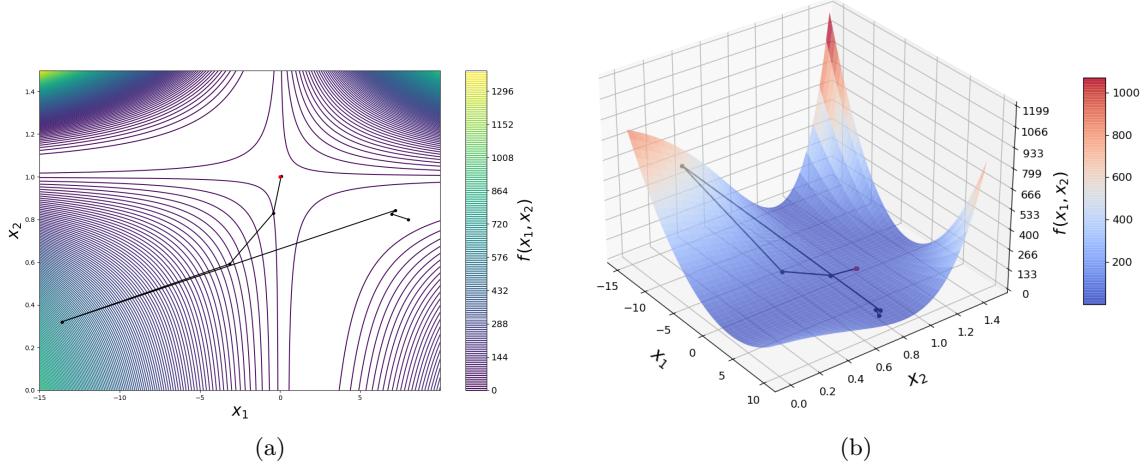


Figure 3: Contour plot (Panel (a)) and 3D plot (Panel (b)) of the function  $f^{(c)}(x_1, x_2)$  where the intermediate points and the minimum point (in red) are obtained by using the standard Newton algorithm starting from the point  $x_0 = (8, 0.2)^T$ .

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	$\ \mathbf{x}_k - \mathbf{x}^*\ _2$	$f_1(\mathbf{x}_k) - f_1(\mathbf{x}^*)$	$-\nabla f_1(\mathbf{x}_k)^T [\nabla^2 f_1(\mathbf{x}_k)]^{-1} \nabla f_1(\mathbf{x}_k)$
0	5.009	2.043	-3.291
1	3.963	$2.553 \times 10^{-1}$	$-5.885 \times 10^{-2}$
2	4.218	$2.328 \times 10^{-1}$	$-6.421 \times 10^{-1}$
3	2.920	$2.131 \times 10^{-1}$	$-7.337 \times 10^{-2}$
4	2.471	$1.649 \times 10^{-1}$	$-1.224 \times 10^{-1}$
5	1.340	$1.502 \times 10^{-1}$	$-1.22 \times 10^{-1}$
6	1.192	$7.989 \times 10^{-2}$	$-1.697 \times 10^{-1}$
7	$6.453 \times 10^{-1}$	$5.012 \times 10^{-2}$	$-5.027 \times 10^{-2}$
8	$3.335 \times 10^{-1}$	$1.73 \times 10^{-2}$	$-2.418 \times 10^{-2}$
9	$8.357 \times 10^{-2}$	$3.009 \times 10^{-3}$	$-5.269 \times 10^{-3}$
10	$2.128 \times 10^{-2}$	$1.004 \times 10^{-4}$	$-1.967 \times 10^{-4}$
11	$2.767 \times 10^{-4}$	$1.503 \times 10^{-7}$	$-3.005 \times 10^{-7}$
12	$1.103 \times 10^{-6}$	$2.293 \times 10^{-13}$	$-4.585 \times 10^{-13}$
13	$2.404 \times 10^{-13}$	$8.166 \times 10^{-25}$	$-1.633 \times 10^{-24}$
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  $x_0 = (8, 0.8)^T$ .

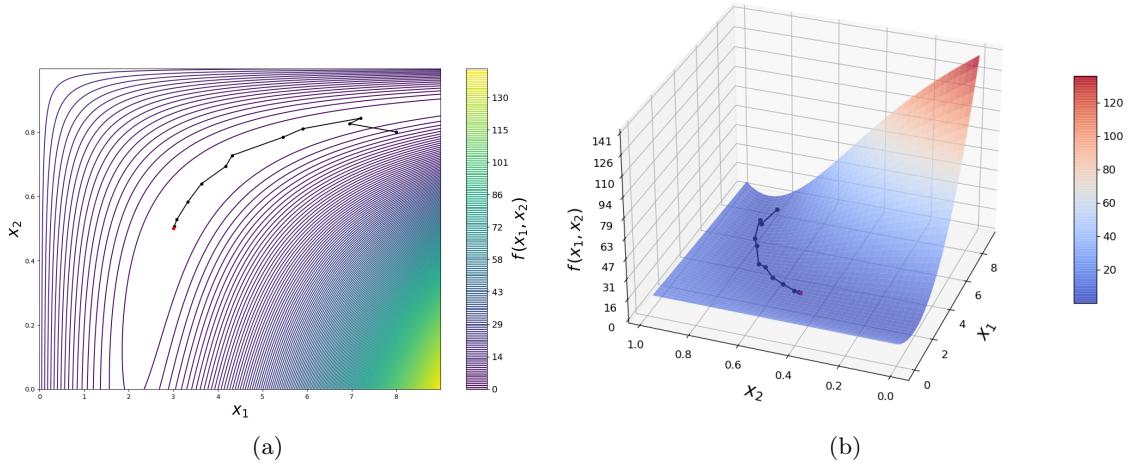


Figure 4: Contour plot (Panel (a)) and 3D plot (Panel (b)) of the function  $f^{(c)}(x_1, x_2)$  where the intermediate points and the minimum point (in red) are obtained by using the Newton algorithm with backtracking, starting from the point  $x_0 = (8, 0.2)^T$ .

### 2.3 (d)

Consider the function  $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ , 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.

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

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	$\ \mathbf{x}_k - \mathbf{x}^*\ _2$	$f_1(\mathbf{x}_k) - f_1(\mathbf{x}^*)$	$-\nabla f_1(\mathbf{x}_k)^T [\nabla^2 f_1(\mathbf{x}_k)]^{-1} \nabla f_1(\mathbf{x}_k)$
0	$1.119 \times 10^{-1}$	$2.385 \times 10^{-2}$	$-4.688 \times 10^{-2}$
1	$4.601 \times 10^{-3}$	$4.517 \times 10^{-5}$	$-8.996 \times 10^{-5}$
2	$2.951 \times 10^{-5}$	$1.406 \times 10^{-9}$	$-3.7 \times 10^{-9}$
3	$3.805 \times 10^{-9}$	$-4.436 \times 10^{-10}$	$-6.372 \times 10^{-18}$
4	$3.061 \times 10^{-9}$	$-4.436 \times 10^{-10}$	0.000
5	$3.061 \times 10^{-9}$	$-4.436 \times 10^{-10}$	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  $x_0 = (0.75, -1.25)^T$ .

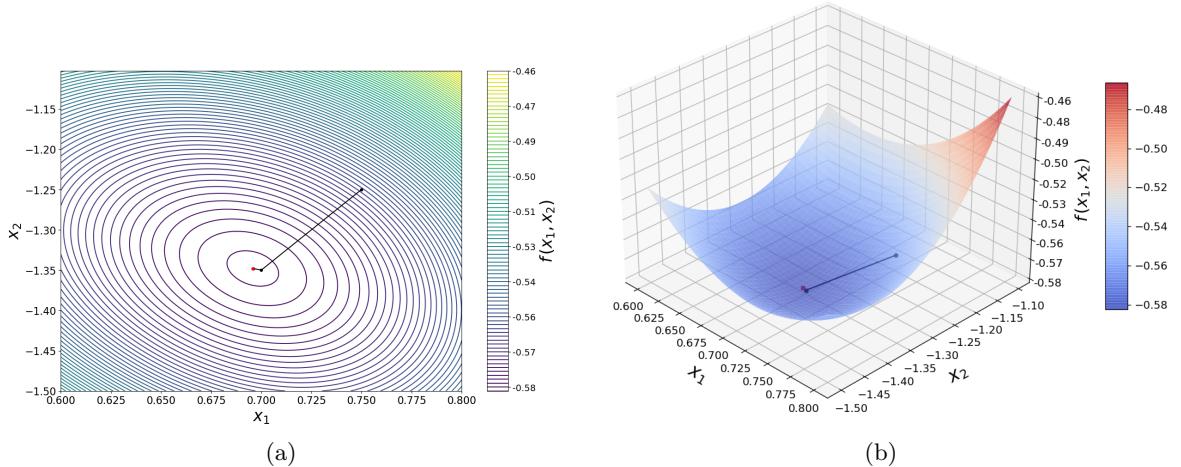


Figure 5: Contour plot (Panel (a)) and 3D plot (Panel (b)) of the function  $f^{(c)}(x_1, x_2)$  where the intermediate points and the minimum point (in red) are obtained by using the standard Newton algorithm with  $\alpha = 1$ , starting from the point  $x_0 = (8, 0.2)^T$ .

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.?? the first and the last values of the required quantities.

k	$\ \mathbf{x}_k - \mathbf{x}^*\ _2$	$f_1(\mathbf{x}_k) - f_1(\mathbf{x}^*)$	$-\nabla f_1(\mathbf{x}_k)^T [\nabla^2 f_1(\mathbf{x}_k)]^{-1} \nabla f_1(\mathbf{x}_k)$
0	1.517	1.582	0.000
1	3.014	$1.758 \times 10^1$	$-2.156 \times 10^1$
2	2.261	4.563	-4.537
3	1.736	1.796	-1.148
4	1.321	1.065	$-6.339 \times 10^{-1}$
5	$7.376 \times 10^{-1}$	$5.459 \times 10^{-1}$	2.140
6	3.088	$1.979 \times 10^1$	$-2.447 \times 10^1$
7	2.311	5.017	-5.116
8	1.772	1.900	-1.262
9	1.353	1.101	$-6.193 \times 10^{-1}$
10	$8.159 \times 10^{-1}$	$6.16 \times 10^{-1}$	1.988
11	3.077	$1.945 \times 10^1$	$-2.402 \times 10^1$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
90	$7.939 \times 10^{-1}$	$5.966 \times 10^{-1}$	1.981
91	3.025	$1.789 \times 10^1$	$-2.197 \times 10^1$
92	2.268	4.627	-4.618
93	1.742	1.810	-1.164
94	1.326	1.070	$-6.309 \times 10^{-1}$
95	$7.501 \times 10^{-1}$	$5.573 \times 10^{-1}$	2.081
96	3.048	$1.859 \times 10^1$	$-2.288 \times 10^1$
97	2.284	4.770	-4.800
98	1.753	1.843	-1.200
99	1.336	1.082	$-6.254 \times 10^{-1}$
100	$7.761 \times 10^{-1}$	$5.807 \times 10^{-1}$	2.004

Table 7: text

k	$\ \mathbf{x}_k - \mathbf{x}^*\ _2$	$f_1(\mathbf{x}_k) - f_1(\mathbf{x}^*)$	$-\nabla f_1(\mathbf{x}_k)^T [\nabla^2 f_1(\mathbf{x}_k)]^{-1} \nabla f_1(\mathbf{x}_k)$
0	1.517	1.582	0.000
1	2.837	$1.208 \times 10^1$	$-1.432 \times 10^1$
2	2.181	3.888	-3.680
3	1.725	1.764	-1.114
4	1.352	1.099	$-6.198 \times 10^{-1}$
5	$8.656 \times 10^{-1}$	$6.593 \times 10^{-1}$	2.174
6	3.138	$2.143 \times 10^1$	$-2.663 \times 10^1$
7	2.424	6.213	-6.657
8	1.910	2.391	-1.830
9	1.514	1.320	$-6.987 \times 10^{-1}$
10	1.126	$8.791 \times 10^{-1}$	-1.402
11	$3.352 \times 10^{-1}$	$3.219 \times 10^{-1}$	$-5.271 \times 10^{-1}$
12	$1.188 \times 10^{-1}$	$3.348 \times 10^{-2}$	$-6.1 \times 10^{-2}$
13	$2.637 \times 10^{-2}$	$1.514 \times 10^{-3}$	$-2.957 \times 10^{-3}$
14	$3.474 \times 10^{-3}$	$2.572 \times 10^{-5}$	$-5.128 \times 10^{-5}$
15	$3.626 \times 10^{-4}$	$2.789 \times 10^{-7}$	$-5.586 \times 10^{-7}$
16	$3.643 \times 10^{-5}$	$2.375 \times 10^{-9}$	$-5.637 \times 10^{-9}$
17	$3.646 \times 10^{-6}$	$-4.154 \times 10^{-10}$	$-5.642 \times 10^{-11}$
18	$3.659 \times 10^{-7}$	$-4.434 \times 10^{-10}$	$-5.643 \times 10^{-13}$

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  $x_0 = (0, 0)^T$ .

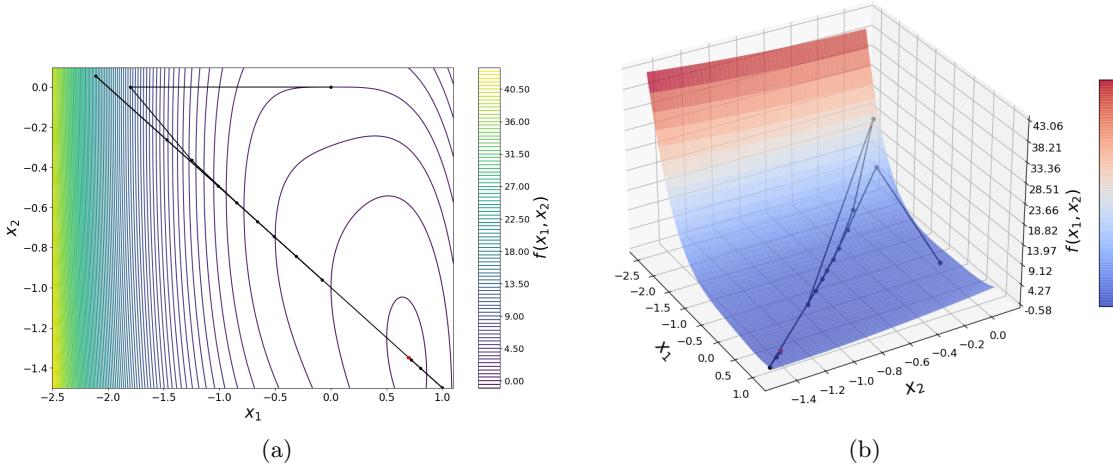


Figure 6: Contour plot and 3D plot for the function  $f^{(d)}(x_1, x_2)$  where the intermediate points and the minimum point (in red) are obtained by using the standard Newton method with  $\alpha = 0.9$ . The starting point considered in this case is  $x_0 = (0, 0)^T$ .

k	$\ \mathbf{x}_k - \mathbf{x}^*\ _2$	$f_1(\mathbf{x}_k) - f_1(\mathbf{x}^*)$	$-\nabla f_1(\mathbf{x}_k)^T [\nabla^2 f_1(\mathbf{x}_k)]^{-1} \nabla f_1(\mathbf{x}_k)$
0	1.517	1.582	0.000
1	$8.014 \times 10^{-1}$	3.716	-5.554
2	$3.959 \times 10^{-1}$	$4.724 \times 10^{-1}$	$-7.576 \times 10^{-1}$
3	$1.232 \times 10^{-1}$	$3.61 \times 10^{-2}$	$-6.559 \times 10^{-2}$
4	$1.717 \times 10^{-2}$	$6.364 \times 10^{-4}$	$-1.253 \times 10^{-3}$
5	$4.011 \times 10^{-4}$	$3.414 \times 10^{-7}$	$-6.834 \times 10^{-7}$
6	$2.275 \times 10^{-7}$	$-4.435 \times 10^{-10}$	$-2.171 \times 10^{-13}$
7	$3.061 \times 10^{-9}$	$-4.436 \times 10^{-10}$	$-2.197 \times 10^{-26}$

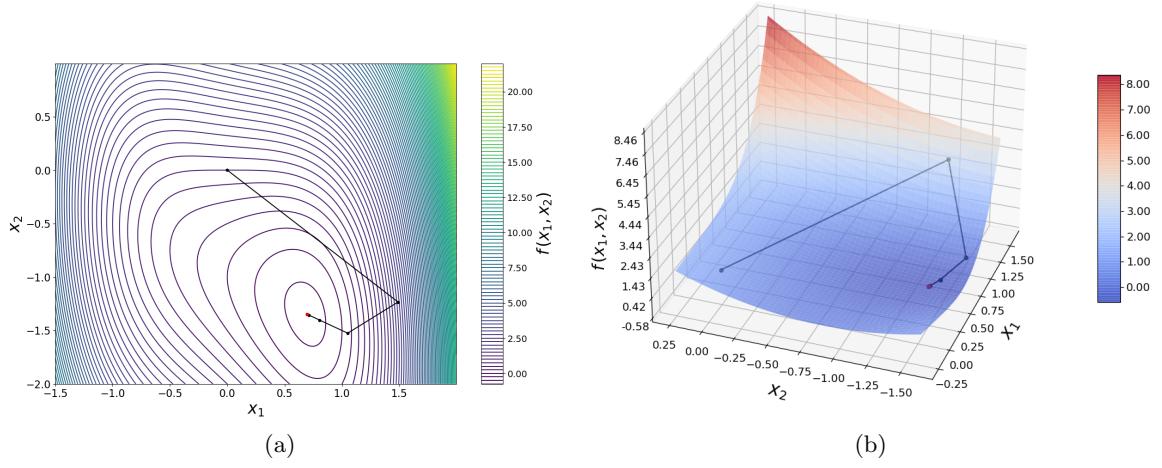


Figure 7: Contour plot and 3D plot of the function  $f^{(d)}(x_1, x_2)$  where the intermediate points and the minimum point (in red) are obtained by using the Newton method with the trust region approach. The starting point considered in this case is  $x_0 = (0, 0)^T$ .