Comparative Analysis of Modified Newton and Modified Newton with Finite Differences Methods

Annalisa Belloni s[redacted] Valeria Petrianni s[redacted]

February 18, 2024

1 Introduction

This report aims to explore unconstrained optimization methods by implementing and testing two different algorithms: the Modified Newton method and the Modified Newton method with Finite Differences. Our goal is to compare them and to understand how they perform in different scenarios, by testing them on various problems.

2 Applied methods

2.1 Modified Newton method

The Modified Newton method, like the classical Newton method, allows minimizing a given function f through an iterative process based on approximation. At each k-th iteration, the function f (evaluated at point x_k) is approximated by a quadratic function $m_{f,x}(p) := f(x) + p^T \nabla f(x) + \frac{1}{2} p^T (\nabla^2 f(x)) p$. Specifically, the direction p_k of movement from x_k to x_{k+1} is obtained by seeking the stationary point of the quadratic approximation, setting its gradient to 0. The linear system that provides the vector p_k is given by:

$$\nabla m_{f,\mathbf{x}}(\mathbf{p}) = 0 \Leftrightarrow \nabla^2 f(x_k) p_k = -\nabla f(x_k). \tag{1}$$

Clearly, the direction p_k just found will be a *descent* direction for f if the function at x_k has positive concavity, i.e., in the presence of a positive definite Hessian matrix of f at x_k. It is evident, therefore, how the Newton Method ensures convergence to the minimum of a function only when it is convex at the points of the sequence $\{x_k\}_{k \in \mathbb{N}}$.
The Modified Newton method, unlike its classical version, allows optimizing functions with regions of negative concavity by adjusting the Hessian matrix. This adjustment ensures that the resulting matrix is positive definite, thus guaranteeing that the direction p_k leads to a descent in

the function f.

The pseudocode for the method is given by:

Algorithm 1 Modified Newton method

Require: initial guess x_0
  $k \leftarrow 0$
  while $k < k_{\max}$ and $\|\nabla f(x_k)\| \geq tol$ **do**
    Compute the matrix $B_k = \nabla^2 f(x_k) + E_k$:
    Beta = norm($Hessf(x_k)$); ▷ with *Frobenius* norm
    if Beta = 0 **then**
      Beta = $\sqrt{\epsilon_m}$; ▷ to avoid infinite loop
    end if
    if min(diag($Hessf(x_k)$)) ≤ 0 **then** ▷ we can at least suppose it is positive definite
      $\tau_k = 0$;
    else
      τ_k = Beta $* 0.5$;
    end if
    while B_k is not pos. def. **do**
      $B_k = Hessf(x_k) + \tau_k$Id;
      if B_k is sym. and min(eig(B_k)) $\geq \sqrt{\epsilon_m}$ **then**
        B_k is marked as *suff. pos. def.*;
      else
        $\tau_k = \max(2 * \tau_k, \text{Beta} * 0.5)$;
      end if
    end while
    Solve $B_k p_k = -\nabla f(x_k)$ with a direct method (e.g., *Cholesky* Factorization of B_k);
    $x_{k+1} \leftarrow x_k + \alpha_k p_k$; where α_k satisfies the Armijo backtracking condition [1]
    $k \leftarrow k + 1$;
  end while

In Algorithm 1, with the expression *sufficiently positive definite*, we mean that the smallest eigenvalue of B_k, that we define as $\lambda_{min}(B_k)$, must satisfy:

$$\lambda_{min}(B_k) \geq \delta > 0 \tag{2}$$

where the value of δ is defined by the user and usually set to $\sqrt{\epsilon_m}$.

2.2 Modified Newton method with Finite Differences

The second algorithm we analysed is the Modified Newton method in which the gradient and the Hessian of the function f are approximated leveraging the Finite Differences technique.
In particular, concerning the gradient we can state:

Theorem 2.1 (Approximation of the gradient (centered finite differences))**.** Given a function

[1] $f(x^{(k+1)}) \leq f(x^{(k)}) + c_1 \alpha^{(k)} (\nabla f(x^{(k)}))^T p^{(k)}$

$f : \mathbb{R}^n \to \mathbb{R}$, the gradient $\nabla f(x)$ of the function at a given point x_0 can be approximated by:

$$\frac{\partial f}{\partial x_i}(x_0) \approx \frac{f(x_0 + he_i) - f(x_0 - he_i)}{h} \qquad \forall i = 1, \ldots, n. \tag{3}$$

where e_i is the i-th vector of the canonical basis of $\mathbb{R}^n$.

While, regarding the approximation of the Hessian matrix, we can assert:

Theorem 2.2 (Approximation of the Hessian)**.** Given a function $f : \Omega \subseteq \mathbb{R}^n \to \mathbb{R}$, if f is $C^2(\Omega)$ the Hessian $H_f(x_0)$ of f in $x_0 \in \Omega$ can be approximated by:

- Hessian's **diagonal** elements:

$$(H_f)_{ii}(x_0) \approx \frac{f(x_0 + he_i) - 2f(x_0) + f(x_0 - he_i)}{h^2} \tag{4}$$

- Hessian's **non diagonal** elements:

$$(H_f)_{ij}(x_0) \approx \frac{f(x_0 + he_i + he_j) - f(x_0 + he_i) - f(x_0 + he_j) + f(x_0)}{h^2} \tag{5}$$

Typically, the value of h in the case of the centered approximation of the gradient is set equal to $h_{grad} = \sqrt{\epsilon_m}$, while in the case of the approximation of the second order derivatives a typical choice of h is $h_{Hess} = \sqrt{h_{grad}}$.
Approximating gradients and Hessians can be beneficial when computing partial derivatives is computationally intensive and can help saving some memory space. However, it generally leads to less accurate optimization and slower convergence. By testing the algorithm on subsequent problems and comparing the results obtained with those provided by the Modified Newton method, we will have proof of this.

3 Problem overview

In the following section, we present the problems that were used to evaluate our implementations of the methods described in Section 2. Specifically, we will seek a minimum of the following objective functions, from different starting points $\bar{x}$.

3.1 *Rosenbrock* function

The *Rosenbrock* function is a non-convex function widely used to test optimization algorithms. In this case, we will use the the 2-dimensional *Rosenbrock* function, which is defined as:

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \tag{6}$$

The gradient and the Hessian of f are given by:

$$\nabla f(x_1, x_2) = \begin{bmatrix} 400x_1^3 - 400x_1x_2 + 2x_1 - 2 \\ 200x_2 - 200x_1^2 \end{bmatrix} \tag{7}$$

$$H_f(x_1, x_2) = \begin{bmatrix} 1200x_1^2 - 400x_2 + 2 & -400x_1 \\ -400x_1 & 200 \end{bmatrix} \tag{8}$$

3.2 Problem 16: *Banded trigonometric problem*

$$F(x) = \sum_{i=1}^{n} i \left[\cos(x_i)) + \sin(x_{i-1}) - \sin(x_{i+1})\right] \tag{9}$$

Where $x_0 = x_{n+1} = 0$ and $\bar{x}_i = 1$ for $i \geq 1$.

3.3 Problem 27

$$F(x) = \frac{1}{2} \sum_{k=1}^{m} f_k^2(x) \tag{10}$$

$$f_k(x) = \frac{1}{\sqrt{100000}}(x_k - 1), \quad 1 \leq k \leq n \tag{11}$$

$$f_k(x) = \sum_{i=1}^{n} x_i^2 - \frac{1}{4}, \quad k = n + 1 \tag{12}$$

where $m = n + 1$ and $\bar{x}_l = l \quad l \geq 1$.

3.4 Problem 87: *Ascher and Russel boundary value problem*

$$F(x) = \frac{1}{2} \sum_{k=1}^{n} f_k(x) \tag{13}$$

$$f_k(x) = 2x_k - 2h^2 \left(\frac{x_k^2 + x_{k+1} - x_{k-1}}{2h} \right) - x_{k-1} - x_{k+1}, \quad 1 \leq k \leq n \tag{14}$$

where $h = \frac{1}{n+1}$, $x_0 = 0$, $x_{n+1} = \frac{1}{2}$, and $x_l = 1 \quad l \geq 1$.
We immediately observe that the objective function $F(x)$ to be minimized in this case is a quadratic function with negative concavity: it is strictly concave, thus it does not admit a minimum. In the subsequent sections, we will delve deeper into this issue, which represents a significant case.

4 Test and Comparison

We start by saying that in all the following cases, we have considered the recommended values for δ (2), h_{grad}, and h_{Hess} (2.2), which are respectively: $\sqrt{\epsilon_m}$, $\sqrt{\epsilon_m}$ and $\sqrt{\sqrt{\epsilon_m}}$.
Furthermore, the execution times reported in this section were derived by calculating the mean of five time values obtained from five repeated executions of the same code for each considered problem and parameter configuration.
Moreover, the convergence rates p, which will be reported in the subsequent tables, represent the slope of the regression line ($y = px$) aimed at fitting the following sets of points:
$y = log(\| e_{k+1} \|)$, where $e_{k+1} = x_{k+1} - x_k$ is an approximation of $x^* - x_{k+1}$ (i.e. the error at iteration $k+1$), and
$x = log(\| e_k \|)$, where $e_k = x_k - x_{k-1}$ is an approximation of $x^* - x_k$ (i.e. the error at iteration k).
Finally, concerning the successes and failures of the algorithms tested on various problems, for simplicity, we've marked in yellow the rows of the summary tables where the Modified Newton method terminated due to reaching the maximum iteration count $kmax$, rather than meeting the gradient norm stopping criterion. Conversely, in green, we've highlighted the rows where the same situation occurred with the Modified Newton method using Finite Differences.

4.1 *Rosenbrock* function

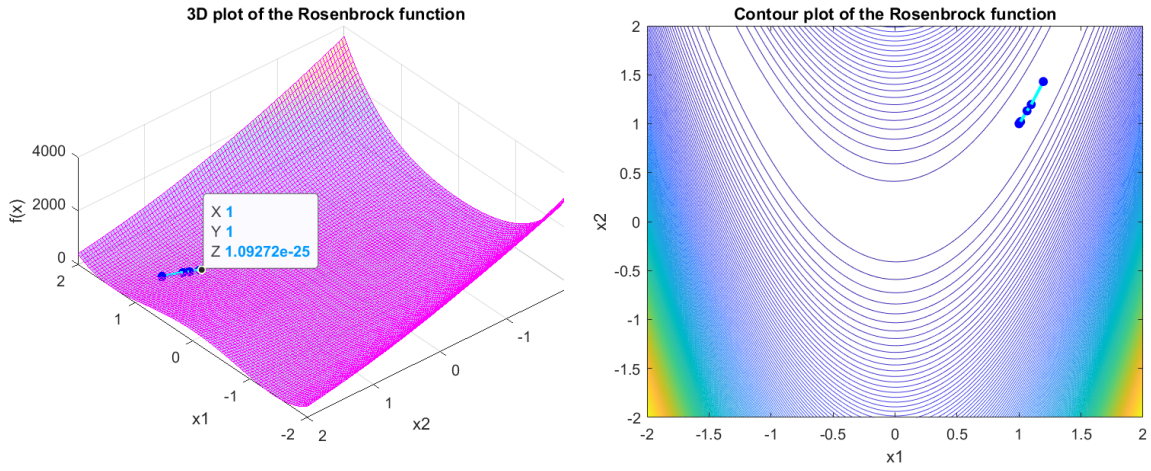
We employed $kmax = 1000$ as the upper limit for outer iterations, and $btmax = 1000$ as the maximum number of inner iterations allowed during the backtracking phase on α_k. As starting points of the sequence $\{x_k\}_{k\in\mathbb{N}}$ we have considered: $\bar{x}_a = (1.2, 1.2)$ and $\bar{x}_b = (-1.2, 1)$ $(*)$. We provide a summary table (1) below, containing the values of the minimum point x_k, the minimum f_k, the gradient of f at x_k, the number of iterations performed k, the execution time in seconds and the rate of convergence p, observed after testing the two methods initialized with different parameter configurations and starting points.
We briefly recall the meaning of the following parameters: $\rho \in (0, 1)$ represents the reduction factor for α_k (used during the backtracking phase); c_1 represents instead the reduction factor of the tangent hyperplane of f at x_k (used to check the Armijo condition: $f(x^{(k+1)}) \leq f(x^{(k)}) + c_1\alpha^{(k)}(\nabla f(x^{(k)}))^T p^{(k)}$); while tol is the tolerance value set on the norm of $\nabla f(x_k)$.

We observe that, for both methods, decreasing the tolerance tol from 10^{-12} to 10^{-8} results in a decrease in the number of iterations k and in the quality of the solution. While, reducing ρ from 10^{-4} to 10^{-7} relaxes the Armijo condition, leading to a reduction in execution times.
In addition, as expected, the Modified Newton Method with finite differences is generally less accurate and slower to converge compared to the Newton Method, due to the repeated approximations that accumulate errors.
Let us now present the graphs representing the sequences $\{x_k\}_{k\in\mathbb{N}}$ initialized with $x_0 = \bar{x}_a$

Method	parameters	x_k	f_k	$\| \nabla f(x_k) \|$	k	time	p
MN	$x_0 = \bar{x}_a$; $\rho = 0.5$; $c_1 = 10^{-4}$; $tol = 10^{-12}$;	(1,1)	2.5559e-28	8.8818e-14	9	0.0156	1.8284
	$x_0 = \bar{x}_b$; $\rho = 0.5$; $c_1 = 10^{-4}$; $tol = 10^{-12}$;	(1,1)	3.7286e-29	1.2212e-14	22	0.0033	1.6160
	$x_0 = \bar{x}_a$; $\rho = 0.5$; $c_1 = 10^{-7}$; $tol = 10^{-8}$;	(1,1)	1.0927e-25	1.4393e-11	8	0.0141	1.5719
	$x_0 = \bar{x}_b$; $\rho = 0.5$; $c_1 = 10^{-7}$; $tol = 10^{-8}$;	(1,1)	3.7419e-21	4.4742e-10	21	0.0032	1.4184
MN&FinDiff	$x_0 = \bar{x}_a$; $\rho = 0.5$; $c_1 = 10^{-4}$; $tol = 10^{-12}$	(1,1)	1.8990e-27	9.7716e-14	13	0.0205	1.1790
	$x_0 = \bar{x}_b$; $\rho = 0.5$; $c_1 = 10^{-4}$; $tol = 10^{-12}$	(1,1)	1.0589e-26	1.4673e-13	26	0.0031	1.1744
	$x_0 = \bar{x}_a$; $\rho = 0.5$; $c_1 = 10^{-7}$; $tol = 10^{-8}$;	(1,1)	6.7505e-19	9.1848e-10	10	0.0187	1.2680
	$x_0 = \bar{x}_b$; $\rho = 0.5$; $c_1 = 10^{-7}$; $tol = 10^{-8}$	(1,1)	6.8614e-19	9.2600e-10	23	0.0034	1.2466

Table 1: Results obtained with the two considered methods, on the *Rosenbrock* Function, varying the parameters *tol* and c_1 and initializing with two different starting points $\bar{x}_a$ and $\bar{x}_b$ $(*)$.

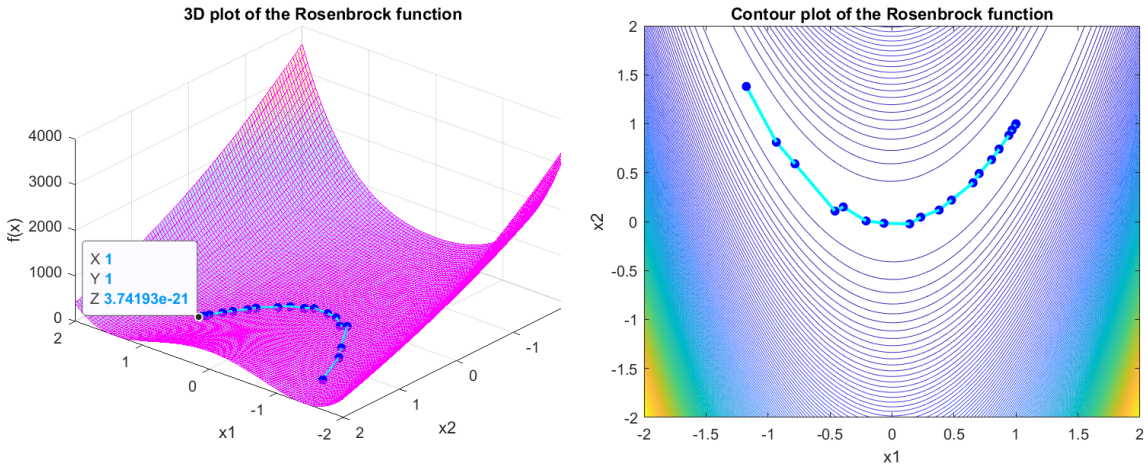
and with $x_0 = \bar{x}_a$, obtained with the following parameter configuration: $\rho = 0.5$; $c_1 = 10^{-4}$; $tol = 10^{-12}$.



(a) 3D plot

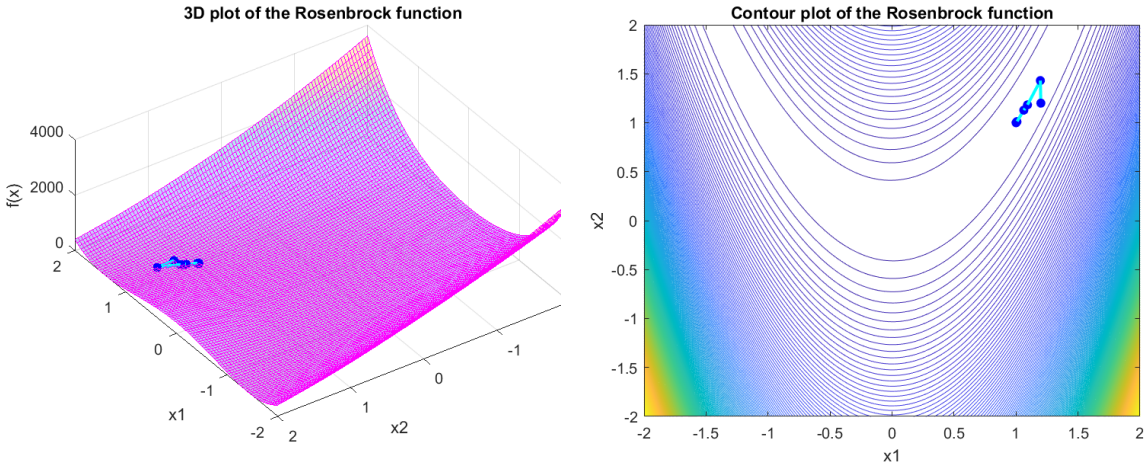
(b) Contour plot

Figure 1: Sequence $\{x_k\}_{k \in \mathbb{N}}$ visualized in 2D and in 3D, obtained with Modified Newton method, and beginning with $\bar{x}_a$.



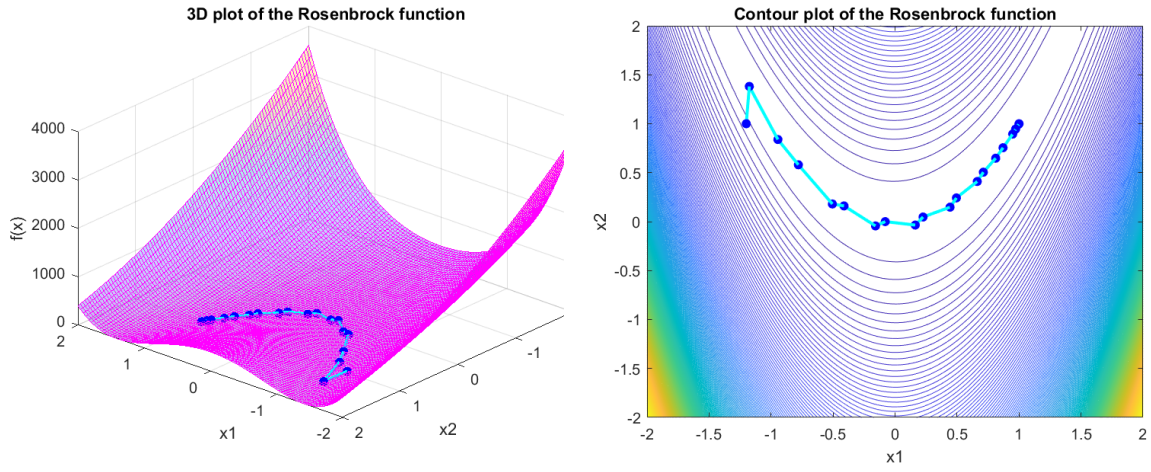
(a) 3D plot (b) Contour plot

Figure 2: Sequence $\{x_k\}_{k\in\mathbb{N}}$ visualized in 2D and in 3D, obtained with Modified Newton method, and beginning with $\bar{x}_b$.



(a) 3D plot (b) Contour plot

Figure 3: Sequence $\{x_k\}_{k\in\mathbb{N}}$ visualized in 2D and 3D, obtained with Modified Newton method with Finite Differences, beginning with $\bar{x}_b$.



(a) 3D plot

(b) Contour plot

Figure 4: Sequence $\{x_k\}_{k\in\mathbb{N}}$ visualized in 2D and 3D, obtained with Modified Newton method with Finite Differences, beginning with $\bar{x}_b$.

4.2 Problem 16: *Banded trigonometric problem*

Here we employed $kmax = 10000$ as the upper limit for outer iterations, and $btmax = 1000$ as the maximum number of iterations allowed during the backtracking phase. As an initial step, in order to visualize the function and understand its characteristic, we tested the algorithms for $n = 2$ and choosing as initial guess $x_0 = (8, 9)$. The results are shown in Figure 5 and 6.

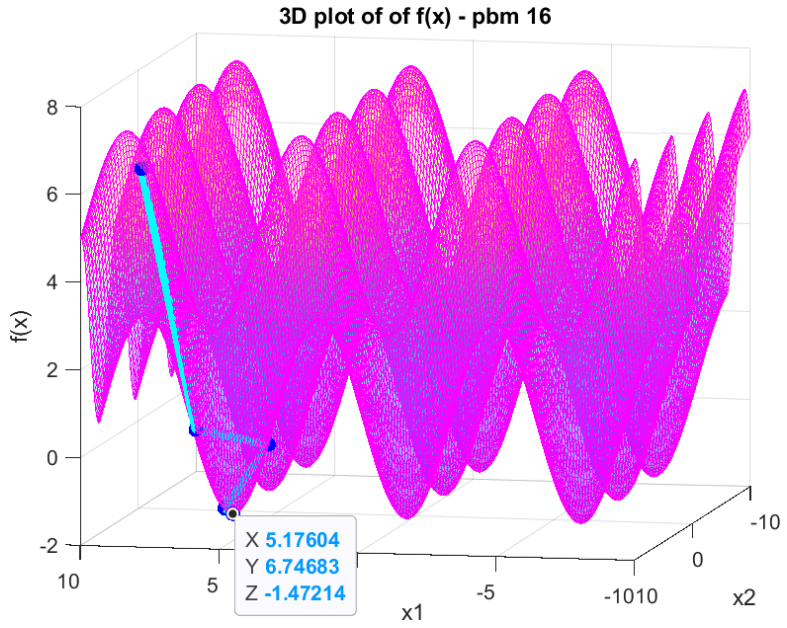


Figure 5: Sequence $\{x_k\}_{k\in\mathbb{N}}$ visualized in 3D, obtained with Modified Newton method, beginning with $\bar{x}_0 = (8, 9)$.

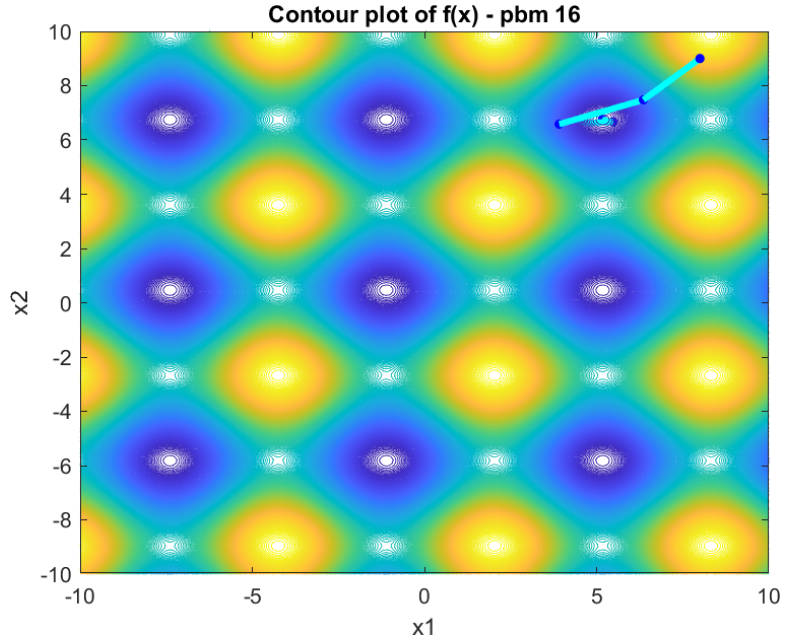


Figure 6: Sequence $\{x_k\}_{k\in\mathbb{N}}$ visualized in 2D, obtained with Modified Newton method, beginning with $\bar{x}_0 = (8, 9)$.

It can be easily observed that the function of the *Banded Trigonometric Problem*, that is

a periodic function, exhibits multiple local minima and the Modified Newton Method usually converges towards the one that is nearest to the initial guess x_0. The periodicity of the objective function introduces additional complexity to the optimization task, as the method must deal with multiple potential minima.

Subsequently, in order to test our implementations of the algorithms with a larger value for the dimension n (i.e. $n = 10^3$), we defined three different starting points $\bar{y}$, $\bar{z}$, $\bar{w}$ that we present here, but that will also be used in the evaluation of the successive problems 27 (section 4.3) and 87 (section 4.4):

- $y_i = 1 \qquad i = 1, \ldots, n$
- $z_i = 0 \qquad i = 1, \ldots, n$
- $w_i = i \qquad i = 1, \ldots, n$

In the following table (Table 4) the results using these three different starting points are provided, for different values of c_1 and tol.

Method	parameters	f_k	$\| \nabla f(x_k) \|$	k	time	p
MN	$x_0 = \bar{y}$; $\rho = 0.5$; $c_1 = 10^{-4}$; $tol = 10^{-12}$;	-427.4045	1.5872e-13	2311	30.5935	0.9991
	$x_0 = \bar{z}$; $\rho = 0.5$; $c_1 = 10^{-4}$; $tol = 10^{-12}$;	-427.4045	1.1376e-13	5	0.0886	3.2860
	$x_0 = \bar{w}$; $\rho = 0.5$; $c_1 = 10^{-4}$; $tol = 10^{-12}$;	-427.4045	5.7695e-10	10000	193.6989	NaN
	$x_0 = \bar{y}$; $\rho = 0.5$; $c_1 = 10^{-7}$; $tol = 10^{-8}$;	-427.4045	1.5872e-13	2311	37.7952	0.9991
	$x_0 = \bar{z}$; $\rho = 0.5$; $c_1 = 10^{-7}$; $tol = 10^{-8}$;	-427.4045	1.1376e-13	5	0.0780	3.2860
	$x_0 = \bar{w}$; $\rho = 0.5$; $c_1 = 10^{-7}$; $tol = 10^{-8}$;	-427.4045	5.7695e-10	5544	70.7947	0.9994
MN&FinDiff	$x_0 = \bar{z}$; $\rho = 0.5$; $c_1 = 10^{-4}$; $tol = 10^{-3}$;	-427.4045	9.7424e-05	4	109.16230	3.6916

Table 2: Results obtained with the two considered methods, on the *Banded trigonometric* function, varying the parameters tol and c_1 and initializing with 3 different starting points. The highlighted row indicates the case where $kmax$ has been reached.

From Table 4 we can make some observations. The initial step that allowed the Modified Newton algorithm to converge in the smallest number of iterations is $\bar{z}$, while starting from $\bar{w}$ we can see how the maximum number of iterations was reached, when the tolerance on the gradient norm was set to 10^{-12}. Choosing a smaller c_1 generally did not influence the computational time or the gradient norm.

Concerning the version of the Modified Newton method where the gradient and the Hessian are computed using Finite Differences, only one test run is reported, with $\bar{z}$ as the starting point. This choice was made due to prohibitively long computational times when considering other starting points. In particular we can notice how, choosing as initial guess $x_0 = \bar{z}$, the value of the experimental rate of convergence p increases to approximately 3 for both methods, pointing at a cubic convergence. This underscores the significance of selecting a good starting point to initialize the methods, as will be further emphasized later.

Finally, it is worth noting that the value $p = \text{NaN}$ reported in the table 4, likely resulted from a division by 0.

4.3 Problem 27

As in the previous problem, we employed $kmax = 10000$ as the upper limit for outer iterations, and $btmax = 1000$ as the maximum number of inner iterations allowed during the backtracking phase on α_k. As in the previous case, to visualize the function and understand its characteristics, we tested the algorithms for $n = 2$, choosing the initial guess $x_0 = (-8, -5)$. The results are depicted in Figures 7 and 8.
In Table 3 the results of the different tests we conducted are shown.

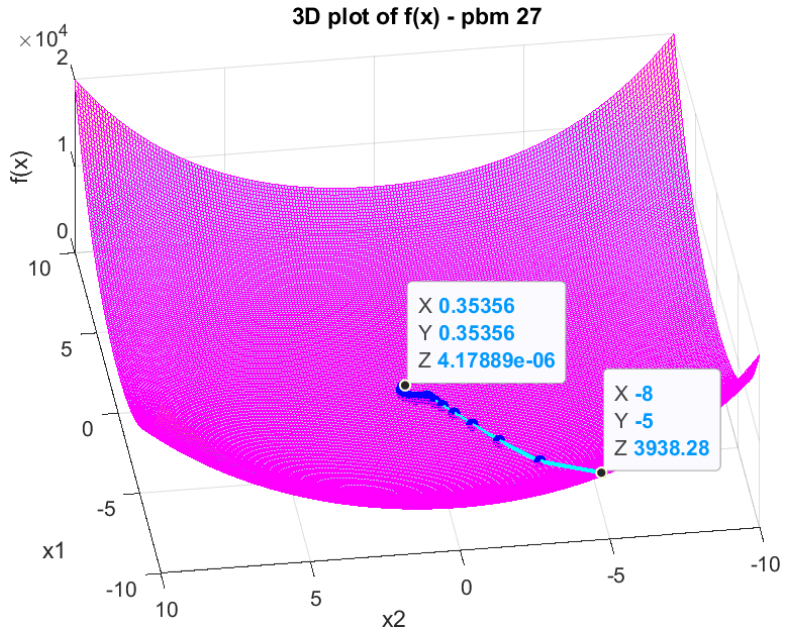


Figure 7: Sequence $\{x_k\}_{k\in\mathbb{N}}$ visualized in 3D, obtained with Modified Newton method, beginning with $\bar{x}_0 = (-8, -5)$.

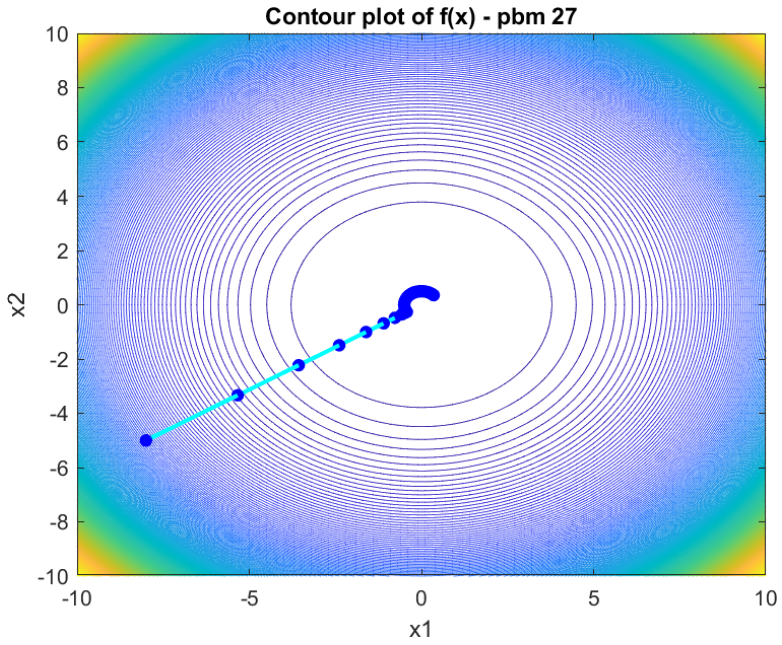


Figure 8: Sequence $\{x_k\}_{k\in\mathbb{N}}$ visualized in 2D, obtained with Modified Newton method, beginning with $\bar{x}_0 = (-8, -5)$.

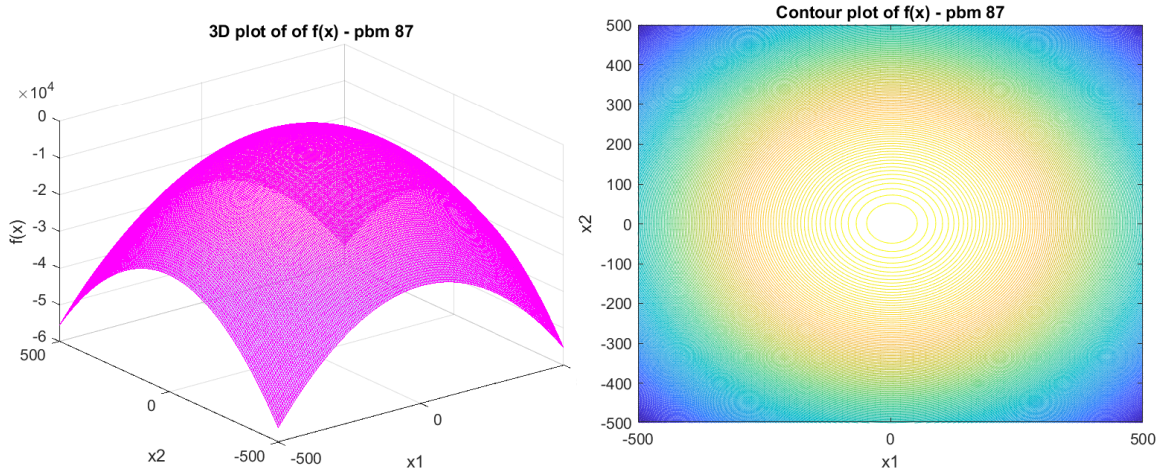
Method	parameters	f_k	$\| \nabla f(x_k) \|$	k	time	p
MN	$x_0 = \bar{y}$; $\rho = 0.5$; $c_1 = 10^{-4}$; $tol = 10^{-12}$;	0.0048	2.6295e-15	15	1.1051	1.7550
	$x_0 = \bar{z}$; $\rho = 0.5$; $c_1 = 10^{-4}$; $tol = 10^{-12}$;	0.0048	6.5825e-14	110	5.1190	1.0860
	$x_0 = \bar{w}$; $\rho = 0.5$; $c_1 = 10^{-4}$; $tol = 10^{-12}$;	0.0048	4.8873e-14	42	2.0452	1.0643
	$x_0 = \bar{y}$; $\rho = 0.5$; $c_1 = 10^{-7}$; $tol = 10^{-8}$;	0.0048	2.6295e-15	15	0.7605	1.7550
	$x_0 = \bar{z}$; $\rho = 0.5$; $c_1 = 10^{-7}$; $tol = 10^{-8}$;	0.0048	6.5825e-14	110	5.1912	1.0086
	$x_0 = \bar{w}$; $\rho = 0.5$; $c_1 = 10^{-7}$; $tol = 10^{-8}$;	0.0048	2.2546e-09	41	1.9914	1.0023
MN&FinDiff	$x_0 = \bar{y}$; $\rho = 0.5$; $c_1 = 10^{-4}$; $tol = 10^{-8}$;	0.0048	9.1666e-10	16	353.7333	1.3206
	$x_0 = \bar{w}$; $\rho = 0.5$; $c_1 = 10^{-4}$; $tol = 10^{-8}$;	0.0048	1.3299e-09	303	1648.1033	0.9830

Table 3: Results obtained with the two considered methods, on the function f of problem 27, varying the parameters tol and c_1 and initializing with 3 different starting points.

In this case we have witnessed all successful runs with the chosen parameters configuration when applying the standard Modified Newton method. Regarding the method with Finite Differences, as in the case of problem 16, in order to conduct successful tests, increasing the tolerance on the gradient norm and choosing optimal values for the starting points were two necessary conditions. Additionally, comparing the performance of this method with that of the Modified Newton method, we can notice a significant increase in computational time and the number of iterations required, as expected.

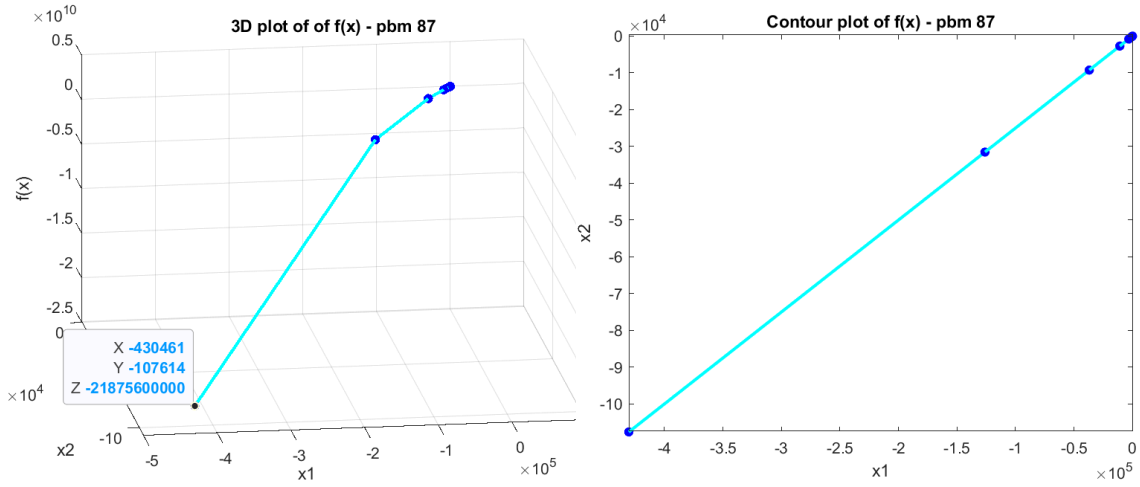
4.4 Problem 87: *Ascher and Russel boundary value problem*

Here, unlike in the other problems, as the objective function does not exhibit a minimum, leading to an impossibility of method convergence, we reduced the maximum number of iterations to $kmax = 100$. Regarding the upper limit for inner iterations, during backtracking, we maintained it at $btmax = 1000$. For clarity of representation, we proceed as follows: first, we display the profile of the objective function surface and its two-dimensional contour plot (Figure 9); then, we present the sequences $\{x_k\}_{k\in\mathbb{N}}$ provided by the two methods (MN in Figure 10, MN with Finite Differences in Figure 11), both in three-dimensional and two-dimensional views.



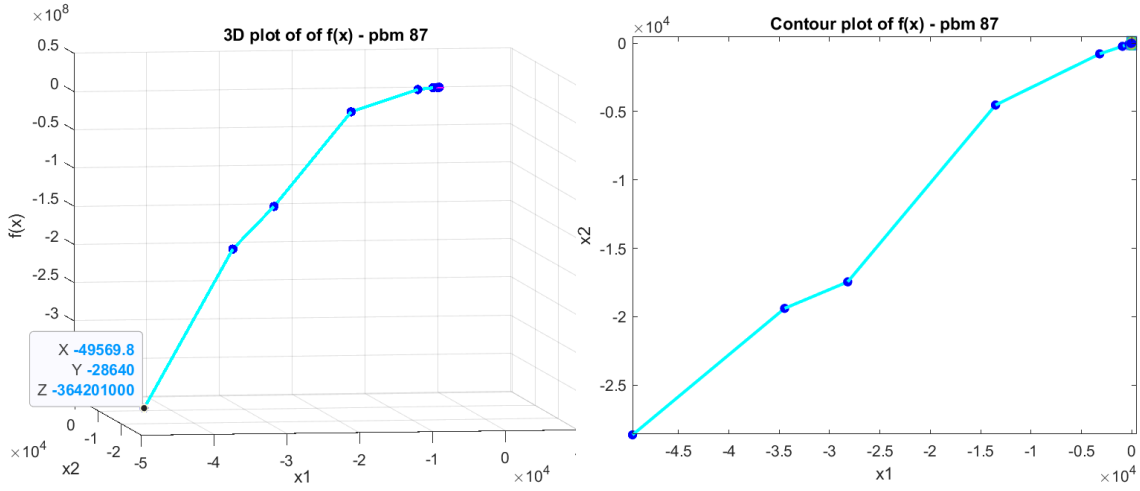
(a) 3D plot (b) Contour plot

Figure 9: Graphic representations in $\mathbb{R}^2$ of the objective function of problem 87.



(a) 3D plot (b) Contour plot

Figure 10: Sequence $\{x_k\}_{k\in\mathbb{N}}$ visualized in 2D and in 3D, obtained with Modified Newton method, and beginning with $(1, 1)$



(a) 3D plot (b) Contour plot

Figure 11: Sequence $\{x_k\}_{k\in\mathbb{N}}$ visualized in 2D and in 3D, obtained with Modified Newton method with Finite Differences, beginning with $(1, 1)$

Method	parameters	f_k	$\| \nabla f(x_k) \|$	k	time
MN	$x_0 = \bar{y}$; $\rho = 0.5$; $c_1 = 10^{-4}$; $tol = 10^{-8}$;	-5.9241e+10	486.3032	100	1.2730
	$x_0 = \bar{z}$; $\rho = 0.5$; $c_1 = 10^{-4}$; $tol = 10^{-8}$;	-5.9241e+10	486.3052	100	1.2560
	$x_0 = \bar{w}$; $\rho = 0.5$; $c_1 = 10^{-4}$; $tol = 10^{-8}$;	-5.9005e+10	485.3355	100	1.2704
MN&FinDiff	$x_0 = \bar{y}$; $\rho = 0.5$; $c_1 = 10^{-4}$; $tol = 10^{-8}$;	-1.7893e+04	0.7560	100	966.1059
	$x_0 = \bar{z}$; $\rho = 0.5$; $c_1 = 10^{-8}$; $tol = 10^{-3}$;	-1.7830e+04	0.7557	100	960.3686
	$x_0 = \bar{w}$; $\rho = 0.5$; $c_1 = 10^{-8}$; $tol = 10^{-3}$;	-1.9160e+03	0.7126	100	868.8868

Table 4: Results obtained with the two considered methods, on the *Banded trigonometric* function, varying the parameters *tol* and c_1 and initializing with 3 different starting points. The highlighted row indicates the case where $kmax$ has been reached.

As mentioned earlier, the objective function $F(x)$ of this problem is strictly concave and therefore does not admit a minimum (recall Figure 9). In this specific scenario, we observe the power of the Modified Newton Method which, through adjustments made on the Hessian matrix, effectively navigates along descent directions towards smaller values of F, despite the challenges posed by the non-positive definite Hessian. This behaviour is in contrast to what would occur with the classical version of the algorithm, which would instead converge in a single iteration to the unique global maximum point of F.

It is essential to note that even if the Modified Newton Method demonstrates remarkable adaptability by moving in the right direction, actually it does not converge in this case, due to the absence of a minimum for the function F. Moreover, since the gradient will never diminish sufficiently (the norm of the gradient increases at each iteration indeed), the stopping criterion on the gradient norm will never be satisfied and, for each run, the maximum number of iterations $kmax$ will be reached. However, we can say that this behaviour shows the method's elasticity and ability to adjust even in challenging optimization tasks.

Finally, since in this scenario neither method can converge, we did not attempt to analyse their respective rates of convergence.

5 Conclusions

Generally speaking, we can assert that both methods studied yield qualitatively satisfactory results. However, in terms of efficiency, the Modified Newton method undeniably outperforms the Modified Newton method with Finite Differences, converging much more rapidly under equivalent parameter configurations and starting point x_0.

The slowness of the latter algorithm becomes particularly apparent with large n, as the computational cost of approximating the gradient and Hessian with finite differences is approximately $O(n)$ and $O(n^2)$ respectively (due to the execution of several *for* loops). Indeed, for this reason, in the case of problems 16 and 27, we limited the tests on the Modified Newton method with Finite Differences to a restricted number of scenarios. Among the initial points $\bar{y}$, $\bar{z}$, and $\bar{w}$ considered, in fact, not all of them allowed convergence within a reasonable time. This aspect

leads us to reflect on the importance of initializing an iterative method with an appropriate starting point. In $\mathbb{R}^n$, especially with a large n, the selection of x_0 indeed proves crucial, as we have empirically witnessed.

Lastly, concerning the rate of convergence of the two algorithms, our experimental data yielded consistent results. On average, it appears that the convergence speed is *superlinear* for both methods ($p > 1$), with the Modified Newton method demonstrating slightly superior performance compared to the Modified Newton method with Finite Differences.

A Codes

A.1 Modified Newton

```
function [xk, fk, gradfk_norm, k, xseq, btseq] = ...
    modified_newton_bcktrck(x0, f, gradf, Hessf, kmax, ...
    tolgrad, c1, rho, btmax)
%
% [xk, fk, gradfk_norm, k, xseq] = ...
%     moodified_newton_bcktrck(x0, f, gradf, Hessf, kmax, ...
%     tolgrad, c1, rho, btmax)
%
% Function that performs the modified newton optimization method,
% implementing the backtracking strategy.
%
% INPUTS:
% x0 = n-dimensional column vector;
% f = function handle that describes a function R^n->R;
% gradf = function handle that describes the gradient of f;
% Hessf = function handle that describes the Hessian of f;
% kmax = maximum number of iterations permitted;
% tolgrad = value used as stopping criterion w.r.t. the norm of
    the
% gradient;
% c1 = the factor of the Armijo condition that must be a scalar
    in (0,1);
% rho = fixed factor, lesser than 1, used for reducing alpha0;
% btmax = maximum number of steps for updating alpha during the
% backtracking strategy.
%
% OUTPUTS:
% xk = the last x computed by the function;
% fk = the value f(xk);
% gradfk_norm = value of the norm of gradf(xk)
% k = index of the last iteration performed
% xseq = n-by-k matrix where the columns are the xk computed
    during the
% iterations
```

```
% btseq = 1-by-k vector where elements are the number of
    backtracking
% iterations at each optimization step.
%

n = length(x0);
% Function handle for the armijo condition
farmijo = @(fk, alpha, gradfk, pk) ...
    fk + c1 * alpha * gradfk' * pk;

% Initializations
xseq = zeros(length(x0), kmax);
btseq = zeros(1, kmax);

xk = x0;
fk = f(xk);
k = 0;
gradfk = gradf(xk);
gradfk_norm = norm(gradfk);

while k < kmax && gradfk_norm >= tolgrad
    % Correct the Hessian Hessf(xk) with Ek s.t. Bk = Hessf(xk) +
        Ek is
    % positive definite.
    Hessfk = Hessf(xk);
    beta = norm(Hessfk,"fro");
    if beta==0
        beta=sqrt(eps);
    end

    if min(diag(Hessfk))>0
        tauk=0;
    else
        tauk = beta/2;
    end

    notPosDef = true;
    while notPosDef
```

```
68          Bk = Hessfk + tauk * eye(n);
69          if issymmetric(Bk) && all(eig(Bk) > 0)
70             notPosDef = false;
71          else
72             tauk = max(2*tauk, beta*0.5);
73          end
74      end
75
76      % Compute the descent direction as solution of Bk*pk = -
           gradf(xk)
77      % Linear System above SOLVED WITH Cholesky decomposition:
78      R = chol(Bk);
79      pk = R\(R'\-gradf(xk));
80
81      % Backtracking with Armijo condition
82      % Reset the value of alpha
83      alpha = 1;
84
85      % Compute the candidate new xk
86      xnew = xk + alpha * pk;
87      % Compute the value of f in the candidate new xk
88      fnew = f(xnew);
89
90      bt = 0;
91      % Backtracking strategy:
92      % 2nd condition is the Armijo condition not satisfied
93      while bt < btmax && fnew > farmijo(fk, alpha, gradfk, pk)
94          % Reduce the value of alpha
95          alpha = rho * alpha;
96          % Update xnew and fnew w.r.t. the reduced alpha
97          xnew = xk + alpha * pk;
98          fnew = f(xnew);
99
100         % Increase the counter by one
101         bt = bt + 1;
102     end
103
104     % Update xk, fk, gradfk_norm
```

```
105     xk = xnew;
106     fk = fnew;
107     gradfk = gradf(xk);
108     gradfk_norm = norm(gradfk);
109
110     % Increase the step by one
111     k = k + 1;
112
113     % Store current xk in xseq
114     xseq(:, k) = xk;
115     % Store bt iterations in btseq
116     btseq(k) = bt;
117
118 end
119
120 % "Cut" xseq and btseq to the correct size
121 xseq = xseq(:, 1:k);
122 btseq = btseq(1:k);
123
124 end
```

Listing 1: Modified Newton

A.2 Modified Newton with Finite Differences

```
1  function [xk, fk, gradfk_norm, k, xseq, btseq] = ...
2      modified_newton_bcktrck_findiff(x0, f, kmax, ...
3      tolgrad, c1, rho, btmax)
4  %
5  % [xk, fk, gradfk_norm, k, xseq] = ...
6  %      moodified_newton_bcktrck_findiff(x0, f, kmax, ...
7  %      tolgrad, c1, rho, btmax)
8  %
9  % Function that performs the modified newton optimization method,
10 % implementing the backtracking strategy. Finite differences are
       used to
11 % approximate the gradient and the Hessian of f.
12 %
```

```
13 % INPUTS:
14 % x0 = n-dimensional column vector;
15 % f = function handle that describes a function R^n->R;
16 % kmax = maximum number of iterations permitted;
17 % tolgrad = value used as stopping criterion w.r.t. the norm of
       the
18 % gradient;
19 % c1 = the factor of the Armijo condition that must be a scalar
       in (0,1);
20 % rho = fixed factor, lesser than 1, used for reducing alpha0;
21 % btmax = maximum number of steps for updating alpha during the
22 % backtracking strategy.
23 %
24 % OUTPUTS:
25 % xk = the last x computed by the function;
26 % fk = the value f(xk);
27 % gradfk_norm = value of the norm of gradf(xk)
28 % k = index of the last iteration performed
29 % xseq = n-by-k matrix where the columns are the xk computed
       during the
30 % iterations
31 % btseq = 1-by-k vector where elements are the number of
       backtracking
32 % iterations at each optimization step.
33 %
34
35 n = length(x0);
36 % Function handle for the armijo condition
37 farmijo = @(fk, alpha, gradfk, pk) ...
38     fk + c1 * alpha * gradfk' * pk;
39
40 %%%%%% FINITE DIFFERENCES application
41 gradf = @(x) findiff_grad(f, x, sqrt(eps), 'c');
42 Hessf = @(x) findiff_Hess(f, x, sqrt(sqrt(eps)));
43
44 % Initializations
45 xseq = zeros(length(x0), kmax);
46 btseq = zeros(1, kmax);
```

```
47
48 xk = x0;
49 fk = f(xk);
50 k = 0;
51 gradfk = gradf(xk);
52 gradfk_norm = norm(gradfk);
53
54 while k < kmax && gradfk_norm >= tolgrad
55     % Correct the Hessian Hessf(xk) with Ek s.t. Bk = Hessf(xk) +
           Ek is
56     % positive definite.
57     Hessfk = Hessf(xk);
58     beta = norm(Hessfk,"fro");
59
60     if beta==0
61         beta = sqrt(eps);
62     end
63
64     if min(diag(Hessfk))>0
65         tauk=0;
66     else
67         tauk = beta/2;
68     end
69
70     notPosDef = true;
71     while notPosDef
72         Bk = Hessfk + tauk * eye(n);
73         if issymmetric(Bk) && all(eig(Bk) > 0)
74            notPosDef = false;
75         else
76            tauk = max(2*tauk, beta*0.5);
77         end
78     end
79
80     % Compute the descent direction as solution of Bk*pk = -
          gradf(xk)
81     % Linear System above SOLVED WITH Cholesky decomposition:
82     R = chol(Bk);
```

```
83      pk = R\(R'\-gradf(xk));
84
85      % Backtracking with Armijo condition
86      % Reset the value of alpha
87      alpha = 1;
88
89      % Compute the candidate new xk
90      xnew = xk + alpha * pk;
91      % Compute the value of f in the candidate new xk
92      fnew = f(xnew);
93
94      bt = 0;
95      % Backtracking strategy:
96      % 2nd condition is the Armijo condition not satisfied
97      while bt < btmax && fnew > farmijo(fk, alpha, gradfk, pk)
98          % Reduce the value of alpha
99          alpha = rho * alpha;
100         % Update xnew and fnew w.r.t. the reduced alpha
101         xnew = xk + alpha * pk;
102         fnew = f(xnew);
103
104         % Increase the counter by one
105         bt = bt + 1;
106     end
107
108     % Update xk, fk, gradfk_norm
109     xk = xnew;
110     fk = fnew;
111     gradfk = gradf(xk);
112     gradfk_norm = norm(gradfk);
113
114     % Increase the step by one
115     k = k + 1;
116
117     % Store current xk in xseq
118     xseq(:, k) = xk;
119     % Store bt iterations in btseq
120     btseq(k) = bt;
```

```
121
122 end
123
124 % "Cut" xseq and btseq to the correct size
125 xseq = xseq(:, 1:k);
126 btseq = btseq(1:k);
127
128 end
```

Listing 2: Modified Newton with Finite Differences

```
1     function [gradfx] = findiff_grad(f, x, h, type)
2 %
3 % function [gradf] = findiff_grad(f, x, h, type)
4 %
5 % Function that approximate the gradient of f in x (column vector
      ) with the
6 % finite difference (forward/centered) method.
7 %
8 % INPUTS:
9 % f = function handle that describes a function R^n->R;
10 % x = n-dimensional column vector;
11 % h = the h used for the finite difference computation of gradf
12 % type = 'fw' or 'c' for choosing the forward/centered finite
      difference
13 % computation of the gradient.
14 %
15 % OUTPUTS:
16 % gradfx = column vector (same size of x) corresponding to the
      approximation
17 % of the gradient of f in x.
18
19 gradfx = zeros(size(x));
20
21 switch type
22     case 'fw'
23         for i=1:length(x)
24             xh = x;
25             xh(i) = xh(i) + h;
```

```
26              gradfx(i) = (f(xh) - f(x))/ h;
27              % ALTERNATIVELY (no xh)
28              % gradf(i) = (f([x(1:i-1); x(i)+h; x(i+1:end)]) - f(x
                    ))/h;
29          end
30      case 'c'
31          for i=1:length(x)
32              xh_plus = x;
33              xh_minus = x;
34              xh_plus(i) = xh_plus(i) + h;
35              xh_minus(i) = xh_minus(i) - h;
36              gradfx(i) = (f(xh_plus) - f(xh_minus))/(2 * h);
37              % ALTERNATIVELY (no xh_plus and xh_minus)
38              % gradf(i) = ((f([x(1:i-1); x(i)+h; x(i+1:end)]) -
                    ...
39              %     f([x(1:i-1); x(i)-h; x(i+1:end)]))/(2*h);
40          end
41      otherwise % repeat the 'fw' case
42          for i=1:length(x)
43              xh = x;
44              xh(i) = xh(i) + h;
45              gradfx(i) = (f(xh) - f(x))/h;
46              % ALTERNATIVELY (no xh)
47              % gradf(i) = (f([x(1:i-1); x(i)+h; x(i+1:end)]) - f(x
                    ))/h;
48          end
49  end
50
51  end
```

Listing 3: findiff_grad

```
1       function [Hessfx] = findiff_Hess(f, x, h)
2   %
3   % [Hessf] = findiff_Hess(f, x, h)
4   %
5   % Function that approximate the Hessian of f in x (column vector)
        with the
6   % finite difference method.
```

```
7  % ATTENTION: we assume a regular f (C^2) s.t. Hessf is symmetric.
8  %
9  % INPUTS:
10 % f = function handle that describes a function R^n->R;
11 % x = n-dimensional column vector;
12 % h = the h used for the finite difference computation of Hessf
13 %
14 % OUTPUTS:
15 % Hessfx = n-by-n matrix corresponding to the approximation of
       the Hessian
16 % of f in x.
17
18 n = length(x);
19 Hessfx = zeros(n);
20
21 for j=1:n
22     % Elements on the diagonal
23     xh_plus = x;
24     xh_minus = x;
25     xh_plus(j) = xh_plus(j) + h;
26     xh_minus(j) = xh_minus(j) - h;
27     Hessfx(j,j) = (f(xh_plus) - 2*f(x) + f(xh_minus))/(h^2);
28     % ALTERNATIVELY (no xh_plus and xh_minus)
29     % Hessf(j,j) = (f([x(1:j-1); x(j)+h; x(j+1:end)]) - 2*f(x) +
           ...
30     %     f([x(1:j-1); x(j)-h; x(j+1:end)]))/(h^2);
31     for i=j+1:n
32         xh_plus_ij = x;
33         xh_plus_ij([i, j]) = xh_plus_ij([i, j]) + h;
34         xh_plus_i = x;
35         xh_plus_i(i) = xh_plus_i(i) + h;
36         xh_plus_j = x;
37         xh_plus_j(j) = xh_plus_j(j) + h;
38         Hessfx(i,j) = (f(xh_plus_ij) - ...
39             f(xh_plus_i) - f(xh_plus_j) + f(x))/(h^2);
40
41         % ALTERNATIVELY (no xh_plus_i/j)
42         % Hessf(i,j) = (f(xh_plus_ij) - ...
```

```
43          %      f([x(1:i-1); x(i)+h; x(i+1:end)]) - ...
44          %      f([x(1:j-1); x(j)+h; x(j+1:end)]) + f(x))/(h^2);
45
46          Hessfx(j,i)=Hessfx(i,j);
47      end
48  end
49
50
51  end
```

Listing 4: findiff_Hess

A.3 Rosenbrock Function

```
1   clear all
2   close all
3   clc
4   load('rosenbrock.mat')
5   n = 2;
6   rho = 0.5;
7   c1 = 1e-7;
8   kmax = 3000;
9   tolgrad = 1e-8;
10  x0 = [1.2; 1.2];
11  %x0 = [-1.2; 1]
12  btmax = 1000;
13
14  % % modified newton
15  % tic
16  % [xk, fk, gradfk_norm, k, xseq, btseq] = modified_newton_bcktrck
        (x0, f, gradf, Hessf, kmax, tolgrad, c1, rho, btmax)
17  % toc
18
19  % modified newton with finite differences
20  tic
21  [xk, fk, gradfk_norm, k, xseq, btseq] =
       modified_newton_bcktrck_findiff(x0, f, kmax, tolgrad, c1, rho,
        btmax)
```

```
toc

%rate of convergence
dist = zeros(1,k);
for i=1:(k-1)
    dist(i) = norm(xseq(:,i)-xseq(:,i+1));
end
y = log(dist(2:k-1));
x = log(dist(1:k-2));
mdl = fitlm(x, y, 'Intercept',false)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % Plot function
% % Define the range for x1 and x2
% x1_range = linspace(-20, 20, 200);
% x2_range = linspace(-20, 20, 200);
% [X1, X2] = meshgrid(-5:0.1:5, -5:0.1:5);
%
% % Compute the function's values on the grid
% Z = zeros(size(X1));
% for i = 1:size(X1, 1)
%     for j = 1:size(X1, 2)
%         Z(i, j) = f([X1(i, j); X2(i, j)]);
%     end
% end
%
% figure;
% h = surf(X1, X2, Z, 'EdgeColor', 'magenta');
% h.FaceAlpha = 0.3;
% xk = xseq(1, :);
% yk = xseq(2, :);
% hold on;
% xk = [x0(1) xk];
% yk = xseq(2, :);
% yk = [x0(2) yk];
% hold on;
% % Initialize an array to store the function values
% zk = zeros(size(xk));
```

```
% % Compute the function values for each column of xseq
% zk(1) = f(x0);
% for i = 1:(numel(xk)-1)
%     zk(i+1) = f(xseq(:, i));
% end
% hold on;
% plot3(xk, yk, zk, 'LineWidth', 2, 'Color', 'cyan', 'LineStyle',
    '-');
% scatter3(xk, yk, zk, 'blue', 'filled');
% hold off;
% % Add labels and title
% xlabel('x1');
% ylabel('x2');
% zlabel('f(x)');
% title('3D plot of the Rosenbrock function');
%
% % Plot the function as a contour plot
% figure;
% contour(X1, X2, Z, 200);
% % Add labels and title
% xlabel('x1');
% ylabel('x2');
% title('Contour plot of the Rosenbrock function');
% hold on;
% plot3(xk,yk, zk, 'LineWidth', 2, 'Color', 'cyan', 'LineStyle',
    '-');
% scatter3(xk,yk, zk, 'blue', 'filled');
% hold off;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Listing 5: Rosenbrock

A.4 Problem 16

```
n = 1000;
f = @(x) compute_f1(x);
gradf = @(x) [(1:length(x(1:n-1)))'.*sin(x(1:n-1)) + 2*cos(x(1:n
    -1)); n*sin(x(n)) - (n-1)*cos(x(n))];
```

```
4  Hessf = @(x) diag([(1:length(x(1:n-1)))'.*cos(x(1:n-1)) - 2*sin(x
       (1:n-1)); n*cos(x(n)) + (n-1)*sin(x(n))]);
5
6  rho = 0.5;
7  c1 = 1e-4;
8  kmax = 10000;
9  tolgrad = 1e-3;
10 %x0 = [8;9];
11 %x0 = ones(n,1);
12 x0 = zeros(n,1);
13 %x0 = (1:n)';
14 btmax = 1000;
15
16 % % modified newton
17 % tic
18 % [xk, fk, gradfk_norm, k, xseq, ~] = modified_newton_bcktrck(x0,
       f, gradf, Hessf, kmax, tolgrad, c1, rho, btmax)
19 % toc
20
21 % modified newton with finite differences
22 tic
23 [xk, fk, gradfk_norm, k, xseq, btseq] =
       modified_newton_bcktrck_findiff(x0, f, kmax, tolgrad, c1, rho,
       btmax)
24 toc
25
26 %rate of convergence
27 dist = zeros(1,k);
28 for i=1:(k-1)
29     dist(i) = norm(xseq(:,i)-xseq(:,i+1));
30 end
31 y = log(dist(2:k-1));
32 x = log(dist(1:k-2));
33 mdl = fitlm(x, y, 'Intercept',false)
34
35 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
36 % % Plot function
37 % % Define the range for x1 and x2
```

```
% x1_range = linspace(-10, 10, 200);
% x2_range = linspace(-10, 10, 200);
% [X1, X2] = meshgrid(-10:0.1:10, -10:0.1:10);
%
% Compute the function's values on the grid
% Z = zeros(size(X1));
% for i = 1:size(X1, 1)
%     for j = 1:size(X1, 2)
%         Z(i, j) = f([X1(i, j); X2(i, j)]);
%     end
% end
%
% figure;
% h = surf(X1, X2, Z, 'EdgeColor', 'magenta');
% h.FaceAlpha = 0.3;
% xk = xseq(1, :);
% xk = [x0(1) xk]
% yk = xseq(2, :);
% yk = [x0(2) yk];
% hold on;
% % Initialize an array to store the function values
% zk = zeros(size(xk));
% % Compute the function values for each column of xseq
% zk(1) = f(x0);
% for i = 1:(numel(xk)-1)
%     zk(i+1) = f(xseq(:, i));
% end
% hold on;
% plot3(xk, yk, zk, 'LineWidth', 4, 'Color', 'cyan', 'LineStyle',
    '-');
% scatter3(xk, yk, zk, 50, 'blue', 'filled');
% hold off;
% % Add labels and title
% xlabel('x1');
% ylabel('x2');
% zlabel('f(x)');
% title('3D plot of of f(x) - pbm 16');
%
```

```
75 % % Plot the function as a contour plot
76 % figure;
77 % contour(X1, X2, Z, 200);
78 % % Add labels and title
79 % xlabel('x1');
80 % ylabel('x2');
81 % title('Contour plot of f(x) - pbm 16');
82 % hold on;
83 % plot3(xk,yk, zk, 'LineWidth', 3, 'Color', 'cyan', 'LineStyle',
       '-');
84 % scatter3(xk,yk, zk, 20, 'blue', 'filled');
85 % hold off;
86 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
87
88 function result = compute_f1(x)
89     n = length(x);
90     x_b = [0;x(1:(n-1))];
91     x_f = [x(2:n);0];
92     result = sum([1:n]'.*((1 - cos(x)) + sin(x_b) - sin(x_f)));
93 end
```

Listing 6: Problem 16

A.5 Problem 27

```
1  clear all
2  close all
3  clc
4
5  n = 1000;
6  f = @(x) 0.5*(sum(((x-1).^2)/100000) + (sum(x.^2)-0.25)^2);
7  gradf = @(x) (x-1)/100000 + 2*x*sum(x.^2) - x*0.5;
8  Hessf = @(x) compute_H(x);
9  rho = 0.5;
10 c1 = 1e-7;
11 kmax = 10000;
12 tolgrad = 1e-8;
13 %x0 = ones(n,1);
```

```
%x0 = zeros(n,1);
x0 = (1:n)';
btmax = 1000;

% % modified newton
% tic
% [xk, fk, gradfk_norm, k, xseq, btseq] = modified_newton_bcktrck
    (x0, f, gradf, Hessf, kmax, tolgrad, c1, rho, btmax);
% toc

%modified newton with finite differences
tic
[xk, fk, gradfk_norm, k, xseq, btseq] =
   modified_newton_bcktrck_findiff(x0, f, kmax, tolgrad, c1, rho,
    btmax)
toc

%rate of convergence
dist = zeros(1,k);
for i=1:(k-1)
    dist(i) = norm(xseq(:,i)-xseq(:,i+1));
end
y = log(dist(2:k-1));
x = log(dist(1:k-2));
mdl = fitlm(x, y, 'Intercept',false)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % Plot function
% % Define the range for x1 and x2
% x1_range = linspace(-10, 10, 200);
% x2_range = linspace(-10, 10, 200);
% [X1, X2] = meshgrid(-10:0.1:10, -10:0.1:10);
%
% % Compute the function's values on the grid
% Z = zeros(size(X1));
% for i = 1:size(X1, 1)
%     for j = 1:size(X1, 2)
%         Z(i, j) = f([X1(i, j); X2(i, j)]);
```

```
%     end
% end
%
% figure;
% h = surf(X1, X2, Z, 'EdgeColor', 'magenta');
% h.FaceAlpha = 0.3;
% xk = xseq(1, :);
% yk = xseq(2, :);
% hold on;
% xk = [x0(1) xk];
% yk = xseq(2, :);
% yk = [x0(2) yk];
% hold on;
% % Initialize an array to store the function values
% zk = zeros(size(xk));
% % Compute the function values for each column of xseq
% zk(1) = f(x0);
% for i = 1:(numel(xk)-1)
%     zk(i+1) = f(xseq(:, i));
% end
% hold on;
% plot3(xk, yk, zk, 'LineWidth', 2, 'Color', 'cyan', 'LineStyle',
    '-');
% scatter3(xk, yk, zk, 'blue', 'filled');
% hold off;
% % Add labels and title
% xlabel('x1');
% ylabel('x2');
% zlabel('f(x)');
% title('3D plot of f(x) - pbm 27');
%
% % Plot the function as a contour plot
% figure;
% contour(X1, X2, Z, 200);
% % Add labels and title
% xlabel('x1');
% ylabel('x2');
% title('Contour plot of f(x) - pbm 27');
```

```
% hold on;
% plot3(xk,yk, zk, 'LineWidth', 2, 'Color', 'cyan', 'LineStyle',
    '-');
% scatter3(xk,yk, zk, 'blue', 'filled');
% hold off;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function Hess = compute_H(x)
    n = length(x);
    Hess = zeros(n,n);
    for i=1:n
        for j=1:n
            if i==j
                Hess(i,j) = 1e-5 + 2*sum([x(1:j-1);x(j+1:n)].^2)
                    + 6*x(i)^2 - 0.5;
            else
                Hess(i,j) = 4*x(i)*x(j);
            end
        end
    end
end
```

Listing 7: Problem 27

A.6 Problem 87

```
% Ascher and Russel boundary value problem
clear all
close all
clc

n = 1000;
f = @(x) compute_f(x);
gradf = @(x) [ (0.5 - (2*x(1))/(n+1)^2 + 1/(2*(n+1))); ((2*x(2:n
    -1))/(n+1)^2) ; (0.5 - (2*x(n))/(n+1)^2 - 1/(2*(n+1))) ];
Hessf = @(x) diag((-2/(n+1)^2)*ones(n,1));

rho = 0.5;
```

```
c1 = 1e-4;
kmax = 100;
tolgrad = 1e-8;
x0 = ones(n,1);
%x0 = zeros(n,1);
%x0 = (1:n)';
btmax = 1000;

% % modified newton
% tic
% [xk, fk, gradfk_norm, k, ~, ~] = modified_newton_bcktrck(x0, f,
    gradf, Hessf, kmax, tolgrad, c1, rho, btmax)
% toc

%modified newton with finite differences
tic
[xk, fk, gradfk_norm, k, ~, ~] = modified_newton_bcktrck_findiff(
   x0, f, kmax, tolgrad, c1, rho, btmax)
toc

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % Plot function
% % Define the range for x1 and x2
% [X1, X2] = meshgrid(-500:5:500, -500:5:500);
%
% % Compute the function's values on the grid
% Z = zeros(size(X1));
% for i = 1:size(X1, 1)
%     for j = 1:size(X1, 2)
%         Z(i, j) = compute_f([X1(i, j); X2(i, j)]);
%     end
% end
%
% figure;
% h = surf(X1, X2, Z, 'EdgeColor', 'magenta');
% h.FaceAlpha = 0.2;
% xk = xseq(1, :);
% yk = xseq(2, :);
```

```
48 % hold on;
49 % xk = [x0(1) xk];
50 % yk = xseq(2, :);
51 % yk = [x0(2) yk];
52 % hold on;
53 % % Initialize an array to store the function values
54 % zk = zeros(size(xk));
55 % % Compute the function values for each column of xseq
56 % zk(1) = f(x0);
57 % for i = 1:(numel(xk)-1)
58 %     zk(i+1) = f(xseq(:, i));
59 % end
60 % hold on;
61 % plot3(xk, yk, zk, 'LineWidth', 2, 'Color', 'cyan', 'LineStyle',
       '-');
62 % scatter3(xk, yk, zk, 'blue', 'filled');
63 % hold off;
64 % % Add labels and title
65 % xlabel('x1');
66 % ylabel('x2');
67 % zlabel('f(x)');
68 % title('3D plot of of f(x) - pbm 87');
69 %
70 % % Plot the function as a contour plot
71 % figure;
72 % contour(X1, X2, Z, 200);
73 % % Add labels and title
74 % xlabel('x1');
75 % ylabel('x2');
76 % title('Contour plot of f(x) - pbm 87');
77 % hold on;
78 % plot3(xk,yk, zk, 'LineWidth', 2, 'Color', 'cyan', 'LineStyle',
       '-');
79 % scatter3(xk,yk, zk, 'blue', 'filled');
80 % hold off;
81 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
82
83 function result = compute_f(x)
```

```
84      n = length(x);
85      x_b = [0;x(1:(n-1))];
86      x_f = [x(2:n);0.5];
87      result = 0.5*sum(2*x - ((2 /(1+n)^2)*(x.^2 + (x_f-x_b)*(1+n)
            /2)) - x_b - x_f);
88  end
```

Listing 8: Problem 87