Numerical Optimization for Large Scale Problems

Report

Assignment on Unconstrained Optimization

Di Battista Simona — 302689
Rostagno Andrea — 349152

June 17, 2025

Contents

1 Introduction

The goal of this project is to implement and compare two numerical methods for unconstrained optimization: the Modified Newton method and the Nelder-Mead method.

First these algorithms are tested on the standard 2-dimensional Rosenbrock function using two different initial conditions, in order to validate their implementation. Subsequently, they are applied to three benchmark problems selected from the test set for unconstrained optimization proposed in [**?**].

For benchmark problems, in accordance with the assignment instructions, the Nelder-Mead method is tested in low dimensions $n = 10, 26, 50$, while the Modified Newton method is evaluated on $n = 10^3, 10^4, 10^5$. Furthermore, for each test function and each method, a fixed starting point $\bar{x}$ suggested in [**?**] is used, together with 10 uniformly randomly generated starting points, sampled in the hypercube $[\bar{x}_1 - 1, \bar{x}_1 + 1] \times \cdots \times [\bar{x}_n - 1, \bar{x}_n + 1] \subset \mathbb{R}^n$. Performance is assessed in terms of number of successful runs, total iterations, CPU time. Also an experimental rate of convergence is computed, starting from the definition of rate of convergence. First it was defined $e^{(k)} = x^{(k)} - x^\star$, where $x^\star$ is the optimal minimizer; since the value of the optimal minimizer is not known a priori, it is replaced in the formula by the last available approximation, obtaining $e^{(k)} \approx x^{(k)} - x^{(k-1)}$. For k large enough

$$\frac{\left\|e^{(k+1)}\right\|}{\|e^{(k)}\|} \approx \left\|\frac{e^{(k)}}{e^{(k-1)}}\right\|^\rho \Rightarrow \rho \approx \frac{\log\left(\left\|e^{(k+1)}\right\| \Big/ \left\|e^{(k)}\right\|\right)}{\log\left(\|e^{(k)}\| \Big/ \|e^{(k-1)}\|\right)}.$$

Ultimately, the experimental convergence rate is determined by implementing the following formula:

$$\rho \approx \frac{\log\left(\|x^{(k+1)} - x^{(k)}\| \Big/ \|x^{(k)} - x^{(k-1)}\|\right)}{\log\left(\|x^{(k)} - x^{(k-1)}\| \Big/ \|x^{(k-1)} - x^{(k-2)}\|\right)}.$$

Each starting point is associated with a run, considered successful when the algorithm satisfies a prescribed stopping criterion within a certain maximum number of iterations, depending on the method. These details will be described better in the next sections. Moreover, in the case of Modified Newton method, in addition to exact gradients and hessians, finite differences approximations are employed with different step sizes and types of increment.

The next sections describe the implemented methods and the test problems in detail, followed by a discussion of the experimental results, cost analysis, and final remarks.

1.1 Modified Newton Method

The Modified Newton method is a variant of the classical Newton method. At each iteration it computes a search direction $p^{(k)}$ by solving the linear system

$$H_{mod}^{(k)} p^{(k)} = -\nabla f(x^{(k)}),$$

involving a modified version of the hessian matrix, computed as follow:

$$H_{mod}^{(k)} = H^{(k)} + E^{(k)}.$$

This modification is to ensure the positive definiteness of the hessian, so it is not necessarily computed. The update rule is

$$x^{(k+1)} = x^{(k)} + \alpha^{(k)} p^{(k)},$$

where $\alpha^{(k)}$ is determined in the code by a backtracking line search, governed by the Armijo condition

$$f(x^{(k)} + \alpha^{(k)} p^{(k)}) \leq f(x^{(k)}) + c\alpha^{(k)} \nabla f(x^{(k)})^\top p^{(k)}.$$

The algorithm attempts a full step $\alpha^{(k)} = 1$ first, and then reduces it geometrically until the condition is met or a maximum number of trials is reached.

In our implemetation:

1. $E^{(k)} = \tau I$. A modified Cholesky factorization is applied (Algorithm 6.3), that attempts to factor $H^{(k)} \approx LL^\top$ and adds the regularization shift τI, if needed. The procedure starts with $\tau = 0$, and doubles it iteratively until the factorization succeeds, ensuring positive definiteness of the hessian at each step.

2. $\rho = 0.5$ and $c = 10^{-4}$ are default parameters in the Armijo condition.

3. max_backtracking_iter = 10 is the maximum number of trials in the backtracking procedure. This value was chosen to prevent the stepsize alpha from becoming too small, and thus not adding more information than the current approximation of x.

Additionally, when testing problems with structure (such as banded functions), some components of the gradient and direction vectors are padded to preserve compatibility with the structure of the objective function and to avoid dimension mismatch. Finally, the following stopping criteria were considered:

- $\|\nabla f(x^{(k+1)})\|_\infty \leq \texttt{tol}$: the norm of the gradient computed in the new approximation is sufficiently small (below a required tolerance). If this criterion is met, the algorithm has reached a stationary point for the function to be minimized;
- $|f(x^{(k+1)}) - f(x^{(k)})| \leq \texttt{tol} \cdot \max(1, |f(x^{(k)})|)$: the relative functional decrement is sufficiently small (below a required tolerance). Such a criterion is introduced to prevent the algorithm from proceeding in an unnecessarily costly number of iterations, which do not significantly improve the function (as in the case of the Banded Trigonometric problem);
- $f(x^{(k+1)}) \leq \texttt{tol}$: the value of the function computed in the new approximation is sufficiently small (below a required tolerance). This stopping criterion is introduced in the case of functions whose minimum value is 0 (as in the case of Extended Rosenbrock and Generalized Broyden problems).

The algorithm breaks before reaching the maximum number of iterates, as soon as one of the previous conditions –checked in that order– is met. In the study of all benchmark problems, when Modified Newton method is applied, each individual execution, associated with a specific initial point, was considered a success if the algorithm breaks since the gradient is close to zero, that is, if the algorithm reaches a stationary point of the function.

1.1.1 Finite differences approximations

Within the elaborate, where possible, the finite differences method was implemented for the computation of first–and second–order derivatives. In the code, this variant is found in the Modified Newton method, which requires calculation of both the gradient and the hessian of the function to be minimized. Recalling that Nelder-Mead is a "zero-order" method, this part does not involve the use of finite differences.

The gradient approximation is obtained using the centered finite differences formula. In fact, although this is more expensive than forward and backward formulas, it offers a more accurate approximation of the required value. Considering $f : \mathbb{R}^n \to \mathbb{R}$ and $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$, with

$$\nabla f(x) = \left(\frac{\partial f}{\partial x_1}(x), \ldots, \frac{\partial f}{\partial x_n}(x) \right),$$

using centered finite differences for the approximated gradient, we obtain

$$\frac{\partial f}{\partial x_i}(x) \approx \frac{f(x + he_i) - f(x - he_i)}{2h}, \quad i = 1, \ldots, n. \tag{1}$$

Instead, in the hessian matrix $\nabla^2 f \in \mathbb{R}^{n,n}$

$$\nabla^2 f(x) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2}(x) & \ldots & \frac{\partial^2 f}{\partial x_1 \partial x_n}(x) \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1}(x) & \ldots & \frac{\partial^2 f}{\partial x_n^2}(x) \end{pmatrix},$$

each entry is obtained using the following calculus:

$$\frac{\partial^2 f}{\partial x_i \partial x_j}(x) \approx \frac{f(x + he_i + he_j) - f(x + he_i) - f(x + he_j) + f(x)}{h^2}. \tag{2}$$

The code implement the finite differences approximations, passing via handles the extra parameters `h` and `type`. In particular

- `h`: is a parameter equal to 10^{-k}, for $k = 2, 4, 6, 8, 10, 12$;
- `type`: is a parameter which indicates the typology of the increment. If type = 1 a variant of the increment scaled componentwise as $h_i = 10^{-k} \cdot |x_i|$ is used (this ensures reasonable accuracy and robustness when computing derivatives in high dimensions), otherwise default increment with $h_i = 10^{-k}$ is employed.

Since application of the Modified Newton method to large benchmak problems ($n = 10^3, 10^4, 10^5$) is required in the assignment, it was not possible to implement the formulas as presented, but it was necessary to make extensive use of each test function's structure and sparsity of each hessian matrix. Alternatively, the time to run the algorithms would have been excessive. In each of the sections regarding individual benchmark problems, it is detailed how these features were exploited in order to obtain reasonable results.

1.2 Nelder–Mead Method

The Nelder–Mead algorithm is a popular derivative–free optimization method designed to minimize a real-valued function $f : \mathbb{R}^n \to \mathbb{R}$ without relying on gradient or hessian informations. The algorithm operates by iteratively updating a simplex—a geometric figure composed of $n+1$ vertices in $\mathbb{R}^n$—based on function evaluations at its vertices.

At initialization, the algorithm constructs the simplex by perturbing the initial guess $x^{(0)}$ along the canonical directions with a small fixed step size. At each iteration k, the vertices are sorted according to their function values, in such a way that

$$f(x_1^{(k)}) \leq f(x_2^{(k)}) \leq \cdots \leq f(x_{n+1}^{(k)}).$$

Then the algorithm computes the centroid $\bar{x}$ of the best n vertices (excluding the worst one $x_{n+1}^{(k)}$), and applies the following operations:

- **Reflection:** the worst vertex is reflected through the centroid to produce a new trial point $x_R^{(k)}$. If $f(x_1^{(k)}) \leq f(x_R^{(k)}) < f(x_n^{(k)})$, then $x_R^{(k)}$ is accepted in the new simplex in place of $x_{n+1}^{(k)}$.

- **Expansion:** if $f(x_R^{(k)}) < f(x_1^{(k)})$, the reflected point significantly improves the function, so an expansion is attempted further along the reflection direction producing the point $x_E^{(k)}$. If $f(x_E^{(k)}) < f(x_R^{(k)})$, then $\tilde{x}_{n+1}^{(k+1)} = x_E^{(k)}$, otherwise $\tilde{x}_{n+1}^{(k+1)} = x_R^{(k)}$.

- **Contraction:** if the reflection fails to improve, the algorithm attempts a contraction between the centroid and the worst vertex, producing the point $x_C^{(k)}$.

- **Shrinkage:** if the contraction step is successful, then $\tilde{x}_{n+1}^{(k+1)} = x_C^{(k)}$, otherwise the entire simplex is shriked around the best vertex.

In our implementation default coefficients which are standard in the literature are used: $\rho = 1$ (reflection parameter), $\chi = 2$ (expansion parameter), $\gamma = 0.5$ (contraction parameter), and $\sigma = 0.5$ (shrinkage parameter).

Convergence is declared when one of the following stopping conditions is met:

- $|f(x_{\max}) - f(x_{\min})| \leq \texttt{tol}$: the maximum difference in function values across the simplex is sufficiently small (below a required tolerance);

- $\max_{i=2,\ldots,n+1} \|x^{(i)} - x^{(1)}\|_\infty \leq$ `tol`: the maximum distance between simplex points is sufficiently small (below a required tolerance).

2 Rosenbrock Function in Dimension 2

2.1 Problem introduction

The Rosenbrock function is a well-known test case for unconstrained optimization. Its 2-dimensional version is defined as

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2,$$

and features a curved valley with a global minimum $f(x^\star) = 0$ at $x^\star = (1, 1)$. Due to the narrow and curved shape of the valley, the problem is nontrivial for many first-order methods.

We tested both the Modified Newton and Nelder–Mead algorithms using the two standard initial guesses required by the assignment:

$$x_A^{(0)} = (1.2,\ 1.2), \qquad x_B^{(0)} = (-1.2,\ 1.0).$$

3D Visualization of the function:

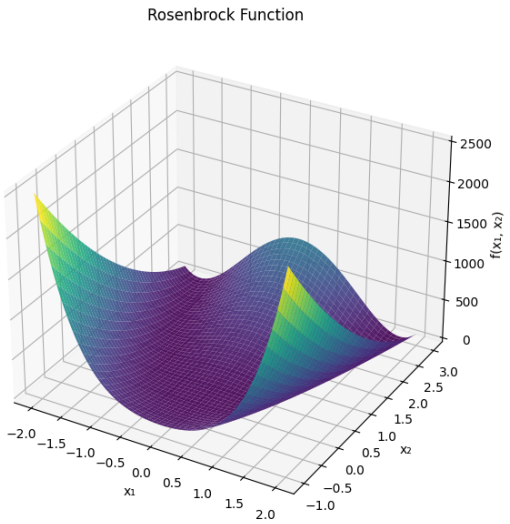


Figure 1: Surface plot of the Rosenbrock function over the domain $[-2, 2] \times [-1, 3]$. The function exhibits a narrow curved valley with a global minimum at $(1, 1)$, where $f(x) = 0$. This geometry makes it a standard benchmark for testing unconstrained optimization algorithms.

2.2 Experimental results

This subsection discusses the results obtained for the minimization of the 2-dimensional Rosenbrock function, comparing the application of the Modified Newton method and the Nelder–Mead method.

- **Modified Newton Method**:

Starting point	Function minimum	Function minimizer	Number of iterations
$x_A^{(0)}$	0.000000	$(1.000050, 1.000083)$	6
$x_B^{(0)}$	0.000000	$(0.999995, 0.999990)$	21

Table 1: The table highlights the convergence results associated with the application of the Modified Newton method. The analysis takes into account the starting point, the value of the minimum and the value of the minimizer reached, and the number of iterations that were required to achieve these results.

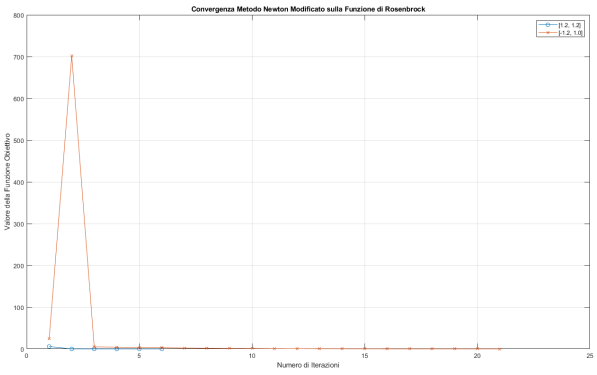


Figure 2: Convergence behaviour of the Modified Newton method on the Rosenbrock function starting from $x_A^{(0)} = [1.2, 1.2]$ and $x_B^{(0)} = [-1.2, 1.0]$.

- **Nelder–Mead Method**:

Starting point	Function minimum	Function minimizer	Number of iterations
$x_A^{(0)}$	0.000000	$((0.999741, 0.999441)$	58
$x_B^{(0)}$	0.053018	$((1.222612, 1.488895)$	29

Table 2: The table highlights the convergence results associated with the application of the Nelder–Mead method. The analysis takes into account the starting point, the value of the minimum and the value of the minimizer reached, and the number of iterations that were required to achieve these results.

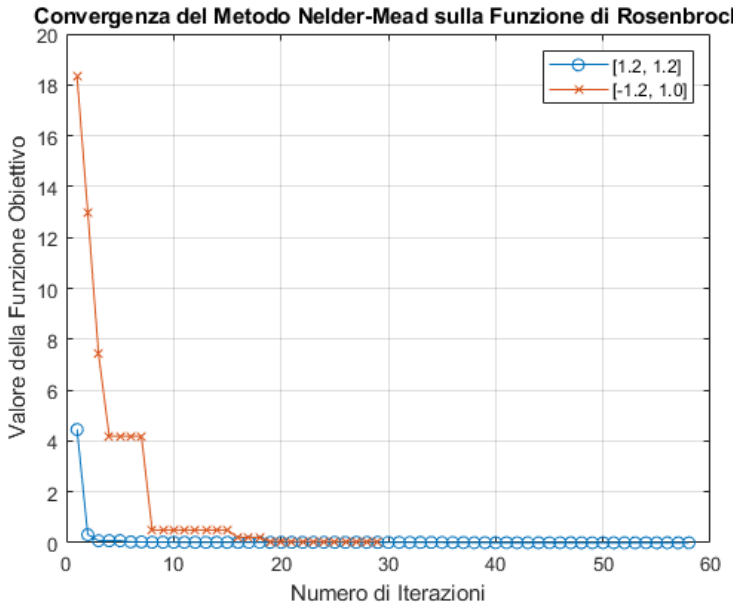


Figure 3: Convergence behaviour of the Nelder–Mead method on the Rosenbrock function starting from $x_A^{(0)} = [1.2, 1.2]$ and $x_B^{(0)} = [-1.2, 1.0]$.

Although both methods converge starting from $x_A^{(0)}$, the Modified Newton method reaches the solution significantly faster (6 iterations vs. 58). Starting from the more challenging initial point $x_B^{(0)}$, the Modified Newton method converges reliably, while Nelder–Mead gets stuck in a suboptimal region, with higher final function value and fewer iterations. This fact confirms the advantage of second–order informations for curved valleys.

3 Extended Rosenbrock Function

3.1 Problem introduction

The Extended Rosenbrock function is an high-dimensional generalization of the classical Rosenbrock function. For even dimensions n, it is defined as follow:

$$F(x) = \frac{1}{2}\sum_{k=1}^{n} f_k^2(x), \qquad f_k(x) = \begin{cases} 10(x_k^2 - x_{k+1}), & \text{if } k \bmod 2 = 1 \\ x_{k-1} - 1, & \text{if } k \bmod 2 = 0 \end{cases}.$$

This function is non–convex and features a narrow curved valley that makes optimization challenging in high dimensions. Its global minimum is $f(x^\star) = 0$ and its minimizer $x^\star$ such that $x_k^\star = 1$, $\forall i = 1, \ldots, n$. It is frequently used as a benchmark problem for large–scale unconstrained optimization algorithms, due to its scalability and pathological curvature.

In this chapter we are going to analyze the behaviour of both the Modified Newton and the Nelder–Mead methods, when they are applied to minimize the Extended Rosenbrock function. The initial points suggested in the benchmark library are $\bar{x} \in \mathbb{R}^n$ such that

$$\bar{x}_k = \begin{cases} -1.2 & \text{if} \quad k \bmod 2 = 1 \\ 1.0 & \text{if} \quad k \bmod 2 = 0 \end{cases}, \tag{3}$$

together with another 10 random initial points sampled uniformly in the hypercube $[\bar{x}_1 - 1, \bar{x}_1 + 1] \times \cdots \times [\bar{x}_n - 1, \bar{x}_n + 1] \subset \mathbb{R}^n$, starting from $\bar{x}$.

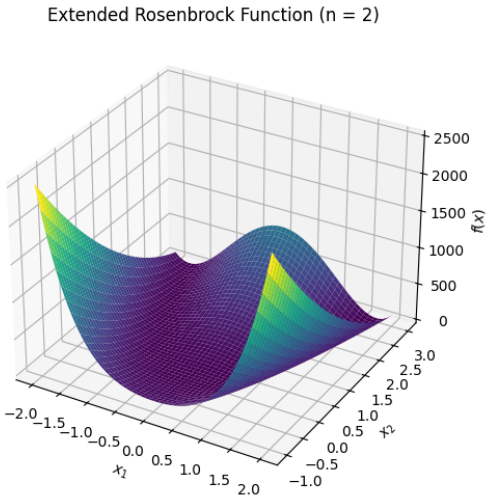


Figure 4: 3D visualization of the Extended Rosenbrock function in dimension $n = 2$. The global minimum lies at $(1, 1)$, and the function exhibits a curved valley that becomes increasingly difficult to navigate in higher dimensions.

3.2 Modified Newton method

In this subsection are shown the results obtained from the minimization of the Extended Rosenbrock function using the Modified Newton method. The study includes both exact derivatives and derivatives computed by finite–differences approximations. Due to its tridiagonal structure, the hessian is stored and manipulated in sparse format; this choice drastically reduces memory requirements and computational cost (in both matrix factorization and Newton direction computation), allowing efficiently handle large–scale problems. Before presenting detailed outcomes, a general experimental setup is given:

- `n` $= 10^3, 10^4, 10^5$;
- `max_iter` $= 5000$;
- `tol` $= 10^{-6}$.

For each run, the following were tracked:

- number of iterations to convergence;
- CPU time;

- number of successful runs;
- experimental rate of convergence ρ:

$$\rho \approx \frac{\log\left(\|x^{(k+1)} - x^{(k)}\| \big/ \|x^{(k)} - x^{(k-1)}\|\right)}{\log\left(\|x^{(k)} - x^{(k-1)}\| \big/ \|x^{(k-1)} - x^{(k-2)}\|\right)}.$$

In correspondence with the randomly generated points, an average behavior of each of the previous catogories is reported. Furthermore, the code is designed to work also with finite–differences case, and this variant is implemented thanks to the extra parameters

- `h`: a parameter equal to 10^{-k}, for $k = 2, 4, 6, 8, 10, 12$;
- `type`: a parameter which indicates if the increment is scaled componentwise as $h_i = 10^{-k} \cdot |x_i|$ or if it is a default increment such that $h_i = 10^{-k}$.

For this part, the several features were computed for each stepsize h.

3.2.1 Modified Newton method with exact derivatives

The Extended Rosenbrock function admits analytical expressions for both gradient and hessian in component–wise form. These are derived by exploiting the structure of the function and are used directly in the implementation for performance and accuracy. Specifically:

- each gradient's component is computed as:

$$\frac{\partial F}{\partial x_k}(x) = \begin{cases} 200(x_k^3 - x_k x_{k+1}) + (x_k - 1), & \text{if } k \bmod 2 = 1 \\ -100(x_{k-1}^2 - x_k), & \text{if } k \bmod 2 = 0 \end{cases};$$

- each entry of the hessian is given by:

$$\frac{\partial^2 F}{\partial x_k \partial x_j}(x) = \begin{cases} 200(3x_k^2 - x_{k+1}) + 1, & \text{if } j = k,\ k \bmod 2 = 1 \\ 100, & \text{if } j = k,\ k \bmod 2 = 0 \\ -200x_k, & \text{if } |k - j| = 1,\ k \bmod 2 = 1 \\ 0 & \text{otherwise} \end{cases}.$$

Dimension	Starting point	Iter	Time (s)	ρ	Success
10^3	$\bar{x}$	20	0.00	2.14	1/1
10^3	Avg (10 pts)	25.3	0.01	2.02	10/10
10^4	$\bar{x}$	20	0.02	2.14	1/1
10^4	Avg (10 pts)	25.4	0.05	1.96	10/10
10^5	$\bar{x}$	20	0.34	2.14	1/1
10^5	Avg (10 pts)	26.0	0.74	1.37	10/10

Table 3: The table highlights the convergence results associated with the application of the Modified Newton method with exact derivatives. The analysis takes into account the starting point (for the 10 random points an average behaviour is reported), the number of iterations that were required to achieve the result, the CPU time (s) and the rate of convergence ρ.

Experimental results. All runs shown in table (3) were successful according to the stopping criterion related to the gradient ($\|\nabla f(x^{(k)})\|_\infty \leq \texttt{tol}$), and reached the known minimum $f^\star = 0$ up to machine precision. The number of iterations remains nearly constant across dimensions both for $\bar{x}$ and for the 10 random starting points, highlighting the scalability of Newton's method when combined with sparse matrix storage. CPU time increases with n, as expected due to the cost of Cholesky factorization on sparse tridiagonal matrices. The experimental convergence rate ρ remains close to quadratic ($\rho \approx 2$) for moderate dimensions, while it slightly drops for $n = 10^5$, possibly due to cumulative rounding errors.

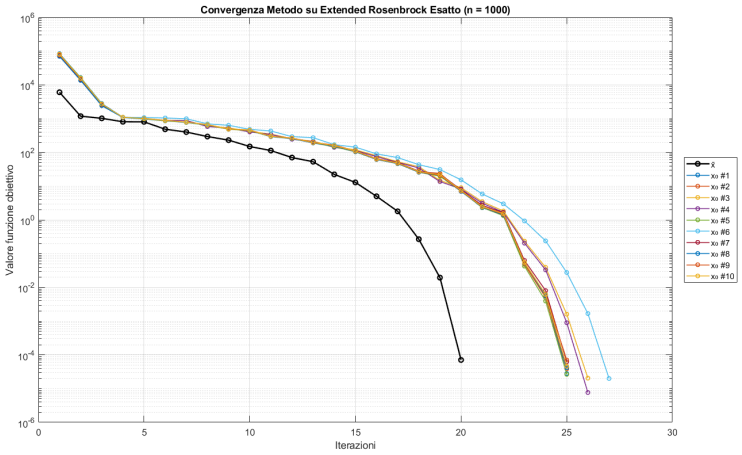


Figure 5: Convergence of the Modified Newton method with exact derivatives on the Extended Rosenbrock function for $n = 1000$. Each curve corresponds to a different initial point. The method converges quadratically with stable behaviour across all tests.

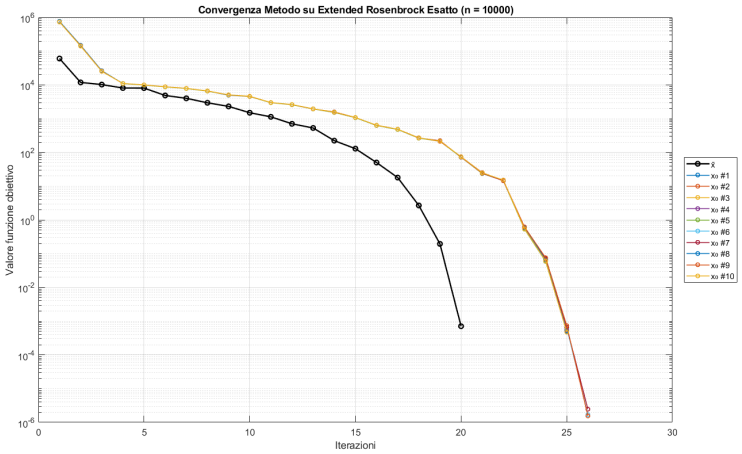


Figure 6: Convergence of the Modified Newton method on the Extended Rosenbrock function for $n = 10\,000$. The method exhibits consistent quadratic convergence also in higher dimension.

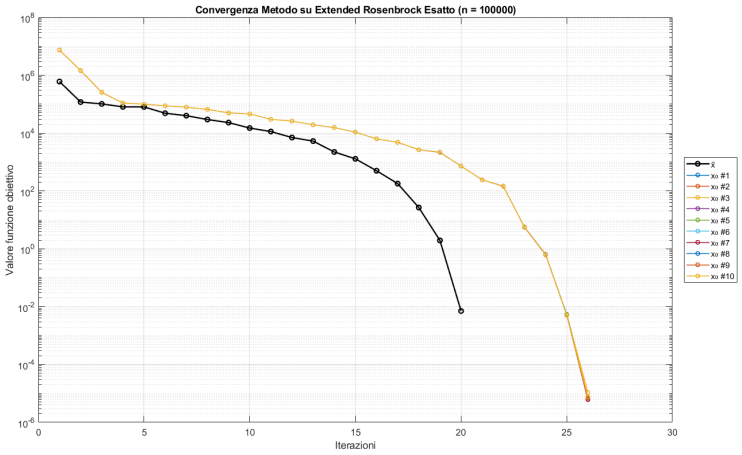


Figure 7: Convergence of the Modified Newton method on the Extended Rosenbrock function for $n = 100\,000$. Despite the high dimensionality, the convergence remains stable with similar iteration counts.

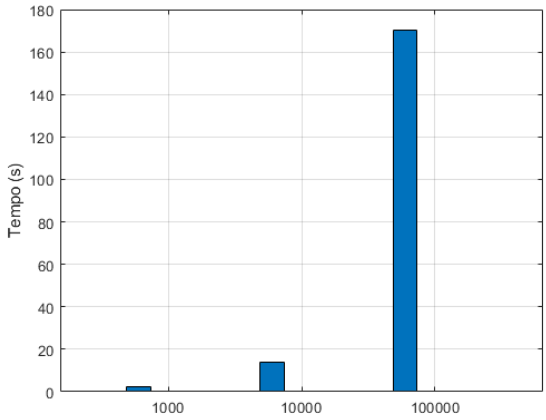


Figure 8: Execution time (in seconds) of the Modified Newton method with exact derivatives for $n = 10^3$, 10^4 and 10^5. The runtime grows approximately linearly with the problem size, confirming the efficiency of sparse matrix operations.

3.2.2 Modified Newton method with approximated derivatives

When exact gradients and hessians are not available, a common alternative is to approximate them using finite differences method. In our implementation,

we manually built the approximations in component-wise form, exploiting the alternating structure of the Extended Rosenbrock function.

Gradient approximation. As we have seen in the introduction part, the gradient vector is computed using the centered finite differences formula (1). Due to the structure of the function, only a small number of f_k survive when computing the numerator in the formula. In particular, fixed k as the component whose derivative is being approximated:

$$\frac{\partial F}{\partial x_k}(x) \approx \begin{cases} \frac{\frac{1}{2}[f_k^2(x+he_k)+f_{k+1}^2(x+he_k)]-\frac{1}{2}[f_k^2(x-he_k)+f_{k+1}^2(x-he_k)]}{2h} & \text{if } k \bmod 2 = 1 \\ \frac{\frac{1}{2}f_{k-1}^2(x+he_k)-\frac{1}{2}f_{k-1}^2(x-he_k)}{2h} & \text{if } k \bmod 2 = 0 \end{cases}. \tag{4}$$

Thus, the finite-difference approximation of the gradient was implemented as:

$$\frac{\partial F}{\partial x_k}(x) \approx \begin{cases} 600x_k^2 - 100x_k x_{k+1} + \frac{1}{2}h + 350x_k^3 + 300hx_k^2, & \text{if } k \bmod 2 = 1 \\ -100x_{k-1}^2 + 100x_k, & \text{if } k \bmod 2 = 0 \end{cases}.$$

Hessian approximation. First, as seen in the previous section, we recall that the hessian matrix of the Extended Rosenbrock function has a tridiagonal structure. Then, appling the same proceeding as before, we can observe that function evaluations needed to compute diagonal entries of $\nabla^2 F(x)$, differ only by the terms

$$\begin{cases} \frac{1}{2}[f_k^2(x) + f_{k+1}^2(x)] & \text{if} \quad k \bmod 2 = 1 \\ \frac{1}{2}f_{k-1}^2(x) & \text{if} \quad k \bmod 2 = 0 \end{cases},$$

while function evaluations needed to compute non-diagonal entries differ only by the terms

$$\begin{cases} \frac{1}{2}[f_k^2(x) + f_{k+1}^2(x)] & \text{if} \quad k \bmod 2 = 1 \\ 0 & \text{if} \quad k \bmod 2 = 0 \end{cases}.$$

So, the approximated component-wise second derivatives can be coded as:

$$\frac{\partial^2 F}{\partial x_k \partial x_j}(x) \approx \begin{cases} 1200hx_k - 200x_{k+1} + 700h^2 + 600x_k^2, & \text{if } j = k,\ k \bmod 2 = 1 \\ 100, & \text{if } j = k,\ k \bmod 2 = 0 \\ -100hx_k - 200x_k, & \text{if } |k-j| = 1,\ k \bmod 2 = 1 \\ 0, & \text{otherwise} \end{cases}$$

Experimental results. In finite differences–case, two figures were generated per dimension: one for fixed and one for scaled increment. As others, each plot contains:

- a black curve for the reference point $\bar{x}$;
- ten colored curves, one for each random starting point;
- log–scaled $f(x_k)$ values against iterations.

In general, the algorithm lead to convergence within reasonable iteration counts and CPU time, despite the considerable increase in size; scaled increments tend to be slightly more robust and stable when compared with default increments, for all dimensions. In case of $n = 1\,000$ all increments succeed consistently with almost full accuracy, but this does not happen as the size of the problem increases. In fact, both for $n = 10\,000$ and $n = 100\,000$, the algorithm fails to converge in all trials, in correspondence with the largest fixed and the largest scaled increments referred to $h = 10^{-2}$. Moreover, for $n = 1\,000$, quadratic convergence is observed for $h \leq 10^{-6}$ (with $\rho \approx 2$) both for $\bar{x}$ and the other 10 random points, while it slightly drops ($\rho \approx 1$) starting from the 10 random points in large dimensions. The following tables contain more detailed outcomes.

- $\boldsymbol{n = 1\,000}$

Increment	**Init.**	**Iter**	**Time (s)**	ρ	**Successes**
10^{-2}	$\bar{x}$	118	0.03	0.8856	1/1
10^{-2}	Avg (10 pts)	122.5	0.038	0.8856	10/10
$10^{-2} \cdot \lvert x \rvert$	$\bar{x}$	110	0.03	0.8794	1/1
$10^{-2} \cdot \lvert x \rvert$	Avg (10 pts)	113.6	0.03	0.8794	10/10
10^{-4}	$\bar{x}$	21	0.01	1.1640	1/1
10^{-4}	Avg (10 pts)	26	0.01	1.3972	10/10
$10^{-4} \cdot \lvert x \rvert$	$\bar{x}$	21	0.01	1.1476	1/1
$10^{-4} \cdot \lvert x \rvert$	Avg (10 pts)	26	0.01	1.4166	10/10
10^{-6}	$\bar{x}$	20	0.01	2.4144	1/1
10^{-6}	Avg (10 pts)	25.2	0.01	2.1933	10/10
$10^{-6} \cdot \lvert x \rvert$	$\bar{x}$	20	0.01	2.4464	1/1
$10^{-6} \cdot \lvert x \rvert$	Avg (10 pts)	25.2	0.01	2.2105	10/10
10^{-8}	$\bar{x}$	20	0.01	2.1463	1/1
10^{-8}	Avg (10 pts)	25.2	0.01	2.2284	10/10
$10^{-8} \cdot \lvert x \rvert$	$\bar{x}$	20	0.01	2.1468	1/1
$10^{-8} \cdot \lvert x \rvert$	Avg (10 pts)	25.2	0.01	2.2284	10/10
10^{-10}	$\bar{x}$	20	0.01	2.1432	1/1
10^{-10}	Avg (10 pts)	25.2	0.01	2.2278	10/10
$10^{-10} \cdot \lvert x \rvert$	$\bar{x}$	20	0.01	2.1432	1/1
$10^{-10} \cdot \lvert x \rvert$	Avg (10 pts)	25.2	0.01	2.2278	10/10
10^{-12}	$\bar{x}$	20	0.01	2.1432	1/1
10^{-12}	Avg (10 pts)	25.2	0.01	2.2278	10/10
$10^{-12} \cdot \lvert x \rvert$	$\bar{x}$	20	0.01	2.1432	1/1
$10^{-12} \cdot \lvert x \rvert$	Avg (10 pts)	25.2	0.01	2.2278	10/10

Table 4: Finite difference results using different typology of increments ($n = 10^3$).

Figure 9: Convergence of Modified Newton method on Extended Rosenbrock function in dimension $n = 10^3$ with fixed increment $h = 10^{-2}$ (left) and scaled increment $h = 10^{-2} \cdot |x|$ (right).

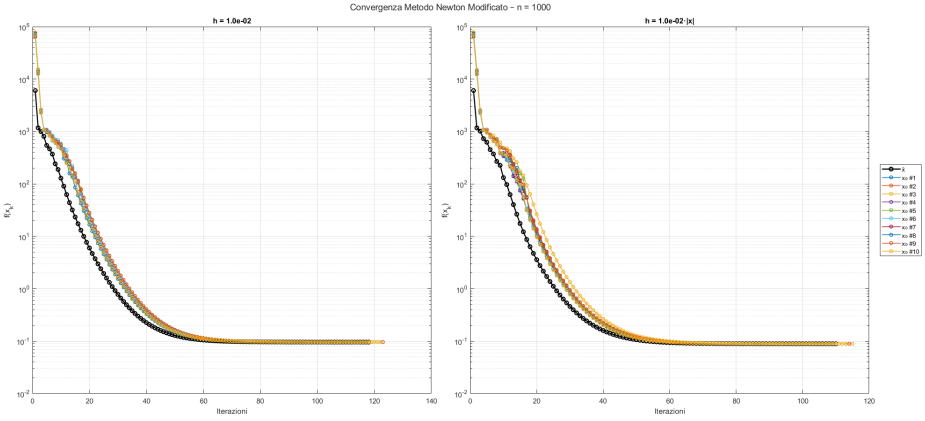
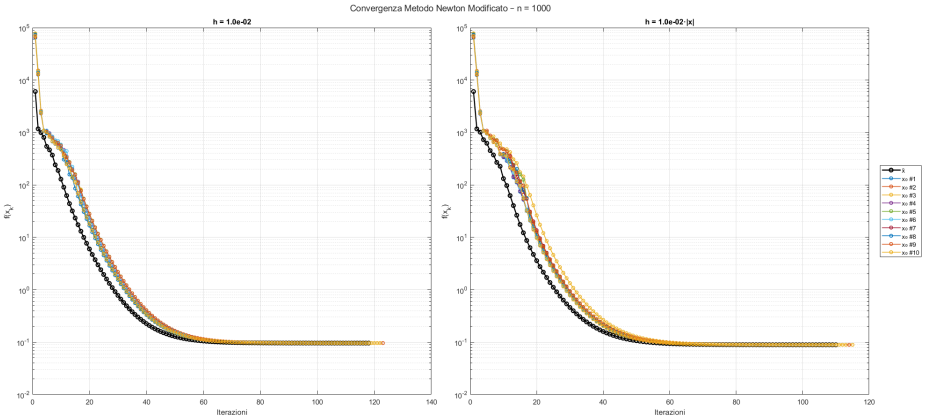


Figure 10: Convergence of Modified Newton method on Extended Rosenbrock function in dimension $n = 10^3$ with fixed increment $h = 10^{-12}$ (left) and scaled increment $h = 10^{-12} \cdot |x|$ (right).



- $\boldsymbol{n = 10\,000}$

Increment	**Init.**	**Iter**	**Time (s)**	ρ	**Successes**
10^{-2}	$\bar{x}$	118	0.23	0.8856	0/10
10^{-2}	Avg (10 pts)	125.3	2.66	0.8856	0/10
$10^{-2} \cdot \lvert x \rvert$	$\bar{x}$	110	0.21	0.8794	0/10
$10^{-2} \cdot \lvert x \rvert$	Avg (10 pts)	114.3	2.44	0.8794	0/10
10^{-4}	$\bar{x}$	21	0.04	1.1640	10/10
10^{-4}	Avg (10 pts)	27.0	0.92	1.2708	10/10
$10^{-4} \cdot \lvert x \rvert$	$\bar{x}$	21	0.04	1.1476	10/10
$10^{-4} \cdot \lvert x \rvert$	Avg (10 pts)	27.0	0.91	1.2298	10/10
10^{-6}	$\bar{x}$	20	0.04	2.4144	10/10
10^{-6}	Avg (10 pts)	25.6	0.89	1.3572	10/10
$10^{-6} \cdot \lvert x \rvert$	$\bar{x}$	20	0.04	2.4464	10/10
$10^{-6} \cdot \lvert x \rvert$	Avg (10 pts)	25.6	0.89	1.3581	10/10
10^{-8}	$\bar{x}$	20	0.04	2.1463	10/10
10^{-8}	Avg (10 pts)	25.6	0.90	1.3626	10/10
$10^{-8} \cdot \lvert x \rvert$	$\bar{x}$	20	0.04	2.1468	10/10
$10^{-8} \cdot \lvert x \rvert$	Avg (10 pts)	25.6	0.90	1.3626	10/10
10^{-10}	$\bar{x}$	20	0.04	2.1432	10/10
10^{-10}	Avg (10 pts)	25.6	0.90	1.3626	10/10
$10^{-10} \cdot \lvert x \rvert$	$\bar{x}$	20	0.04	2.1432	10/10
$10^{-10} \cdot \lvert x \rvert$	Avg (10 pts)	25.6	0.90	1.3626	10/10
10^{-12}	$\bar{x}$	20	0.04	2.1432	10/10
10^{-12}	Avg (10 pts)	25.6	0.90	1.3626	10/10
$10^{-12} \cdot \lvert x \rvert$	$\bar{x}$	20	0.04	2.1432	10/10
$10^{-12} \cdot \lvert x \rvert$	Avg (10 pts)	25.6	0.90	1.3626	10/10

Table 5: Finite difference results for using different typology of increments ($n = 10^4$).

Figure 11: Convergence of Modified Newton method on Extended Rosenbrock function in dimension $n = 10^4$ with fixed increment $h = 10^{-2}$ (left) and scaled increment $h = 10^{-2} \cdot |x|$ (right).

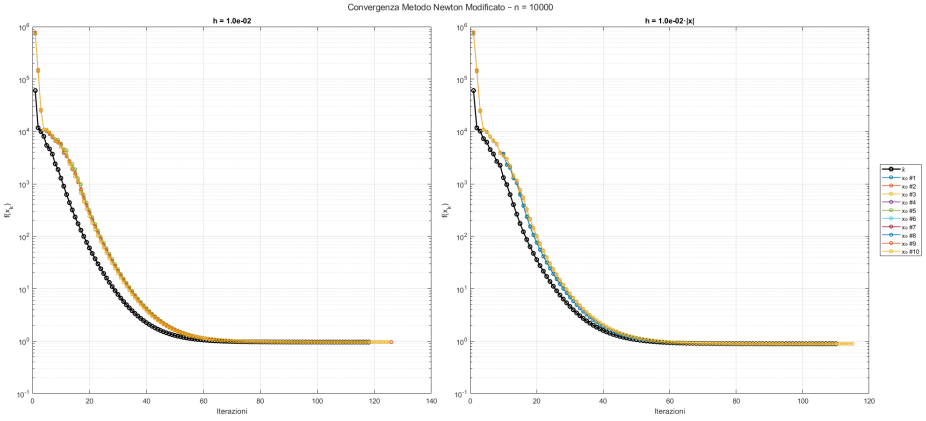
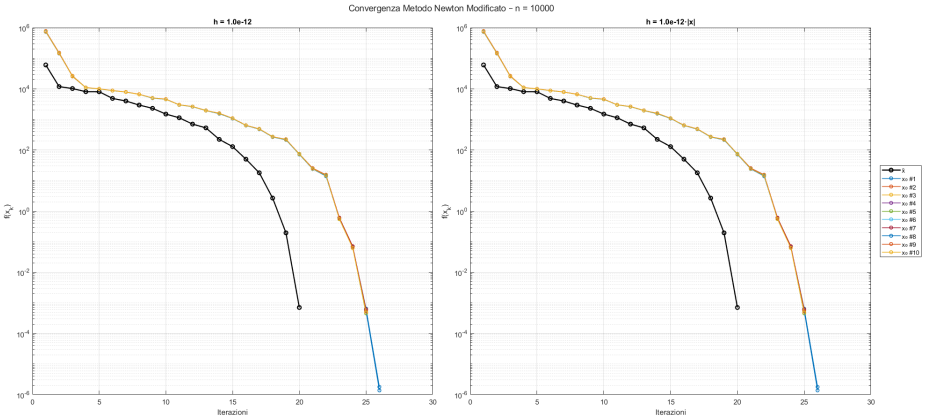


Figure 12: Convergence of Modified Newton method on Extended Rosenbrock function in dimension $n = 10^4$ with fixed increment $h = 10^{-12}$ (left) and scaled increment $h = 10^{-12} \cdot |x|$ (right).



- $\boldsymbol{n = 100\,000}$

Increment	**Init.**	**Iter**	**Time (s)**	ρ	**Successes**
10^{-2}	$\bar{x}$	118	2.66	0.8856	0/10
10^{-2}	Avg (10 pts)	126.0	3.26	0.8856	0/10
$10^{-2} \cdot \lvert x \rvert$	$\bar{x}$	110	2.41	0.8794	0/10
$10^{-2} \cdot \lvert x \rvert$	Avg (10 pts)	114.2	2.94	0.8794	0/10
10^{-4}	$\bar{x}$	22	0.49	0.9324	10/10
10^{-4}	Avg (10 pts)	27.0	0.92	1.2738	10/10
$10^{-4} \cdot \lvert x \rvert$	$\bar{x}$	22	0.49	0.9453	10/10
$10^{-4} \cdot \lvert x \rvert$	Avg (10 pts)	27.0	0.91	1.2309	10/10
10^{-6}	$\bar{x}$	20	0.44	2.4144	10/10
10^{-6}	Avg (10 pts)	26.0	0.89	1.3622	10/10
$10^{-6} \cdot \lvert x \rvert$	$\bar{x}$	20	0.44	2.4464	10/10
$10^{-6} \cdot \lvert x \rvert$	Avg (10 pts)	26.0	0.89	1.3620	10/10
10^{-8}	$\bar{x}$	20	0.44	2.1463	10/10
10^{-8}	Avg (10 pts)	26.0	0.90	1.3633	10/10
$10^{-8} \cdot \lvert x \rvert$	$\bar{x}$	20	0.44	2.1468	10/10
$10^{-8} \cdot \lvert x \rvert$	Avg (10 pts)	26.0	0.90	1.3633	10/10
10^{-10}	$\bar{x}$	20	0.45	2.1432	10/10
10^{-10}	Avg (10 pts)	26.0	0.90	1.3632	10/10
$10^{-10} \cdot \lvert x \rvert$	$\bar{x}$	20	0.44	2.1432	10/10
$10^{-10} \cdot \lvert x \rvert$	Avg (10 pts)	26.0	0.90	1.3632	10/10
10^{-12}	$\bar{x}$	20	0.44	2.1432	10/10
10^{-12}	Avg (10 pts)	26.0	0.89	1.3632	10/10
$10^{-12} \cdot \lvert x \rvert$	$\bar{x}$	20	0.44	2.1432	10/10
$10^{-12} \cdot \lvert x \rvert$	Avg (10 pts)	26.0	0.89	1.3632	10/10

Table 6: Finite difference results using different increments ($n = 10^5$).

Figure 13: Convergence of Modified Newton method on Extended Rosenbrock function ($n = 10^5$) with fixed increment $h = 10^{-2}$ (left) and scaled increment $h = 10^{-2} \cdot |x|$ (right).

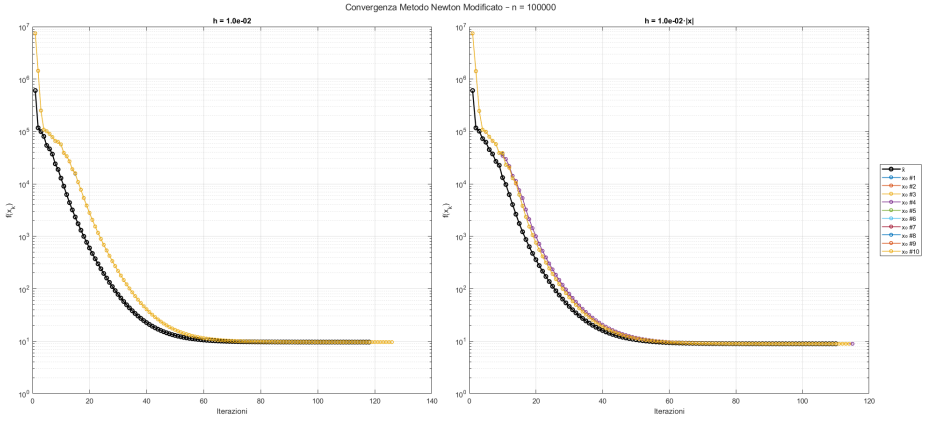
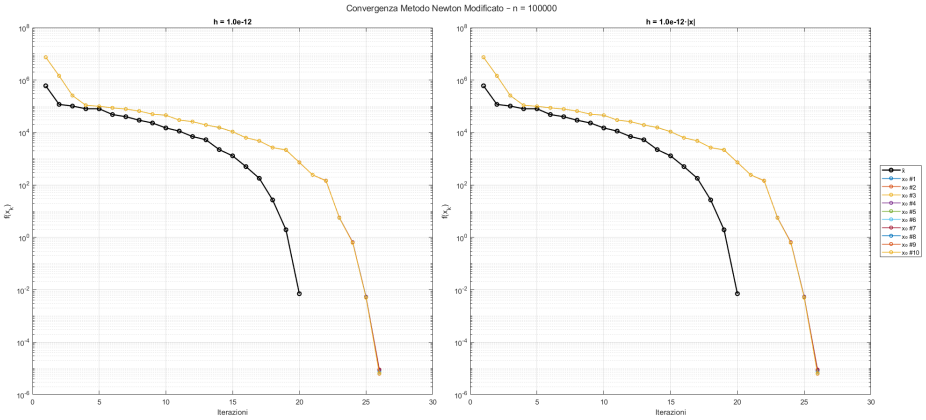


Figure 14: Convergence of Modified Newton method on Extended Rosenbrock function ($n = 10^5$) with fixed increment $h = 10^{-12}$ (left) and scaled increment $h = 10^{-12} \cdot |x|$ (right).



3.3 Nelder–Mead method

In this subsection are shown the results obtained from the minimization of the Extended Rosenbrock function using the Nelder–Mead method. Before presenting the results, a general experimental setup is given:

- `n` $= 10, 26, 50$;
- `max_iter` $= 80.000$;
- `tol` $= 10^{-6}$.

Moreover, for each run were recorded:

- number of iterations to convergence;
- CPU time;
- final objective value found f_{min};
- experimental rate of convergence ρ:

$$\rho \approx \frac{\log\left(\|x^{(k+1)} - x^{(k)}\| \big/ \|x^{(k)} - x^{(k-1)}\|\right)}{\log\left(\|x^{(k)} - x^{(k-1)}\| \big/ \|x^{(k-1)} - x^{(k-2)}\|\right)}.$$

In correspondence with the randomly generated points, an average behavior of each of the previous catogories is reported.

Experimental results. The algorithm successfully runs on all dimensions, but convergence was never achieved within tolerance. The number of iterations and function values clearly indicate that Nelder–Mead is not effective on this problem; incidentally, in all cases the final function values remained far from zero, indicating that Nelder–Mead struggled with this function even in low dimensions. The high curvature and narrow valleys of the Extended Rosenbrock function severely impact the simplex–based exploration. Furthermore, while runtime remains reasonable, the algorithm not only requires thousands of iterations to make progress but also does not reach values close to zero. This confirms the known limitations of Nelder–Mead in high–dimensional landscapes.

Dimension	Starting point	$f_{\min}$	Iter	Time (s)	ρ
10	$\bar{x}$	4.755805	1807	0.04	0.9082
	Avg (10 pts)	8.248833	1221	0.016	0.0500
26	$\bar{x}$	21.632097	13592	0.20	-0.8190
	Avg (10 pts)	31.887709	9669	0.142	-6.6754
50	$\bar{x}$	36.483857	49703	1.15	-97.5929
	Avg (10 pts)	121.868407	42439	0.953	-1.3963

Table 7: Results of Nelder–Mead on Extended Rosenbrock function.

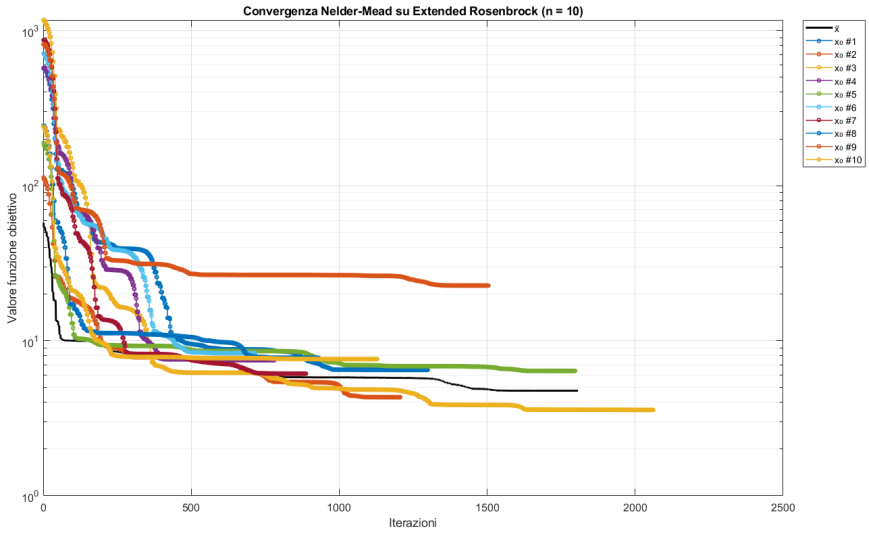


Figure 15: Convergence of Nelder–Mead on Extended Rosenbrock function ($n = 10$) from reference and 10 random initial points. The objective decreases but stagnates above the global minimum.

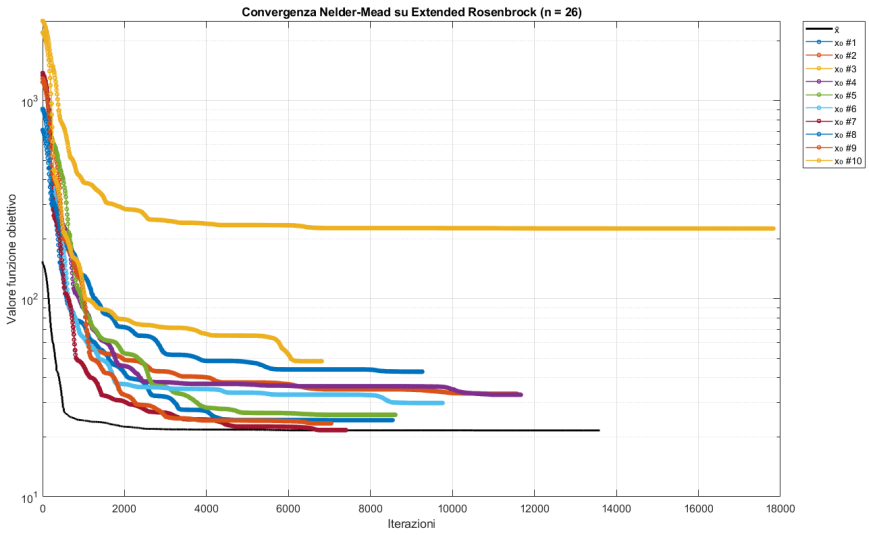


Figure 16: Convergence of Nelder–Mead on Extended Rosenbrock function ($n = 26$). Progress slows down and most trajectories fail to improve after early iterations.

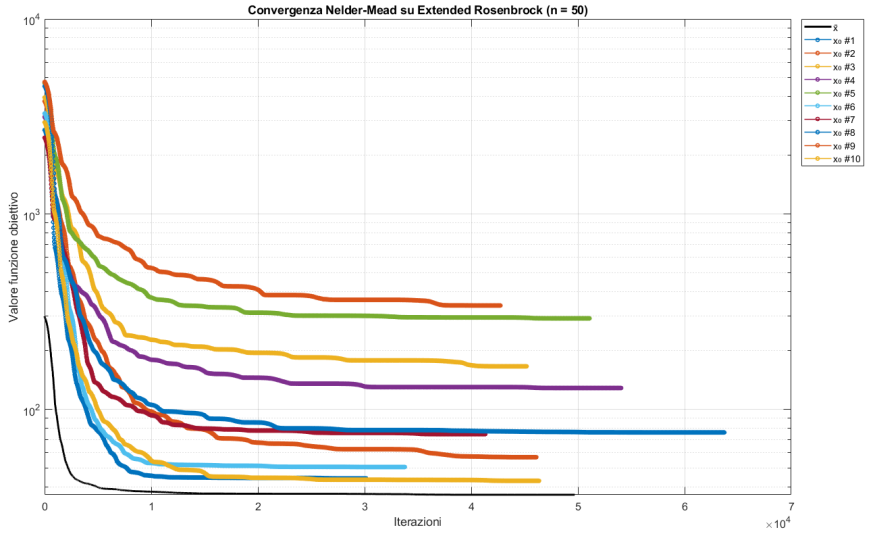


Figure 17: Convergence of Nelder–Mead on Extended Rosenbrock function ($n = 50$). None of the trials reach satisfactory objective values, confirming the method's limits in high dimensions.

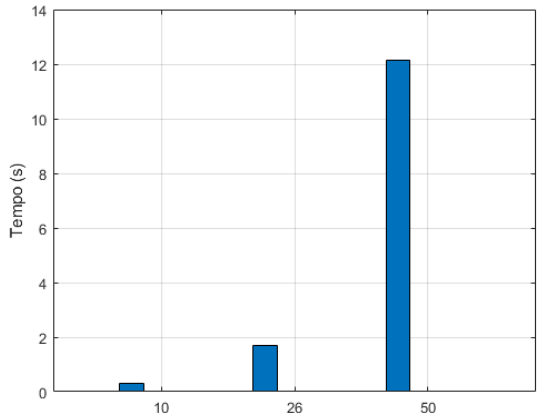


Figure 18: Execution time of Nelder–Mead on Extended Rosenbrock for increasing dimensions $n = 10$, 26, and 50. The cost grows sharply due to the increased simplex size.

4 Generalized Broyden Tridiagonal Function

4.1 Problem introduction

The Generalized Broyden Tridiagonal function is a classical benchmark used in unconstrained optimization, known for its nonlinear and tridiagonal structure. Let $x \in \mathbb{R}^n$,

$$F(x) = \frac{1}{2}\sum_{k=1}^{n} f_k^2(x) \qquad f_k(x) = (3 - 2x_k)x_k + 1 - x_{k-1} - x_{k+1}, \quad (5)$$

with boundary conditions $x_0 = x_{n+1} = 0$. $F(x)$ combines a quadratic component in x_i, two linear coupling terms with the neighbors x_{i-1} and x_{i+1}, and adds a constant shift. The global minimum of the function is $F(x^\star) = 0$ is achieved when all terms inside the squared expression are zero. Thus, the global minimizer $x^* \in \mathbb{R}^n$ is such that

$$x_i^* = 1, \quad \forall i = 1, \ldots, n.$$

In this chapter we are going to analyze the behaviour of both the Modified Newton and the Nelder–Mead methods, when they are applied to minimize the Generalized Broyden Tridigonal function. The initial points suggested in the benchmark library are $\bar{x} \in \mathbb{R}^n$ such that

$$\bar{x}_k = -1.0 \quad \text{for} \quad k = 1, 2, \ldots n,$$

together with another 10 random initial points sampled uniformly in the hypercube $[\bar{x}_1 - 1, \bar{x}_1 + 1] \times \cdots \times [\bar{x}_n - 1, \bar{x}_n + 1] \subset \mathbb{R}^n$, starting from $\bar{x}$.

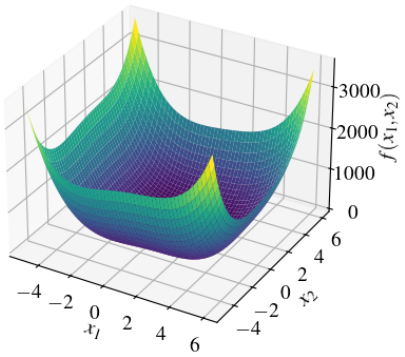


Figure 19: 3D visualization of the Generalized Broyden Tridiagonal function for $n = 2$.

4.2 Modified Newton method

In this subsection are shown the results obtained from the minimization of the Generalized Broyden Tridiagonal function using the Modified Newton method; the study includes both exact derivatives and derivatives computed by finite–differences approximations. Due to its pentadiagonal structure, the hessian is stored and manipulated in sparse format; this choice drastically reduces memory requirements and computational cost (in both matrix factorization and Newton direction computation), allowing efficiently handle large–scale problems. Before presenting detailed outcomes, a general experimental setup is given:

- `n` $= 10^3, 10^4, 10^5$;
- `max_iter` $= 5000$;
- `tol` $= 10^{-6}$.

For each run, the following were tracked:

- number of iterations to convergence;
- CPU time;
- number of successful runs;
- experimental rate of convergence ρ:

$$\rho \approx \frac{\log\left(\|x^{(k+1)} - x^{(k)}\| \Big/ \|x^{(k)} - x^{(k-1)}\|\right)}{\log\left(\|x^{(k)} - x^{(k-1)}\| \Big/ \|x^{(k-1)} - x^{(k-2)}\|\right)}.$$

In correspondence with the randomly generated points, an average behavior of each of the previous catogories is reported. Furthermore, the code is designed to work also with finite–differences case, and this variant is implemented thanks to the extra parameters

- `h`: a parameter equal to 10^{-k}, for $k = 2, 4, 6, 8, 10, 12$;
- `type`: a parameter which indicates if the increment is scaled componentwise as $h_i = 10^{-k} \cdot |x_i|$ or if it is a default increment such that $h_i = 10^{-k}$.

For this part, the several features were computed for each stepsize h.

4.2.1 Modified Newton method with exact derivatives

In this section we test the Modified Newton method on the Generalized Broyden Tridiagonal function using exact gradient and hessian. Both were derived analytically and implemented exploiting the banded pattern of the function. Specifically:

- each gradient's component is computed as:

$$\frac{\partial F}{\partial x_k}(x) = \begin{cases} (3-4x_1)f_1(x) - f_2(x), & k = 1 \\ (3-4x_k)f_k(x) - f_{k+1}(x) - f_{k-1}(x), & 1 < k < n \\ (3-4x_n)f_n(x) - f_{n-1}(x), & k = n \end{cases};$$

- each entry of the hessian is given by:

$$\frac{\partial^2 F}{\partial x_k \partial x_j}(x) = \begin{cases} (3-4x_1)^2 - 4f_1(x) + 1, & k = j = 1 \\ (3-4x_k)^2 - 4f_k(x) + 2, & 1 < k < n,\ k = j \\ (3-4x_n)^2 - 4f_n(x) + 1, & k = j = n \\ 4x_k + 4x_{k+1} - 6, & |k-j| = 1 \\ 1, & |k-j| = 2 \\ 0 & \text{otherwise} \end{cases}.$$

Dim	**Init.**	$f_{\min}$	**Iter**	**Time (s)**	ρ
10^3	$\bar{x}$	0.000000	5	0.02	1.93
	Avg (10 pts)	$< 10^{-6}$	8.1	0.02	1.63
10^4	$\bar{x}$	0.000000	5	0.17	1.93
	Avg (10 pts)	$< 10^{-6}$	8.0	0.17	1.64
10^5	$\bar{x}$	0.000000	5	2.01	1.93
	Avg (10 pts)	$< 10^{-6}$	8.0	2.08	1.62

Table 8: Results of Modified Newton method on Generalized Broyden function using exact derivatives.

Experimental results. As shown in table (4.2.1), the Modified Newton method with exact derivatives proves to be extremely effective on the Generalized Broyden tridiagonal function. In all tested dimensions, convergence is achieved in very few iterations (5 for the reference starting point, around 8

on average from random initializations). Function values reach machine precision accuracy, and the convergence rate ρ consistently approaches the ideal quadratic rate of 2. Importantly, the computational time remains reasonable even for large dimensions, thanks to the efficient sparse implementation of the hessian matrix. These results confirm that when second-order information is available and well–structured, Newton–type methods offer both precision and speed — outperforming derivative-free approaches by a large margin in terms of both accuracy and reliability.

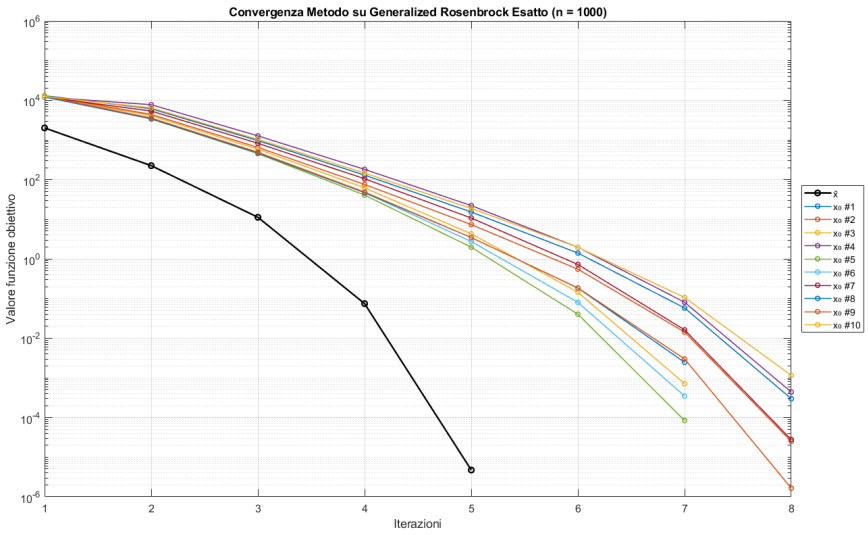


Figure 20: Convergence of the Modified Newton method on Generalized Broyden function with $n = 1000$ using exact gradient and Hessian. The method converges in few iterations with a consistent superlinear rate.

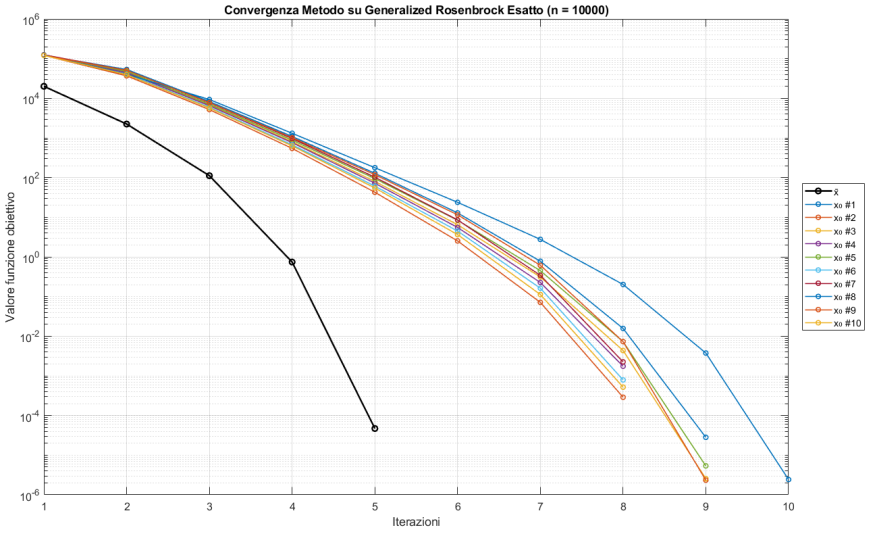


Figure 21: Convergence of the Modified Newton method on Generalized Broyden function with $n = 10000$. The convergence behavior remains stable across all random initializations.

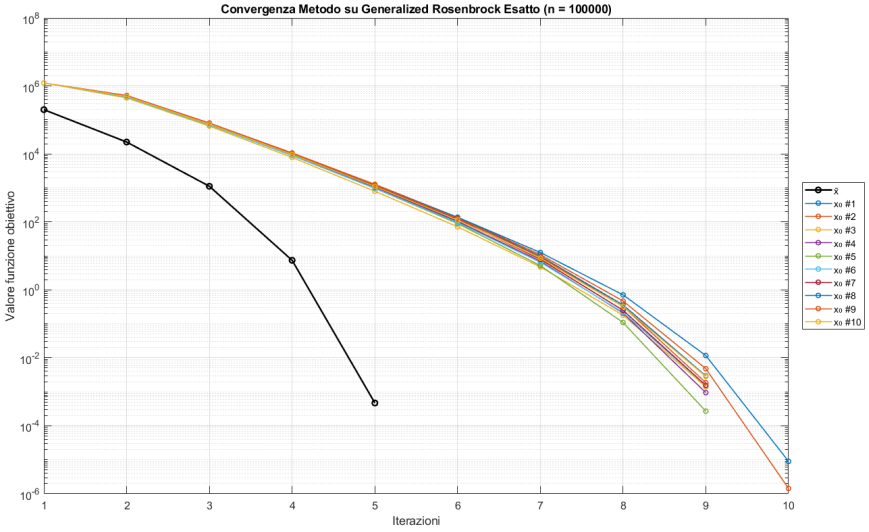


Figure 22: Convergence of the Modified Newton method on Generalized Broyden function with $n = 100000$. Even in high dimensions, the algorithm remains robust and fast.

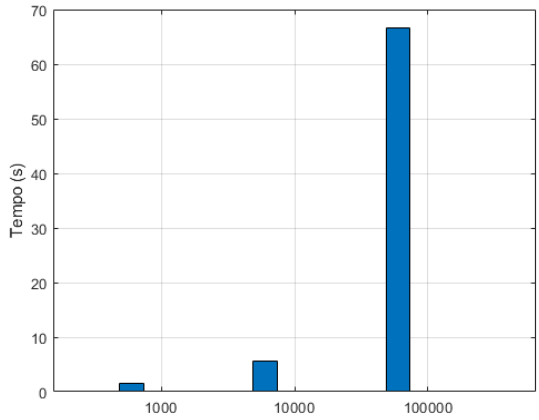


Figure 23: Execution time of the Modified Newton method with exact derivatives on Generalized Broyden function for increasing dimensions. The computation scales efficiently due to sparse Hessian structure.

4.2.2 Modified Newton method with approximated derivatives

In this experiment, we use the Modified Newton method on the Generalized Broyden Tridiagonal function with gradient and hessian computed via finite differences. This approach avoids symbolic derivation, at the cost of numerical approximations and additional function evaluations. The approximated gradient and Hessian components were expanded manually and implemented directly to avoid cancellation and redundancy. The sparsity pattern of the exact hessian was preserved to limit computational cost.

Gradient approximation. Due to the structure of the function, only a small number of f_k survive when computing the numerator in the formula (1). In particular, fixed k as the component whose derivative is being approximated, and defining:

$$g_k(x) = \frac{1}{2}[f_k^2(x) + f_{k+1}^2(x) + f_{k-1}^2(x)],$$

then

$$\frac{\partial F}{\partial x_k}(x) \approx \frac{g_k(x + he_k) - g_k(x - he_k)}{2h}. \tag{6}$$

It is not reported its explicit expression, as it is corpus.

Hessian approximation. Appling the same proceeding as before, we can observe that function evaluations needed to compute diagonal entries of $\nabla^2 F(x)$, differ only by the terms

$$\frac{1}{2}[f_k^2(x) + f_{k+1}^2(x) + f_{k-1}^2(x)].$$

Function evaluations needed to compute non-diagonal entries, on the first codiagonal, differ only by the terms

$$\frac{1}{2}[f_k^2(x) + f_{k+1}^2(x)],$$

while function evaluations needed to compute the ones on the second codiagonal, differ by the terms

$$\frac{1}{2}f_{k-1}^2(x).$$

Specifically, the explicit expressions, derived by expanding these differences, are given in the code.

Experimental results. In finite differences–case, two figures were generated per dimension: one for fixed and one for scaled increment. As others, each plot contains:

- a black curve for the reference point $\bar{x}$;
- ten colored curves, one for each random starting point;
- log–scaled $f(x_k)$ values against iterations.

The finite–differences approach is remarkably stable on this problem. All tested step sizes yield convergence in under 30 iterations. Success rates remain at 100% for all increments, confirming the robustness of the method even in the presence of approximation noise. However, when compared to exact derivatives, the convergence rate ρ is consistently lower, and runtime increases due to the extra evaluations required. For large–scale problems, this gap grows further due to the cost of computing full gradients and hessians numerically. Overall, this shows that while finite differences can provide reliable optimization results, they are computationally expensive and less efficient than analytical derivatives, when available. Below are the results, divided by problem size.

- $\boldsymbol{n = 1\,000}$

Increment	**Init.**	**Iter**	**Time (s)**	ρ	**Successes**
10^{-2}	$\bar{x}$	11	0.04	0.0216	1/1
10^{-2}	Avg (10 pts)	8.6	0.01	0.1495	10/10
$10^{-2} \cdot \lvert x \rvert$	$\bar{x}$	11	0.01	1.0263	1/1
$10^{-2} \cdot \lvert x \rvert$	Avg (10 pts)	8.2	0.00	0.9274	10/10
10^{-4}	$\bar{x}$	10	0.00	1.6064	1/1
10^{-4}	Avg (10 pts)	7.7	0.00	1.6201	10/10
$10^{-4} \cdot \lvert x \rvert$	$\bar{x}$	10	0.00	1.6122	1/1
$10^{-4} \cdot \lvert x \rvert$	Avg (10 pts)	7.7	0.00	1.6200	10/10
10^{-6}	$\bar{x}$	10	0.00	1.6112	1/1
10^{-6}	Avg (10 pts)	7.7	0.00	1.6200	10/10
$10^{-6} \cdot \lvert x \rvert$	$\bar{x}$	10	0.00	1.6113	1/1
$10^{-6} \cdot \lvert x \rvert$	Avg (10 pts)	7.7	0.00	1.6200	10/10
10^{-8}	$\bar{x}$	10	0.00	1.6113	1/1
10^{-8}	Avg (10 pts)	7.7	0.00	1.6200	10/10
$10^{-8} \cdot \lvert x \rvert$	$\bar{x}$	10	0.00	1.6113	1/1
$10^{-8} \cdot \lvert x \rvert$	Avg (10 pts)	7.7	0.00	1.6200	10/10
10^{-10}	$\bar{x}$	10	0.00	1.6113	1/1
10^{-10}	Avg (10 pts)	7.7	0.00	1.6200	10/10
$10^{-10} \cdot \lvert x \rvert$	$\bar{x}$	10	0.00	1.6113	1/1
$10^{-10} \cdot \lvert x \rvert$	Avg (10 pts)	7.7	0.00	1.6200	10/10
10^{-12}	$\bar{x}$	10	0.00	1.6113	1/1
10^{-12}	Avg (10 pts)	7.7	0.00	1.6200	10/10
$10^{-12} \cdot \lvert x \rvert$	$\bar{x}$	10	0.00	1.6113	1/1
$10^{-12} \cdot \lvert x \rvert$	Avg (10 pts)	7.7	0.00	1.6200	10/10

Table 9: Finite differences results for $n = 1000$ using different increments h.

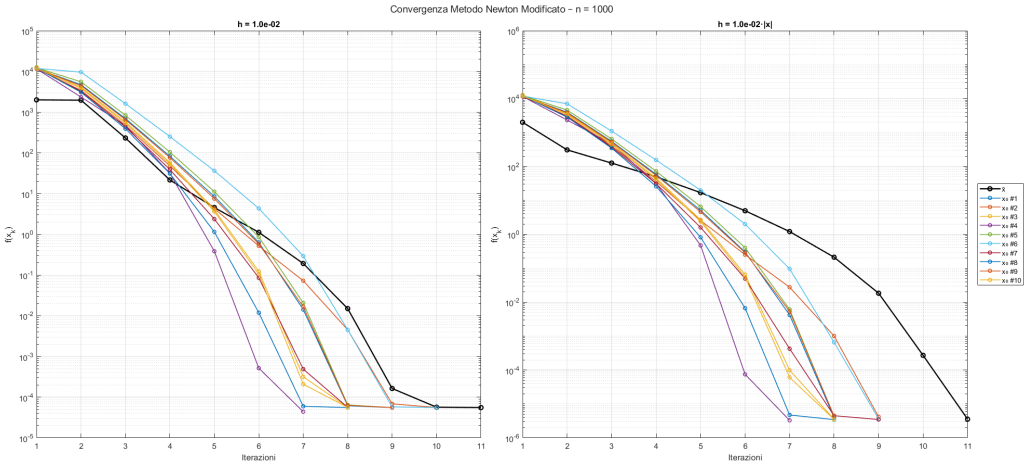


Figure 24: Convergence of Modified Newton method on Generalized Broyden function ($n = 1000$) with fixed increment $h = 10^{-2}$ (left) and scaled increment $h = 10^{-2} \cdot |x|$ (right).

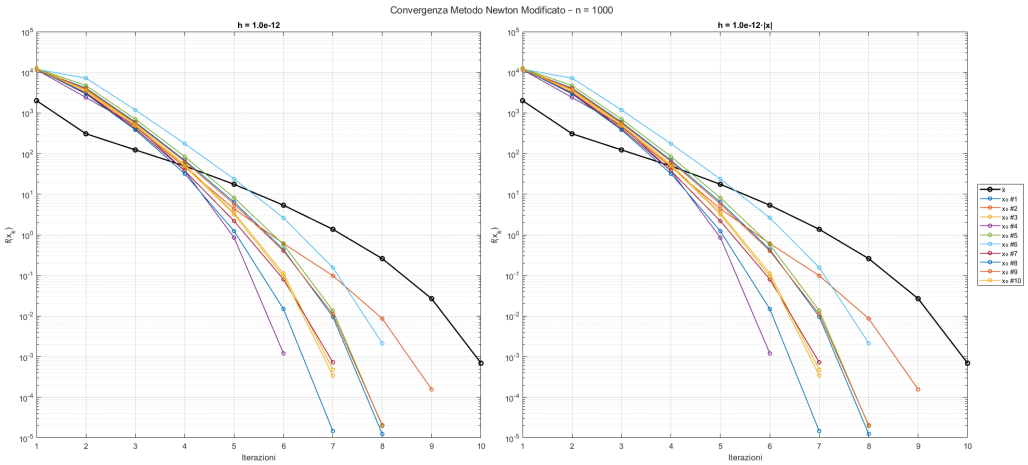


Figure 25: Convergence of Modified Newton method on Generalized Broyden function ($n = 1000$) with fixed increment $h = 10^{-12}$ (left) and scaled increment $h = 10^{-12} \cdot |x|$ (right).

- $\boldsymbol{n = 10\,000}$

Increment	**Init.**	**Iter**	**Time (s)**	ρ	**Successes**
10^{-2}	$\bar{x}$	12	0.04	0.0014	1/1
10^{-2}	Avg (10 pts)	9.4	0.04	0.0629	10/10
$10^{-2} \cdot \lvert x \rvert$	$\bar{x}$	11	0.03	0.5256	1/1
$10^{-2} \cdot \lvert x \rvert$	Avg (10 pts)	9.4	0.04	0.4660	10/10
10^{-4}	$\bar{x}$	10	0.03	1.6064	1/1
10^{-4}	Avg (10 pts)	8.5	0.03	1.5916	10/10
$10^{-4} \cdot \lvert x \rvert$	$\bar{x}$	10	0.03	1.6122	1/1
$10^{-4} \cdot \lvert x \rvert$	Avg (10 pts)	8.5	0.03	1.5916	10/10
10^{-6}	$\bar{x}$	10	0.03	1.6112	1/1
10^{-6}	Avg (10 pts)	8.5	0.03	1.5916	10/10
$10^{-6} \cdot \lvert x \rvert$	$\bar{x}$	10	0.03	1.6113	1/1
$10^{-6} \cdot \lvert x \rvert$	Avg (10 pts)	8.5	0.03	1.5916	10/10
10^{-8}	$\bar{x}$	10	0.03	1.6113	1/1
10^{-8}	Avg (10 pts)	8.5	0.03	1.5916	10/10
$10^{-8} \cdot \lvert x \rvert$	$\bar{x}$	10	0.03	1.6113	1/1
$10^{-8} \cdot \lvert x \rvert$	Avg (10 pts)	8.5	0.03	1.5916	10/10
10^{-10}	$\bar{x}$	10	0.03	1.6113	1/1
10^{-10}	Avg (10 pts)	8.5	0.03	1.5916	10/10
$10^{-10} \cdot \lvert x \rvert$	$\bar{x}$	10	0.03	1.6113	1/1
$10^{-10} \cdot \lvert x \rvert$	Avg (10 pts)	8.5	0.03	1.5916	10/10
10^{-12}	$\bar{x}$	10	0.03	1.6113	1/1
10^{-12}	Avg (10 pts)	8.5	0.03	1.5916	10/10
$10^{-12} \cdot \lvert x \rvert$	$\bar{x}$	10	0.03	1.6113	1/1
$10^{-12} \cdot \lvert x \rvert$	Avg (10 pts)	8.5	0.03	1.5916	10/10

Table 10: Finite difference results for $n = 10\,000$ using different increments h.

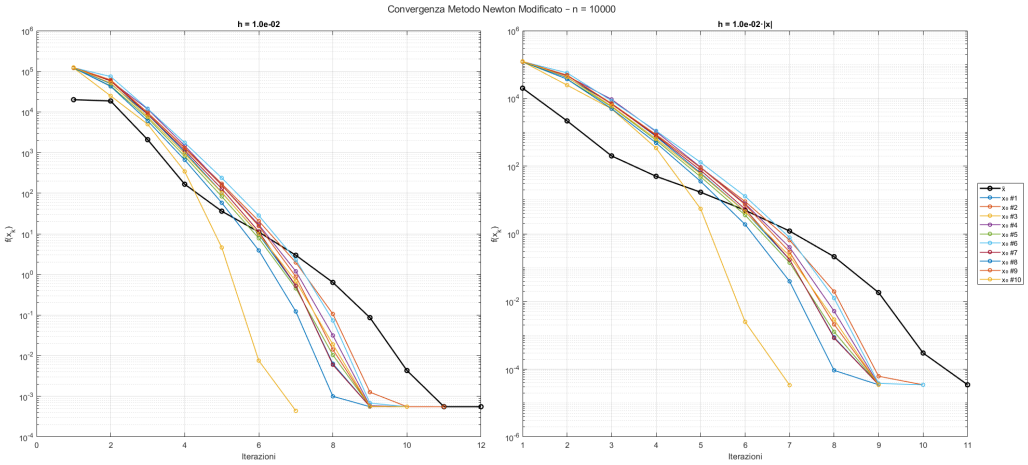


Figure 26: Convergence of Modified Newton method on Generalized Broyden function ($n = 10000$) with fixed increment $h = 10^{-2}$ (left) and scaled increment $h = 10^{-2} \cdot |x|$ (right).

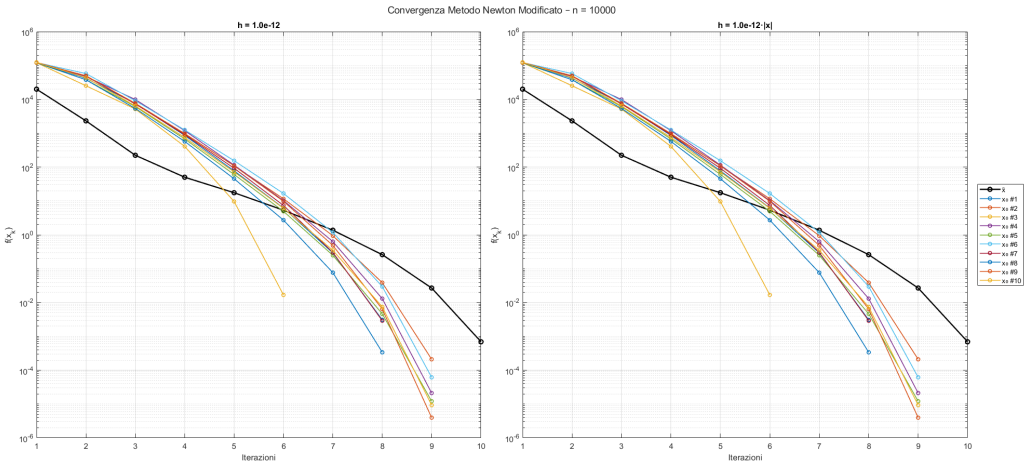


Figure 27: Convergence of Modified Newton method on Generalized Broyden function ($n = 10000$) with fixed increment $h = 10^{-12}$ (left) and scaled increment $h = 10^{-12} \cdot |x|$ (right).

- $\boldsymbol{n = 100\,000}$

Increment	**Init.**	**Iter**	**Time (s)**	ρ	**Successes**
10^{-2}	$\bar{x}$	14	0.51	0.0031	1/1
10^{-2}	Avg (10 pts)	10.9	0.56	0.0424	10/10
$10^{-2} \cdot \lvert x \rvert$	$\bar{x}$	11	0.47	0.1670	1/1
$10^{-2} \cdot \lvert x \rvert$	Avg (10 pts)	9.6	0.49	0.1538	10/10
10^{-4}	$\bar{x}$	10	0.36	1.6064	1/1
10^{-4}	Avg (10 pts)	8.7	0.43	1.6182	10/10
$10^{-4} \cdot \lvert x \rvert$	$\bar{x}$	10	0.34	1.6122	1/1
$10^{-4} \cdot \lvert x \rvert$	Avg (10 pts)	8.7	0.43	1.6173	10/10
10^{-6}	$\bar{x}$	10	0.35	1.6112	1/1
10^{-6}	Avg (10 pts)	8.7	0.43	1.6171	10/10
$10^{-6} \cdot \lvert x \rvert$	$\bar{x}$	10	0.35	1.6113	1/1
$10^{-6} \cdot \lvert x \rvert$	Avg (10 pts)	8.7	0.43	1.6170	10/10
10^{-8}	$\bar{x}$	10	0.35	1.6113	1/1
10^{-8}	Avg (10 pts)	8.7	0.43	1.6170	10/10
$10^{-8} \cdot \lvert x \rvert$	$\bar{x}$	10	0.35	1.6113	1/1
$10^{-8} \cdot \lvert x \rvert$	Avg (10 pts)	8.7	0.43	1.6170	10/10
10^{-10}	$\bar{x}$	10	0.35	1.6113	1/1
10^{-10}	Avg (10 pts)	8.7	0.43	1.6170	10/10
$10^{-10} \cdot \lvert x \rvert$	$\bar{x}$	10	0.35	1.6113	1/1
$10^{-10} \cdot \lvert x \rvert$	Avg (10 pts)	8.7	0.43	1.6170	10/10
10^{-12}	$\bar{x}$	10	0.36	1.6113	1/1
10^{-12}	Avg (10 pts)	8.7	0.43	1.6170	10/10
$10^{-12} \cdot \lvert x \rvert$	$\bar{x}$	10	0.35	1.6113	1/1
$10^{-12} \cdot \lvert x \rvert$	Avg (10 pts)	8.7	0.43	1.6170	10/10

Table 11: Finite difference results for $n = 100\,000$ using different increments h.

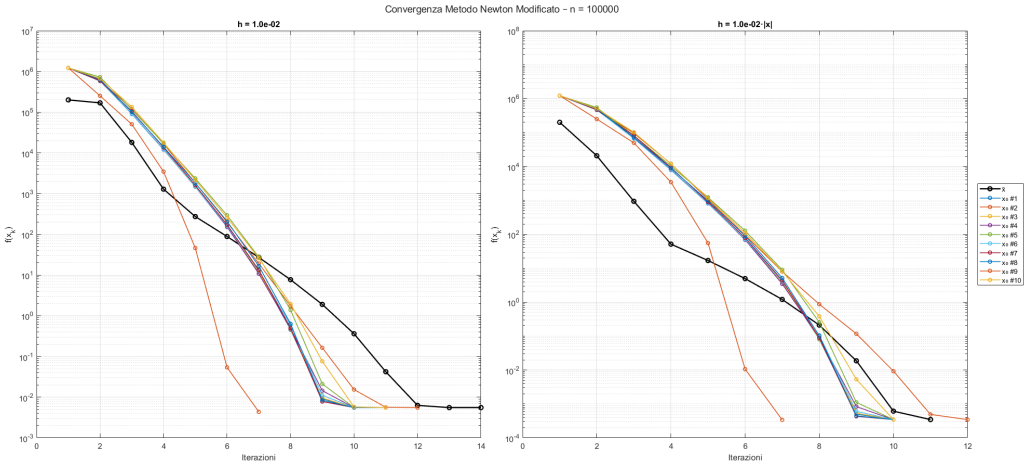


Figure 28: Convergence of Modified Newton method on Generalized Broyden function ($n = 100000$) with fixed increment $h = 10^{-2}$ (left) and scaled increment $h = 10^{-2} \cdot |x|$ (right).

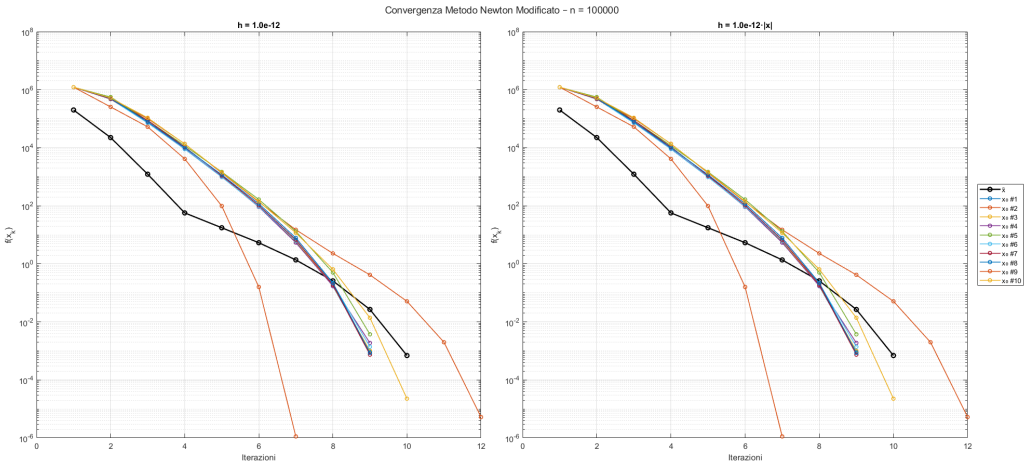


Figure 29: Convergence of Modified Newton method on Generalized Broyden function ($n = 100000$) with fixed increment $h = 10^{-12}$ (left) and scaled increment $h = 10^{-12} \cdot |x|$ (right).

4.3 Nelder–Mead method

In this subsection are shown the results obtained from the minimization of the Generalized Broyden function using the Nelder–Mead method. Before presenting the results, a general experimental setup is given:

- `n` $= 10, 26, 50$;
- `max_iter` $= 80.000$;
- `tol` $= 10^{-6}$.

Moreover, for each run have recorded:

- number of iterations to convergence;
- CPU time;
- final objective value found f_{min};
- experimental rate of convergence ρ:

$$\rho \approx \frac{\log\left(\|x^{(k+1)} - x^{(k)}\| \big/ \|x^{(k)} - x^{(k-1)}\|\right)}{\log\left(\|x^{(k)} - x^{(k-1)}\| \big/ \|x^{(k-1)} - x^{(k-2)}\|\right)}.$$

In correspondence with the randomly generated points, an average behavior of each of the previous catogories is reported.

Dimension	**Starting point**	$f_{\min}$	**Iter**	**Time (s)**	ρ
10	$\bar{x}$	0.036184	1510	0.03	-1.2743
	Avg (10 pts)	4.974638	1467	0.018	0.3595
26	$\bar{x}$	0.187379	7040	0.11	-0.7587
	Avg (10 pts)	45.657701	11562	0.169	-3.0863
50	$\bar{x}$	0.206557	36027	0.83	-0.6742
	Avg (10 pts)	92.211543	52928	1.218	-2.2175

Table 12: Results of Nelder–Mead on Generalized Broyden tridiagonal function.

Experimental results. The Nelder–Mead method performs reasonably well in very low dimensions, such as $n = 10$, where it achieves successful convergence in most runs and a moderate number of iterations. However, as the problem dimension increases, the method becomes significantly less effective. For $n = 25$ and $n = 50$, none of the runs achieved a function value below the target threshold, and both the number of iterations and runtime increase considerably. The convergence rate ρ also becomes unstable, with values indicating erratic or oscillatory behavior. These results highlight the limitations of Nelder–Mead in structured, high-dimensional, non-separable optimization problems, especially when compared to Newton-type methods equipped with derivative information.

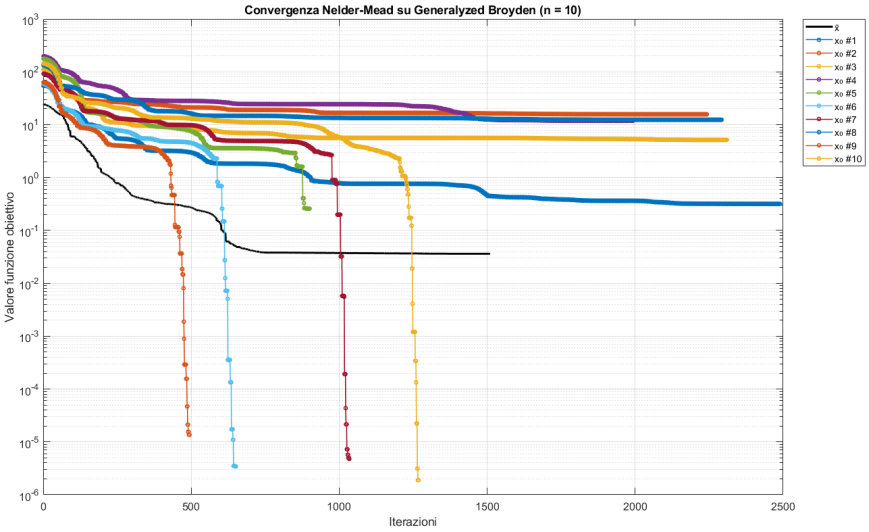


Figure 30: Convergence of Nelder-Mead method on Generalized Broyden function ($n = 10$) for the reference point $\bar{x}$ (black) and 10 randomly generated starting points.

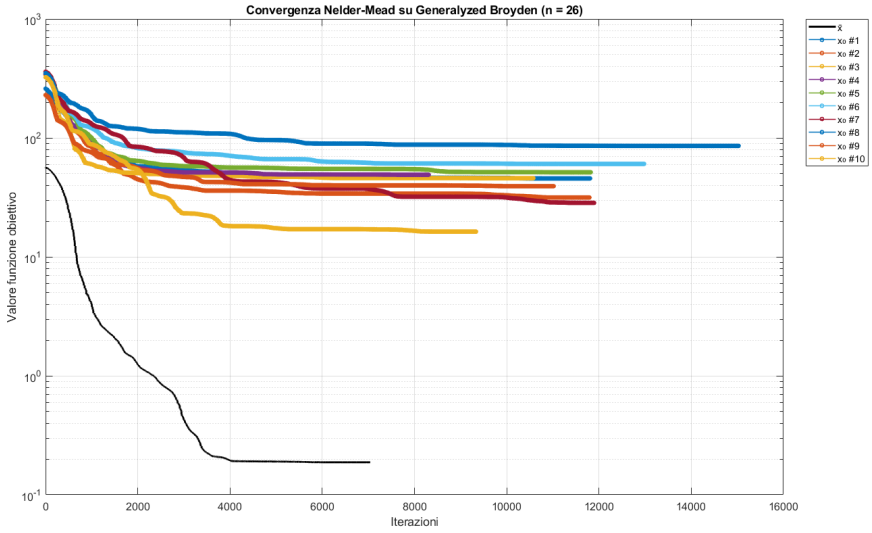


Figure 31: Convergence of Nelder-Mead method on Generalized Broyden function ($n = 26$) for the reference point $\bar{x}$ (black) and 10 randomly generated starting points.

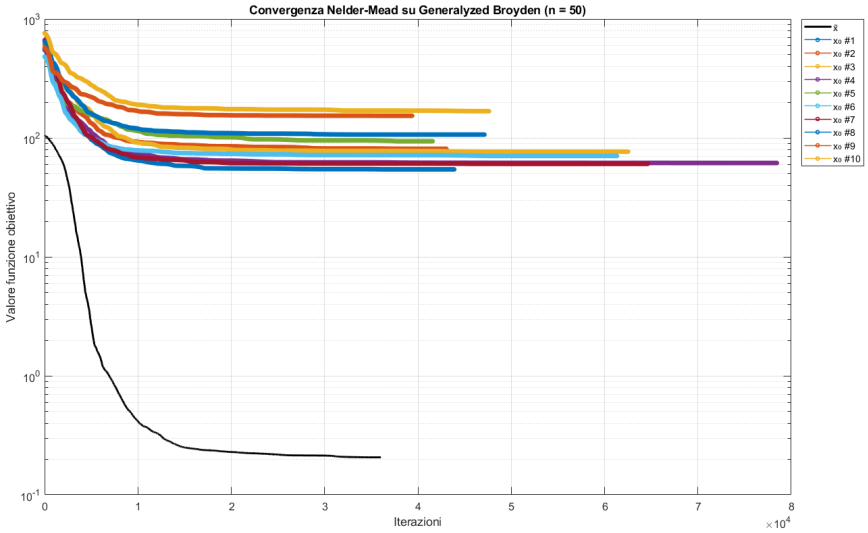


Figure 32: Convergence of Nelder-Mead method on Generalized Broyden function ($n = 50$) for the reference point $\bar{x}$ (black) and 10 randomly generated starting points.

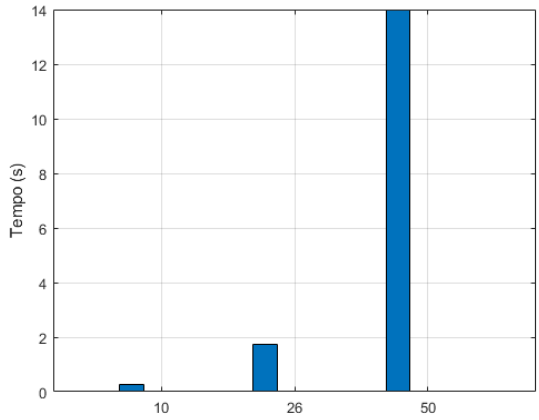


Figure 33: Computational time (in seconds) for Nelder-Mead method applied to the Generalized Broyden function for increasing dimensions.

5 Banded Trigonometric Function

5.1 Problem introduction

The Banded Trigonometric function is a structured nonlinear objective function characterized by trigonometric dependencies among three consecutive variables. It is defined as:

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

with boundary conditions $x_0 = x_{n+1} = 0$, where $x \in \mathbb{R}^n$. This objective function arises in problems where local periodic variations influence adjacent terms. Its structure creates a banded dependency pattern, meaning each component interacts only with its two immediate neighbors. Because of the involvement of the sine and cosine trigonometric functions, the Banded Trigonometric function has a single minimum but not a single minimizer, as can be seen in the plot reported in figure (34). In this chapter we are going to analyze the behaviour of both the Modified Newton and the Nelder–Mead methods, when they are applied to minimize the Banded Trigonometric function. The initial points suggested in the benchmark library are $\bar{x} \in \mathbb{R}^n$ such that

$$\bar{x}_k = 1.0 \quad \text{for} \quad k = 1, 2, \ldots, n,$$

together with another 10 random initial points sampled uniformly in the hypercube $[\bar{x}_1 - 1, \bar{x}_1 + 1] \times \cdots \times [\bar{x}_n - 1, \bar{x}_n + 1] \subset \mathbb{R}^n$, starting from $\bar{x}$.

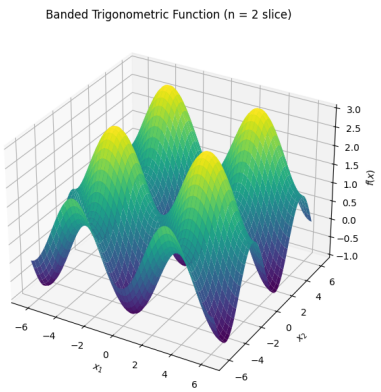


Figure 34: 3D visualization of the Banded Trigonometric function for $n = 2$.

5.2 Modified Newton method

In this subsection are shown the results obtained from the minimization of the Banded Trigonometric function using the Modified Newton method; the study includes both exact derivatives and derivatives computed by finite-differences approximations. Due to its diagonal structure, the hessian is stored and manipulated in sparse format; this choice drastically reduces memory requirements and computational cost (in both matrix factorization and Newton direction computation), allowing efficiently handle large–scale problems.

For this particular function the resulting hessian matrices are often ill-conditioned, especially in higher dimensions. To address this, the Modified Newton method were designed to apply preconditioning through the MATLAB `pcg` solver (Preconditioned Conjugate Gradient). This choice enables efficient direction computation even when Cholesky factorization becomes unreliable or fails, particularly for large-scale sparse systems. The algorithm applies a modified Cholesky approach (Algorithm 6.3 in) to shift the hessian until a positive definite approximation is obtained. Then, an incomplete Cholesky factorization is used to precondition the matrix, ensuring robust convergence even in the presence of large condition numbers.

Before presenting detailed outcomes, a general experimental setup is given:

- `n` $= 10^3, 10^4, 10^5$;
- `max_iter` $= 5000$;
- `tol` $= 10^{-6}$;
- `max_iter_pcg` $= 50$;
- `tol_pcg` $= 10^{-6}$;
- `preconditioner_pcg` $= LL^T$, where the factor L is obtained with the incomplete Cholesky factorization of the coefficient matrix.

For each run, the following were tracked:

- number of iterations to convergence;
- CPU time;
- number of successful runs;

- experimental rate of convergence ρ:

$$\rho \approx \frac{\log\left(\|x^{(k+1)} - x^{(k)}\| \big/ \|x^{(k)} - x^{(k-1)}\|\right)}{\log\left(\|x^{(k)} - x^{(k-1)}\| \big/ \|x^{(k-1)} - x^{(k-2)}\|\right)}.$$

In correspondence with the randomly generated points, an average behavior of each of the previous catogories is reported. It should be emphasized that among the data shown within the tables, rho values approximately equal to or greater than three can be identified. These results are not a correct approximation of the required rate of convergence, as this is at most quadratic in the case of Modified Newton method; These results could be traced to replacing the optimal value of the minimizer with its last available approximation.

Furthermore, the code is designed to work also with finite–differences case, and this variant is implemented thanks to the extra parameters

- `h`: a parameter equal to 10^{-k}, for $k = 2, 4, 6, 8, 10, 12$;
- `type`: a parameter which indicates if the increment is scaled componentwise as $h_i = 10^{-k} \cdot |x_i|$ or if it is a default increment such that $h_i = 10^{-k}$.

For this part, the several features were computed for each stepsize h.

5.2.1 Modified Newton method with exact derivatives

In this section the Banded Trigonometric function is minimized using the Modified Newton method with exact gradient and hessian formulas. Both were derived analytically and implemented exploiting the banded pattern of the function. Specifically:

- each gradient's component is constructed as:

$$\frac{\partial F}{\partial x_k} = \begin{cases} \sin(x_1) + 2\cos(x_1), & i = 1 \\ k\sin(x_k) + 2\cos(x_k), & 2 \leq k \leq n-1 \\ n\sin(x_n) - (n-1)\cos(x_n), & k = n \end{cases};$$

- each entry of the hessian matrix is computed as:

$$\frac{\partial^2 F}{\partial x_k \partial x_j} = \begin{cases} \cos(x_1) - 2\sin(x_1), & i = 1 \\ k\cos(x_k) - 2\sin(x_k), & 2 \leq k \leq n-1 \\ n\cos(x_n) + (n-1)\sin(x_n), & i = n \\ 0, & \text{otherwise} \end{cases}.$$

Experimental Results. As shown in table (5.2.1), the Modified Newton method with exact derivatives fails in terms of minimizing the Banded Trigonometric function, which turns out to be a particularly challenging function to optimize, if compared to the ones in the other sections. In all dimensions tested and for all initialization points, not only is convergence not satisfied, but the f_{min} values achieved are far from the actual global minimum. Despite preconditioning the hessian matrix during the search for the descent direction, the algorithm fails to reach a value that is reasonably close to the minimum of the function. Below are tables and figures explaining the situation in detail.

Dim	**Init.**	**Iter**	**Time (s)**	$f_{\min}$	ρ	**Successes**
10^3	$\bar{x}$	6	0.04	-427.40	2.52	1/1
	Avg (10 pts)	16.4	0.02	-565.12	2.62	10/10
10^4	$\bar{x}$	7	0.04	-4159.93	2.22	1/1
	Avg (10 pts)	20.4	0.22	-2835.42	2.25	10/10
10^5	$\bar{x}$	6	0.36	-41443.76	4.11	1/1
	Avg (10 pts)	29.7	4.78	-37052.49	1.78	10/10

Table 13: Results of Modified Newton method on Banded Trigonometric function using exact derivatives.

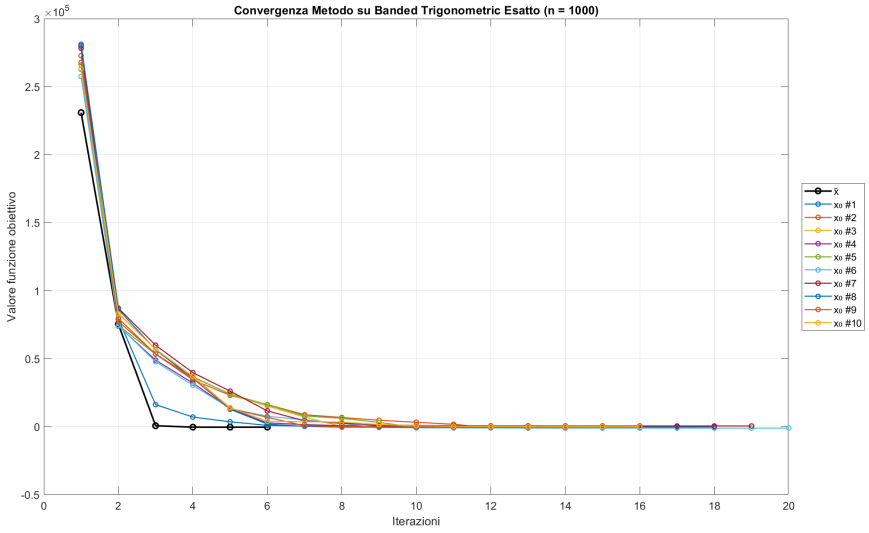


Figure 35: Convergence of the Modified Newton method on the Banded Trigonometric function ($n = 1000$) using exact derivatives.

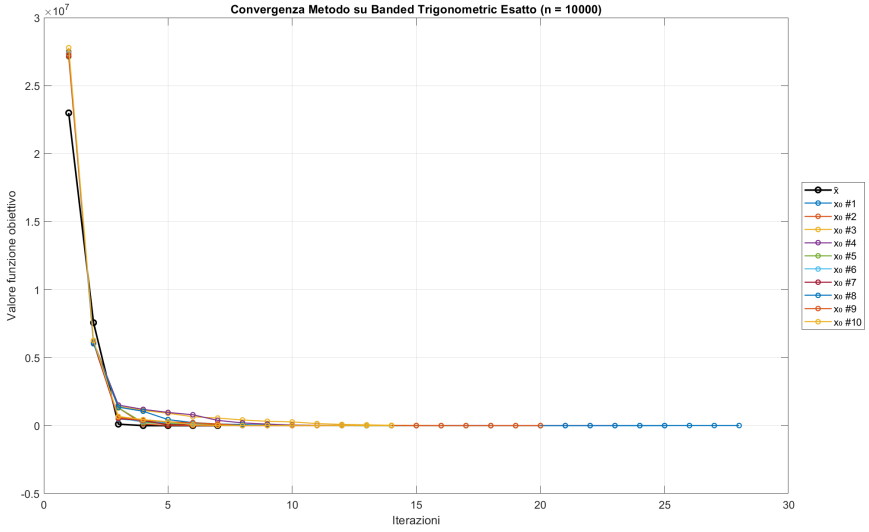


Figure 36: Convergence of the Modified Newton method on the Banded Trigonometric function ($n = 10000$) using exact derivatives.

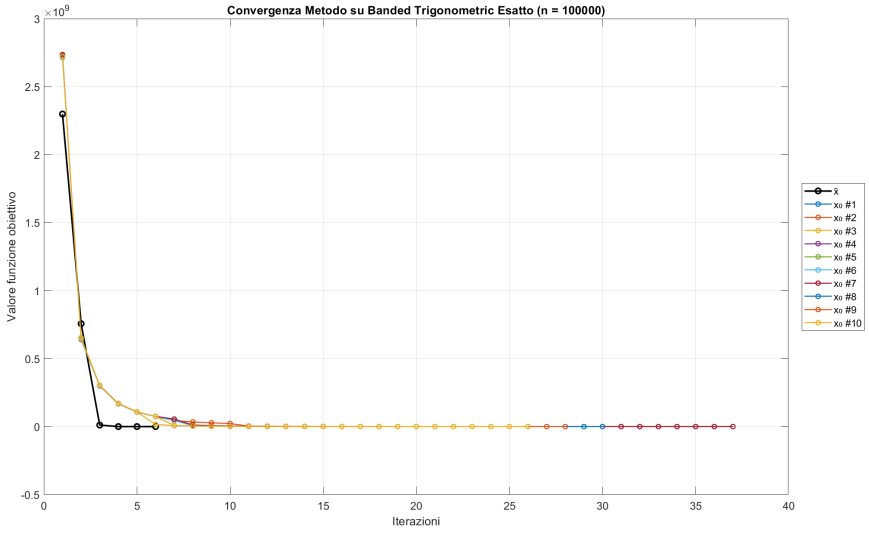


Figure 37: Convergence of the Modified Newton method on the Banded Trigonometric function ($n = 100000$) using exact derivatives.

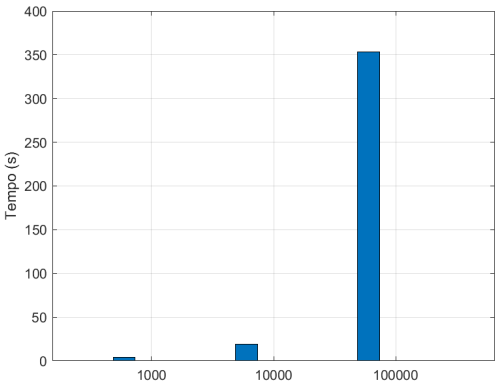


Figure 38: Computational time required by the Modified Newton method on the Banded Trigonometric function with exact derivatives for each problem size.

5.2.2 Modified Newton method with approximated derivatives

In this experiment, we use the Modified Newton method on the Banded Trigonometric function with gradient and hessian computed via finite differences. The approximated gradient and hessian components were expanded manually and implemented directly to avoid cancellation and redundancy. The sparsity pattern of the exact hessian was preserved to limit computational cost. The expressions used for approximating the gradient and the hessian are reported below, following the same structure adopted in previous sections.

Gradient approximation. Due to the structure of the function, only a small number of addends to the summation survive when computing the numerator in the formula (1). These terms are identified by the following function:

$$g(x) = \sum_{i=k-1}^{k+1} i \left[(1 - \cos(x_i)) + \sin(x_{i-1}) - \sin(x_{i+1}) \right].$$

Then, using subtraction and addition formulas for sine and cosine, and fixed k as the component whose derivative is being approximated:

$$\frac{\partial F}{\partial x_k} \approx \frac{g(x + he_k) - g(x + he_k)}{2h} = \begin{cases} \frac{4 \cos x_k \sin h + 2k \sin x_k \sin h}{2h}, & 1 \leq k < n \\ \frac{2n \sin x_n \sin h - 2(n-1) \cos x_k \sin h}{2h}, & k = n \end{cases}. \tag{7}$$

Hessian approximation. Also in the hessian computation, the definition of the function $g(x)$ can be used to describe the terms in the numerator of (2). Rearranging the terms and using a Taylor expansion for the terms involving the cosine of h, is obtained:

$$\frac{\partial^2 F}{\partial x_k \partial x_j} \approx \begin{cases} (k \cos x_k - 2 \sin x_k) - h(k \sin x_k + 2 \cos x_k), & 1 \leq k = j < n \\ (n \cos x_n - (n-1) \sin x_n) - h(n \sin x_n - (n-1) \cos x_n), & k = j = n \\ 0, & \text{otherwise} \end{cases}. \tag{8}$$

Experimental Results. Also in this cas the Modified Newton method in unable to correctly optimize the Banded Trigonometric function. In all dimensions tested and for all starting points (suggested or randomly generated), convergence is not met, and the algorithm stops after a few iterations regardless of whether the increment used in calculating finite differences is fixed or scaled by component. In this case it makes no sense to talk about the rate of convergence of the method implemented, in fact in some cases it cannot even be calculated (NaN). More precise results are captured in the following tables and figures, divided by problem size.

- $\boldsymbol{n = 1\,000}$

Increment	**Init.**	**Iter**	**Time (s)**	ρ	**Successes**
10^{-2}	$\bar{x}$	6	0.33	8.1635	1/1
10^{-2}	Avg (10 pts)	15.9	0.03	5.44	7/10
$10^{-2} \cdot \lvert x \rvert$	$\bar{x}$	1	0.00	NaN	0/10
$10^{-2} \cdot \lvert x \rvert$	Avg (10 pts)	519	0.39	3.50	7/10
10^{-4}	$\bar{x}$	7	0.03	1.8294	0/1
10^{-4}	Avg (10 pts)	18.1	0.03	5.35	9/10
$10^{-4} \cdot \lvert x \rvert$	$\bar{x}$	1	0.00	NaN	0/1
$10^{-4} \cdot \lvert x \rvert$	Avg (10 pts)	16.3	0.02	6.00	8/10
10^{-6}	$\bar{x}$	7	0.01	1.1623	1/1
10^{-6}	Avg (10 pts)	15.8	0.02	5.39	6/10
$10^{-6} \cdot \lvert x \rvert$	$\bar{x}$	1	0.00	NaN	0/1
$10^{-6} \cdot \lvert x \rvert$	Avg (10 pts)	15.5	0.02	5.20	7/10
10^{-8}	$\bar{x}$	7	0.01	6.4828	1/1
10^{-8}	Avg (10 pts)	14.9	0.02	6.45	6/10
$10^{-8} \cdot \lvert x \rvert$	$\bar{x}$	1	0.00	NaN	0/1
$10^{-8} \cdot \lvert x \rvert$	Avg (10 pts)	15	0.02	7.63	6/10
10^{-10}	$\bar{x}$	7	0.01	6.4832	1/1
10^{-10}	Avg (10 pts)	15.1	0.02	7.15	6/10
$10^{-10} \cdot \lvert x \rvert$	$\bar{x}$	1	0.00	NaN	0/1
$10^{-10} \cdot \lvert x \rvert$	Avg (10 pts)	15.1	0.03	7.28	7/10
10^{-12}	$\bar{x}$	7	0.01	6.4832	1/1
10^{-12}	Avg (10 pts)	15.1	0.02	7.01	7/10
$10^{-12} \cdot \lvert x \rvert$	$\bar{x}$	1	0.00	NaN	0/1
$10^{-12} \cdot \lvert x \rvert$	Avg (10 pts)	15.1	0.03	7.01	7/10

Table 14: Finite difference results for $n = 1\,000$ using different increments h and strategies.

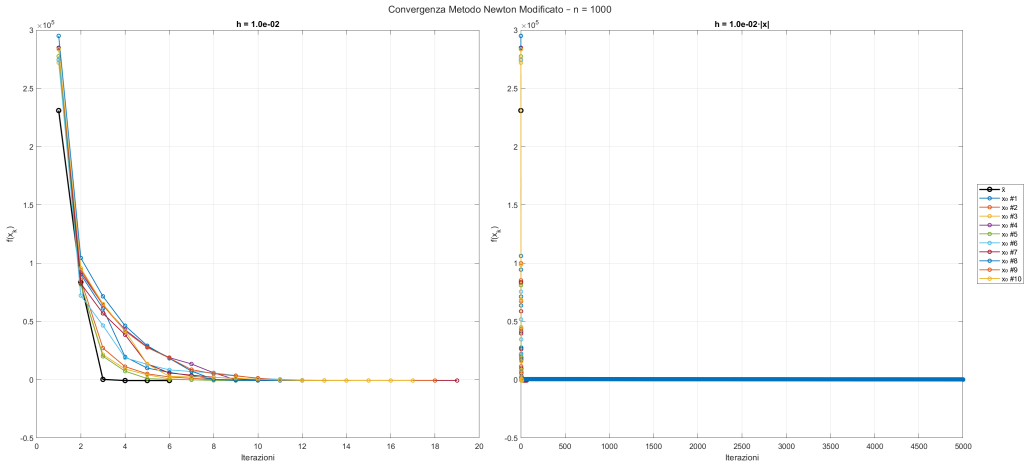


Figure 39: Convergence of the Modified Newton method on the Banded Trigonometric function ($n = 1000$) using fixed increment $h = 10^{-2}$ (left) and scaled increment $h = 10^{-2} \cdot |x|$ (right).

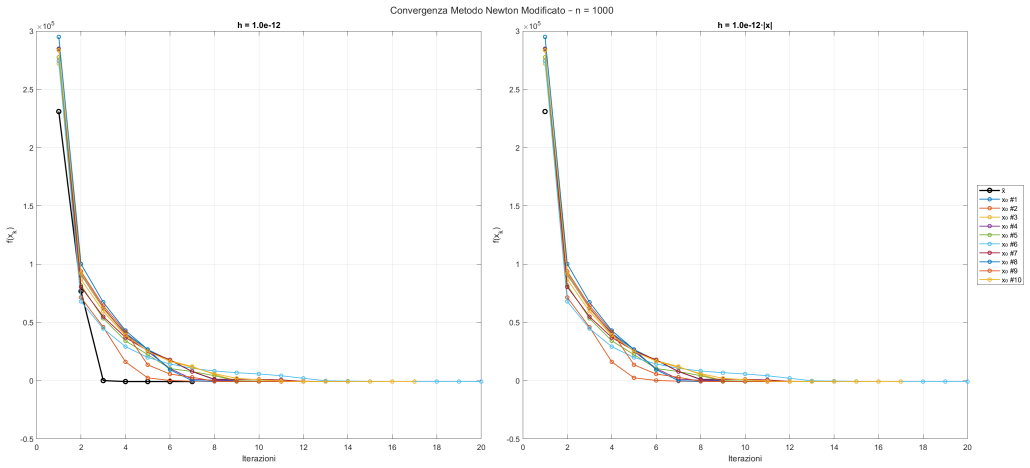


Figure 40: Convergence of the Modified Newton method on the Banded Trigonometric function ($n = 1000$) using fixed increment $h = 10^{-12}$ (left) and scaled increment $h = 10^{-12} \cdot |x|$ (right).

- $\boldsymbol{n = 10\,000}$

Increment	**Init.**	**Iter**	**Time (s)**	ρ	**Successes**
10^{-2}	$\bar{x}$	6	0.05	-7087.89	1/1
10^{-2}	Avg (10 pts)	21.6	0.25	3.65	9/10
$10^{-2} \cdot \lvert x \rvert$	$\bar{x}$	1	0.01	NaN	0/1
$10^{-2} \cdot \lvert x \rvert$	Avg (10 pts)	31.9	0.27	2.84	7/10
10^{-4}	$\bar{x}$	7	0.06	1.83	0/1
10^{-4}	Avg (10 pts)	20.8	0.24	6.73	8/10
$10^{-4} \cdot \lvert x \rvert$	$\bar{x}$	1	0.01	NaN	0/1
$10^{-4} \cdot \lvert x \rvert$	Avg (10 pts)	20	0.23	5.13	8/10
10^{-6}	$\bar{x}$	7	0.06	1.16	1/1
10^{-6}	Avg (10 pts)	20.5	0.23	4.48	8/10
$10^{-6} \cdot \lvert x \rvert$	$\bar{x}$	1	0.01	NaN	0/1
$10^{-6} \cdot \lvert x \rvert$	Avg (10 pts)	19.3	0.22	1.11	7/10
10^{-8}	$\bar{x}$	7	0.05	3.4828	1/1
10^{-8}	Avg (10 pts)	20.2	0.23	6.38	9/10
$10^{-8} \cdot \lvert x \rvert$	$\bar{x}$	1	0.01	NaN	0/1
$10^{-8} \cdot \lvert x \rvert$	Avg (10 pts)	19.8	0.22	6.30	8/10
10^{-10}	$\bar{x}$	7	0.05	6.4832	1/1
10^{-10}	Avg (10 pts)	20.2	0.22	4.87	9/10
$10^{-10} \cdot \lvert x \rvert$	$\bar{x}$	1	0.01	NaN	0/1
$10^{-10} \cdot \lvert x \rvert$	Avg (10 pts)	20.5	0.23	4.48	9/10
10^{-12}	$\bar{x}$	7	0.06	6.4832	1/1
10^{-12}	Avg (10 pts)	20.1	0.22	4.98	9/10
$10^{-12} \cdot \lvert x \rvert$	$\bar{x}$	1	0.01	NaN	0/1
$10^{-12} \cdot \lvert x \rvert$	Avg (10 pts)	20.2	0.22	4.69	9/10

Table 15: Finite difference results for $n = 10\,000$ using different increments h and strategies.

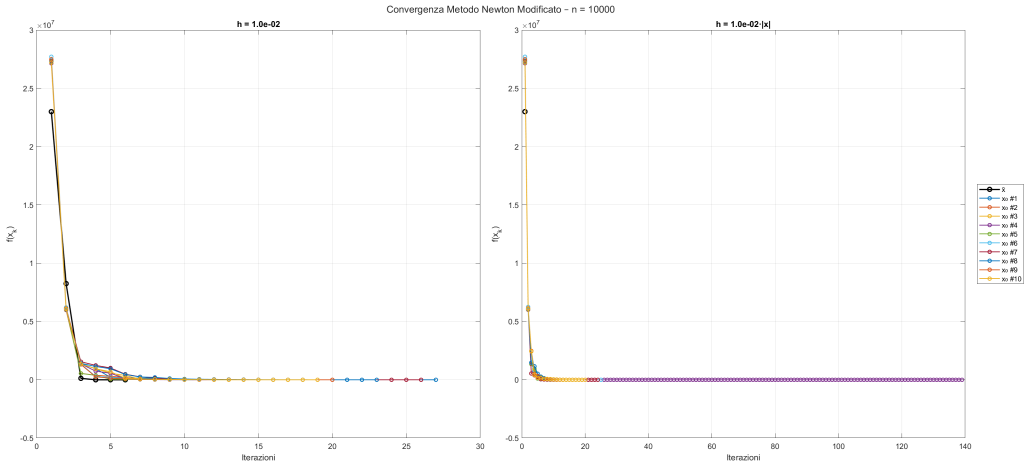


Figure 41: Convergence of the Modified Newton method on the Banded Trigonometric function ($n = 10000$) using fixed increment $h = 10^{-2}$ (left) and scaled increment $h = 10^{-2} \cdot |x|$ (right).

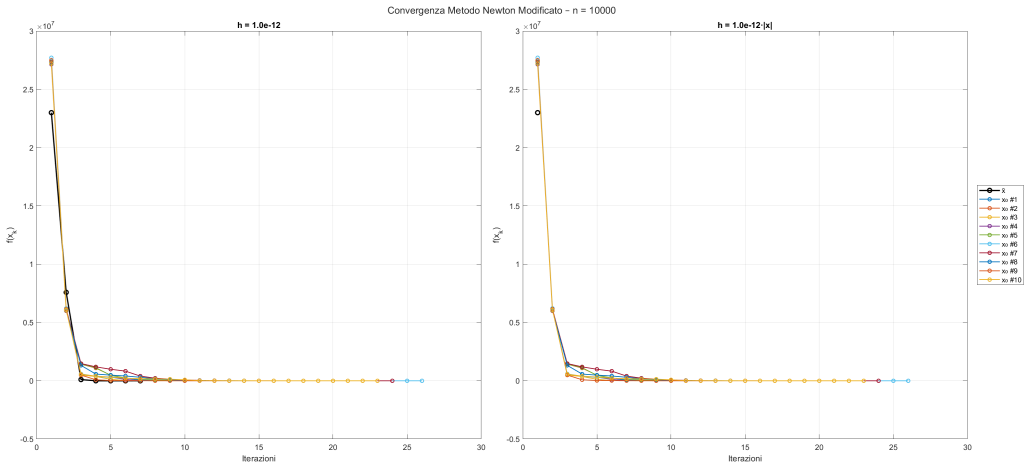


Figure 42: Convergence of the Modified Newton method on the Banded Trigonometric function ($n = 10000$) using fixed increment $h = 10^{-12}$ (left) and scaled increment $h = 10^{-12} \cdot |x|$ (right).

- $\boldsymbol{n = 100\,000}$

Increment	**Init.**	**Iter**	**Time (s)**	ρ	**Successes**
10^{-2}	$\bar{x}$	6	0.37	8.1604	1/1
10^{-2}	Avg (10 pts)	29.3	4.59	6.36	8/10
$10^{-2} \cdot \lvert x \rvert$	$\bar{x}$	1	0.09	NaN	0/1
$10^{-2} \cdot \lvert x \rvert$	Avg (10 pts)	41.3	7.00	4.22	8/10
10^{-4}	$\bar{x}$	7	0.50	1.8294	0/1
10^{-4}	Avg (10 pts)	28.4	4.61	4.51	8/10
$10^{-4} \cdot \lvert x \rvert$	$\bar{x}$	1	0.09	NaN	0/1
$10^{-4} \cdot \lvert x \rvert$	Avg (10 pts)	29	4.59	5.31	8/10
10^{-6}	$\bar{x}$	7	0.50	1.1623	1/1
10^{-6}	Avg (10 pts)	28.9	4.92	6.26	8/10
$10^{-6} \cdot \lvert x \rvert$	$\bar{x}$	1	0.09	NaN	0/1
$10^{-6} \cdot \lvert x \rvert$	Avg (10 pts)	28.6	4.72	8.31	7/10
10^{-8}	$\bar{x}$	7	0.58	6.4828	1/1
10^{-8}	Avg (10 pts)	29.6	4.70	4.01	8/10
$10^{-8} \cdot \lvert x \rvert$	$\bar{x}$	1	0.09	NaN	0/1
$10^{-8} \cdot \lvert x \rvert$	Avg (10 pts)	30.8	5.19	6.75	8/10
10^{-10}	$\bar{x}$	7	0.53	6.4832	1/1
10^{-10}	Avg (10 pts)	28	4.43	5.83	9/10
$10^{-10} \cdot \lvert x \rvert$	$\bar{x}$	1	0.09	NaN	0/1
$10^{-10} \cdot \lvert x \rvert$	Avg (10 pts)	31.1	4.95	2.63	9/10
10^{-12}	$\bar{x}$	7	0.51	6.4832	1/1
10^{-12}	Avg (10 pts)	29.8	4.83	0.88	8/10
$10^{-12} \cdot \lvert x \rvert$	$\bar{x}$	1	0.09	NaN	0/1
$10^{-12} \cdot \lvert x \rvert$	Avg (10 pts)	30.9	4.86	4.18	8/10

Table 16: Finite difference results for $n = 100\,000$ using different increments h and strategies.

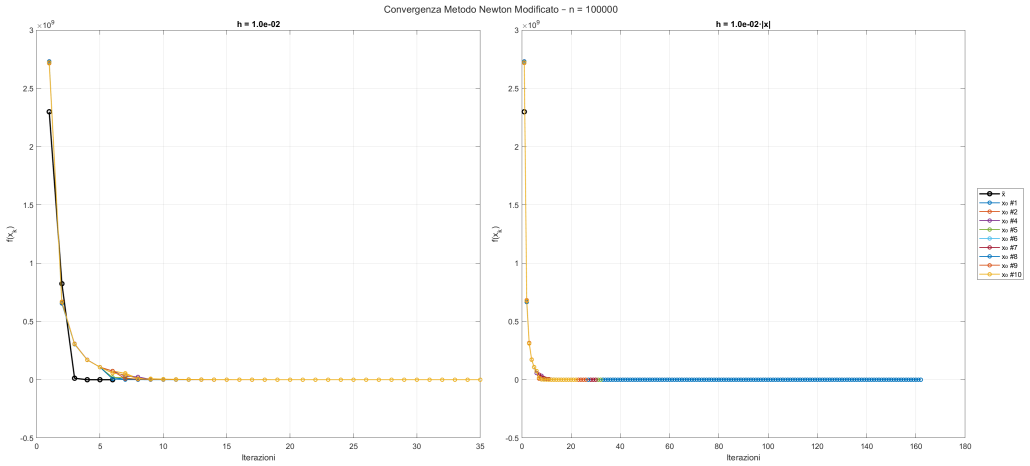


Figure 43: Convergence of the Modified Newton method on the Banded Trigonometric function ($n = 100000$) using fixed increment $h = 10^{-2}$ (left) and scaled increment $h = 10^{-2} \cdot |x|$ (right).

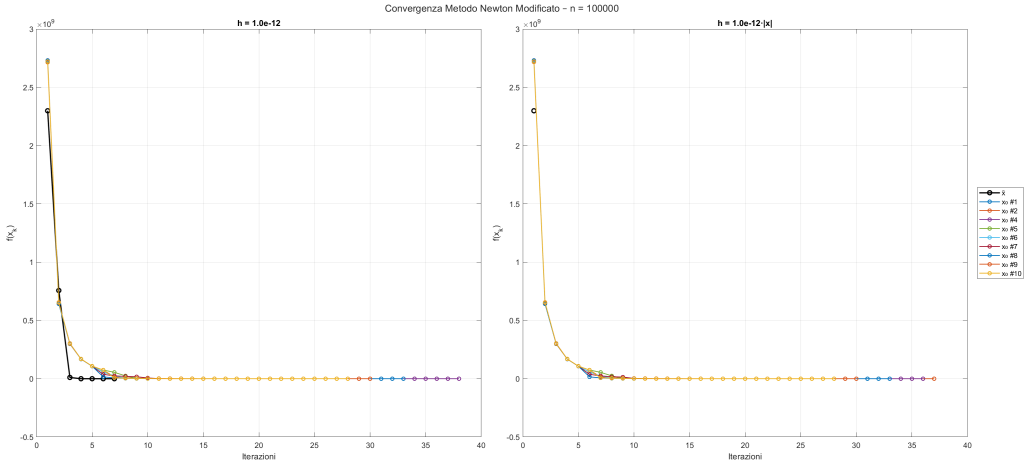


Figure 44: Convergence of the Modified Newton method on the Banded Trigonometric function ($n = 100000$) using fixed increment $h = 10^{-12}$ (left) and scaled increment $h = 10^{-12} \cdot |x|$ (right).

5.3 Nelder–Mead method

In this subsection are shown the results obtained from the minimization of the Banded Trigonometric function using the Nelder–Mead method. This method does not exploit derivative information and therefore struggles more with ill-conditioned landscapes and high-dimensional domains. As expected, the performance worsens as the dimension increases.

Before presenting the results, a general experimental setup is given:

- `n` $= 10, 26, 50$;
- `max_iter` $= 100.000$
- `tol` $= 10^{-6}$.

Moreover, for each run have been recorded:

- number of iterations to convergence;
- CPU time;
- final objective value found f_{min};
- experimental rate of convergence ρ:

$$\rho \approx \frac{\log\left(\|x^{(k+1)} - x^{(k)}\| \big/ \|x^{(k)} - x^{(k-1)}\|\right)}{\log\left(\|x^{(k)} - x^{(k-1)}\| \big/ \|x^{(k-1)} - x^{(k-2)}\|\right)}.$$

In correspondence with the randomly generated points, an average behavior of each of the previous catogories is reported.

Experimental results. The experimental results for the Banded Trigonometric function confirm the limitations of the Nelder–Mead method in high-dimensional and structured nonlinear problems. For $n = 10$, the method achieves very low computational times and iteration counts, but only 2 out of 10 runs with random initializations are successful, and function values remain distant from the global minimum in most cases. As the dimension increases to $n = 26$ and $n = 50$, the number of iterations and total computational time increase drastically, with several runs hitting the iteration cap (notably, two runs at $n = 26$ required 80000 iterations and over 50 seconds). No successful runs are observed for these higher dimensions, and

the convergence rates ρ become highly erratic, alternating between extreme positive and negative values, indicating a lack of stability and robustness. Overall, the method fails to provide accurate or reliable solutions in medium and high dimensions, reinforcing the need for derivative-based or more advanced global optimization methods when tackling large-scale, structured, and highly nonlinear problems.

Dimension	**Starting point**	$f_{\min}$	**Iter**	**Time (s)**	ρ
10	$\bar{x}$	-2.180187	187	0.01	0.5000
	Avg (10 pts)	-9.743851	1297	0.019	0.5390
26	$\bar{x}$	-6.731719	1376	0.04	-7.0805
	Avg (10 pts)	2.086111	77855	6.393	4.1805
50	$\bar{x}$	-5.995540	4955	0.28	-1.8208
	Avg (10 pts)	-2.092789	65323	0.429	0.1752

Table 17: Results of Nelder–Mead on Banded Trigonometric function.

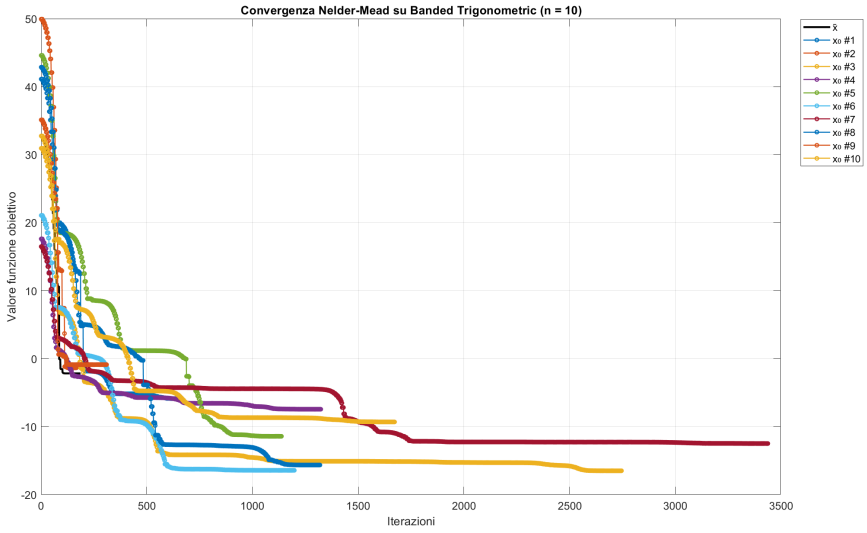


Figure 45: Convergence of Nelder-Mead on the Banded Trigonometric function with $n = 10$.

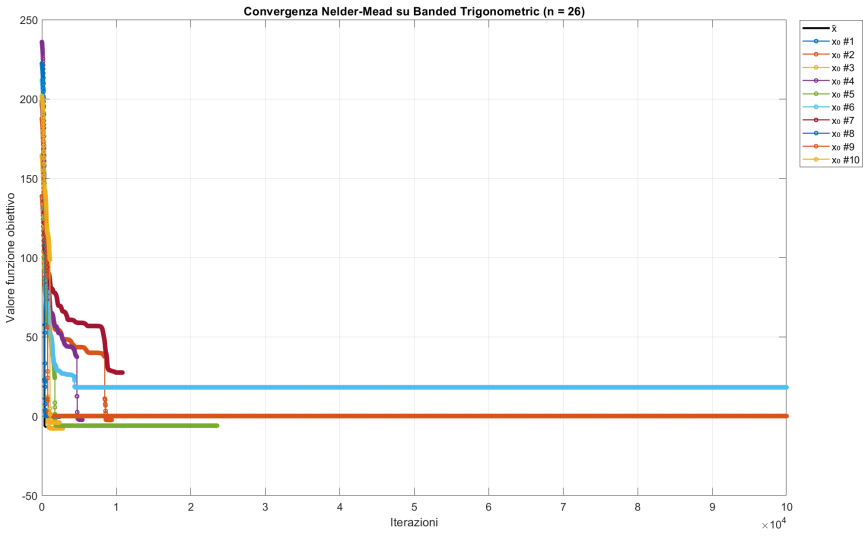


Figure 46: Convergence of Nelder-Mead on the Banded Trigonometric function with $n = 26$.

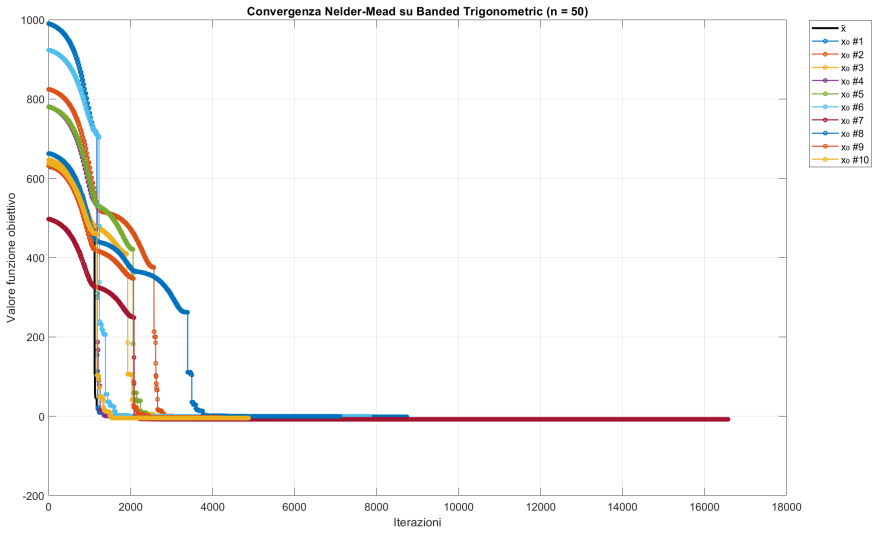


Figure 47: Convergence of Nelder-Mead on the Banded Trigonometric function with $n = 50$.

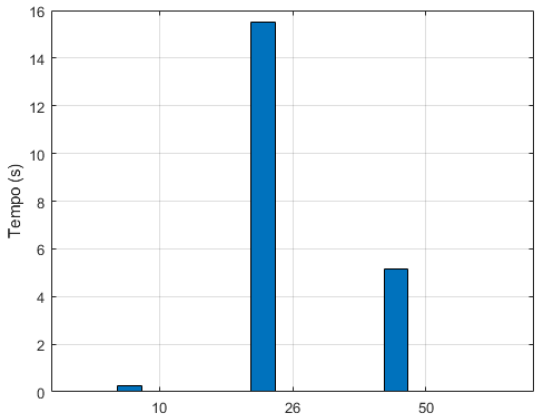


Figure 48: Computation time of Nelder-Mead on the Banded Trigonometric function for $n \in \{10, 26, 50\}$.

6 Conclusions

In this project, we implemented and compared the performance of two numerical optimization methods—Modified Newton and Nelder–Mead—on three benchmark unconstrained problems:

1. Extended Rosenbrock function,
2. Generalized Broyden Tridiagonal function,
3. Banded Trigonometric function.

The Modified Newton method consistently delivered superior results across all tested functions and dimensions. When exact derivatives were available, the method achieved rapid convergence—often in fewer than 10 iterations—and scaled efficiently even in high-dimensional settings (e.g. $n = 10^5$), thanks to sparse linear algebra and preconditioning techniques. Finite difference approximations, although more computationally expensive and slightly less accurate, proved to be reliable when properly tuned, particularly with smaller step sizes (e.g. $h \leq 10^{-6}$).

In contrast, the Nelder–Mead method showed acceptable performance only in low-dimensional cases. While easy to implement and free of derivative requirements, its convergence became erratic and inefficient as the dimension increased. The algorithm often failed to reach low objective values, especially in problems with ill-conditioned or structured landscapes, such as the Banded Trigonometric function.

Overall, this study highlights the importance of leveraging second-order information and exploiting problem structure—such as sparsity or tridiagonality—for scalable and reliable optimization. Derivative-free methods may still be useful in small-scale or noisy settings, but for large, structured problems, Newton-type algorithms remain the preferred choice.

Appendix: MATLAB Codes

In this appendix we report the commented MATLAB scripts and functions developed for the assignment. All codes are provided with inline comments to clarify each computational step and choice. The file and function names correspond to those used in the computational experiments described in the main text.

Modified Newton Method on Rosenbrock function 2D

```
function [x_min, f_min, iter, min_history] = modified_newton(f, grad_f, hess_f, x0, tol, max_iter,name)
% Implementation of Modified Newton Method with Armijo Backtracking Line Search

% Inputs:
%   f: Function handle of the objective function
%   grad_f: Function handle of the gradient
%   hess_f: Function handle of the Hessian
%   x0: Initial guess
%   tol: Convergence tolerance
%   max_iter: Max number of iterations

% Outputs:
%   x_min: Point that minimizes f
%   f_min: Minimum value of f
%   iter: Number of iterations performed
%   min_history: Sequence of function values (for plot)

x = x0;
min_history = zeros(1, max_iter);

rho = 0.5;      % Reduction factor for backtracking
c = 1e-4;       % Armijo condition constant
iter = 1;

while iter <= max_iter
g = grad_f(x);
H = hess_f(x);

% Store current function value
min_history(iter) = f(x);

% Modified Hessian (ensure positive definiteness)
%tao = max(0, sqrt(1) - min(eig(H)));
```

```
34          %H_mod = H + tao * eye(n);  % Adds diagonal damping
                if needed
35
36          [L, ~] = alg63_cholesky(H,50);
37
38          % Compute Newton direction
39          %p = -H_mod \ g;
40
41          p = - L'\(L\g);
42
43          % Backtracking line search (Armijo rule)
44          alpha = 1;
45          f_curr = f(x);
46          max_backtracking_iter = 40;
47          backtracking_iter = 0;
48
49          if name == "bt" || name == "gb"
50
51          p=[0;p;0];
52          g=[0;g;0];
53
54          end
55
56          while f(x + alpha * p) > f_curr + c * alpha * g' * p
                && backtracking_iter < max_backtracking_iter
57          alpha = rho * alpha;
58          backtracking_iter = backtracking_iter + 1;
59          end
60
61          % Update iterate
62          x = x + alpha * p;
63
64          f_old = f_curr;
65          f_curr = f(x);
66          g      = grad_f(x);
67
68          % Check stopping criterion
69          if norm(g,inf) <= tol
70          break
71          end
72
73          if abs(f_curr - f_old) <= tol*max(1,abs(f_old))
74          break
75          end
76
77          if f(x) <= tol
78          break;
79          end
80
```

```
81          iter = iter + 1;
82          end
83
84          % Output final results
85          x_min = x;
86          f_min = f(x);
87          min_history = min_history(1:iter);
88
89          end
90
91          function [L, tau] = alg63_cholesky(A, maxIter)
92
93          n    = size(A,1);
94
95          % Step 1: beta = norm(A, 'fro');
96          beta = norm(A, 'fro');
97
98          % % Step 2: tau0
99          % if min(diag(A)) > 0
100         %     tau = 0;
101         % else
102         %     tau = min(beta/2, 1e-1);   % non partire oltre
                0.1
103         % end
104
105         % Step 2: tau iniziale
106         tau0 = 1e-3;
107         tau  = 0;
108
109         I = speye(n);
110
111         % Step 3
112         for k = 0:maxIter
113         [L,flag] = chol(A + tau*I,'lower');   % L*L' = A+tau
                I
114         if flag == 0                          % OK
115         return
116         end
117         %tau = max(2*tau, beta/2);
118         if tau == 0
119         tau = max(tau0, min(beta/2, 1e-1));
120         else
121         tau = 2 * tau;
122         end
123
124         end
125
126         error('Alg63: fallito dopo %d tentativi', maxIter);
127         end
```

```
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % PARTE 2 - TEST DELL'ALGORITMO SULLA FUNZIONE DI ROSENBROCK
    % Test con due punti iniziali richiesti dall'assignment:
    % [1.2, 1.2] e [-1.2, 1.0]
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % Test del Metodo di Newton modificato sulla funzione di Rosenbrock
    clc; clear; close all;

    % Funzione di Rosenbrock e sue derivate
    rosenbrock = @(x) 100*(x(2) - x(1)^2)^2 + (1 - x(1))^2;
    grad_rosen = @(x) [-400*(x(2) - x(1)^2)*x(1) - 2*(1 - x(1));
    200*(x(2) - x(1)^2)];

    hess_rosen = @(x) [1200*x(1)^2 - 400*x(2) + 2, -400*x(1);
    -400*x(1), 200];

    % Punti iniziali
    x0_1 = [1.2, 1.2];
    x0_2 = [-1.2, 1.0];

    % Parametri
    tol = 1e-6;
    max_iter = 100;

    % Esecuzione algoritmo
    [x_min1, f_min1, iter1, hist1] = modified_newton(rosenbrock, grad_rosen, hess_rosen, x0_1, tol, max_iter,"rn");
    [x_min2, f_min2, iter2, hist2] = modified_newton(rosenbrock, grad_rosen, hess_rosen, x0_2, tol, max_iter,"rn");

    % Stampa risultati
    fprintf('\n==============================================\n')
```

```
            ;
162         fprintf(' TEST SU ROSENBROCK - METODO NEWTON
                MODIFICATO\n');
163         fprintf('
                ================================================\n\n
                ');
164
165         fprintf('Starting point: [1.2, 1.2]\n');
166         fprintf('Minimum found: [%f, %f]\n', x_min1(1),
                x_min1(2));
167         fprintf('Function value: %f\n', f_min1);
168         fprintf('Iterations: %d\n\n', iter1);
169
170         fprintf('Starting point: [-1.2, 1.0]\n');
171         fprintf('Minimum found: [%f, %f]\n', x_min2(1),
                x_min2(2));
172         fprintf('Function value: %f\n', f_min2);
173         fprintf('Iterations: %d\n\n', iter2);
174
175         % Grafico
176         figure('Units', 'normalized', 'Position', [0.2 0.2
                0.6 0.6]);  % finestra grande
177         plot(1:iter1, hist1, '-o', 'DisplayName', '[1.2, 1.2]
                ');
178         hold on;
179         plot(1:iter2, hist2, '-x', 'DisplayName', '[-1.2,
                1.0]');
180         xlabel('Numero di Iterazioni');
181         ylabel('Valore della Funzione Obiettivo');
182         title('Convergenza Metodo Newton Modificato sulla
                Funzione di Rosenbrock');
183         legend show;
184         grid on;
```

Listing 1: Modified Newton Method with Armijo Backtracking Line Search

Modified Newton Method on Extended Rosenbrock

```
clc; clear; close all;
format long e

function [x_min, f_min, iter, min_history, grad_norm,
    e_rate] = modified_newton(f,grad_f,hess_f,x0,tol,
   max_iter,fd,h,type)

x = x0;
n = length(x);
min_history = zeros(1, max_iter);
e_rate = zeros(n,4);

rho = 0.5;      % Reduction factor for backtracking
c = 1e-4;       % Armijo condition constant
iter = 1;

while iter <= max_iter

if fd
g = grad_f(x,h,type);
H = hess_f(x,h,type);
else
g = grad_f(x);
H = hess_f(x);
end

% Store current function value
min_history(iter) = f(x);

% Modified Hessian (ensure positive definiteness)
%tao = max(0, sqrt(1) - min(eig(H)));
%H_mod = H + tao * eye(n);  % Adds diagonal damping
    if needed

[L, ~] = alg63_cholesky(H,100);

% Compute Newton direction
%p = -H_mod \ g;

p = - L'\(L\g);

% Backtracking line search (Armijo rule)
alpha = 1;
f_curr = f(x);
max_backtracking_iter = 40;
backtracking_iter = 0;

```

```
45          while f(x + alpha * p) > f_curr + c * alpha * g' * p
               && backtracking_iter < max_backtracking_iter
46          alpha = rho * alpha;
47          backtracking_iter = backtracking_iter + 1;
48          end
49
50          % Update iterate
51          x = x + alpha * p;
52
53          f_old = f_curr;
54          f_curr = f(x);
55
56          if iter < 5
57          e_rate(:,iter) = x;
58          end
59
60          if iter >= 5
61          e_rate = [e_rate(:, 2:4), x];
62          end
63
64          if fd
65          g = grad_f(x,h,type);
66          else
67          g = grad_f(x);
68          end
69
70          % Check stopping criterion
71          if norm(g,inf) <= tol
72          break
73          end
74
75          if abs(f_curr - f_old) <= tol*max(1,abs(f_old))
76          break
77          end
78
79          if f(x) <= tol
80          break;
81          end
82
83          iter = iter + 1;
84          end
85
86          % Output final results
87          x_min = x;
88          f_min = f(x);
89          min_history = min_history(1:iter);
90          grad_norm = norm(g,inf);
91
92          end
```

```
93
94          function [L, tau] = alg63_cholesky(A, maxIter)
95
96          n     = size(A,1);
97
98          % Step 1: beta = norm(A, 'fro')
99          beta = norm(A, 'fro');
100
101         % % Step 2: tau0
102         % if min(diag(A)) > 0
103         %      tau = 0;
104         % else
105         %      tau = min(beta/2, 1e-1);    % non partire oltre
                0.1
106         % end
107
108         % Step 2: tau iniziale
109         tau0 = 1e-3;
110         tau  = 0;
111
112         I = speye(n);
113
114         % Step 3
115         for k = 0:maxIter
116         [L,flag] = chol(A + tau*I,'lower');    % L*L' = A+tau
                I
117         if flag == 0                           %  OK
118         return
119         end
120         %tau = max(2*tau, beta/2);
121         if tau == 0
122         tau = max(tau0, min(beta/2, 1e-1));
123         else
124         tau = 2 * tau;
125         end
126
127         end
128
129         error('Alg63: fallito dopo %d tentativi', maxIter);
130         end
131
132         function xbar = initial_solution_er(n)
133
134         xbar = ones(n, 1);
135         xbar(1:2:end) = -1.2;
136
137         end
138
139         function q = compute_ecr(X)
```

```
d = zeros(1,3);
for k = 1:3
d(k) = norm(X(:,k+1) - X(:, k), 2);
end

q = log(d(3) / d(2)) / log(d(2) / d(1));
end


function X0 = generate_initial_points(x_bar, num_points)

n = length(x_bar);
X0 = repmat(x_bar, 1, num_points) + 2*rand(n, num_points) - 1;
end


function esito = is_success(f_min, tol_success)

if f_min > 0 && f_min < tol_success
esito = 1;
elseif f_min < 0 && f_min > -tol_success
esito = 1;
else
esito = 0;
end
end

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%      TEST DELL'ALGORITMO SULLA FUNZIONE DI EXTENDED ROSENBROCK
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Exact gradient
function [gradf] = grad_extended_rosenbrock(x)
n = length(x);
gradf = zeros(n,1);
gradf(1:2:n-1) = 200*x(1:2:n-1).^3 - 200*x(2:2:n).*x(1:2:n-1) + x(1:2:n-1) - 1;
gradf(2:2:n) = -100*(x(1:2:n-1).^2 - x(2:2:n));
end

% Exact hessian
```

```
181         function [Hessf] = extended_rosenbrock_Hess(x)
182         n = length(x);
183         diags = zeros(n,3);
184         diags(1:2:n-1, 2) = 200*(3*x(1:2:n-1).^2 - x(2:2:n))
                + 1;
185         diags(2:2:n, 2) = 100;
186         diags(1:2:n-1, 1) = -200*x(1:2:n-1);
187         diags(2:2:n-2, 1) = 0;
188         diags(2:n, 3) = diags(1:n-1, 1);
189         Hessf = spdiags(diags, -1:1, n, n);
190         end
191
192         % Finite differences gradient
193         function grad_fd = extended_rosenbrock_gradf_fd(x, h,
                type)
194         n = length(x);
195         if type
196         hs = h*abs(x);
197         else
198         hs = h*ones(n, 1);
199         end
200
201         grad_fd = zeros(n, 1);
202         grad_fd(1:2:n) = 2*x(1:2:n).*hs(1:2:n) - 2*hs(1:2:n)
                + 400*x(1:2:n).^3.*hs(1:2:n) + 400*x(1:2:n).*hs
                (1:2:n).^3 - 400*x(1:2:n).*x(2:2:n).*hs(1:2:n);
203         grad_fd(2:2:n) = -200.*hs(2:2:n).*x(1:2:n).^2 + 200*
                hs(2:2:n).*x(2:2:n);
204         grad_fd = grad_fd ./ (2*hs);
205         end
206
207         % Finite differences hessian
208         function H = extended_rosenbrock_Hessf_fd(x, h, type)
209         n = length(x);
210         diag = zeros(n,1); % diagonal elements
211         codiag = zeros(n-1,0); % codiagonal elements
212
213         % type of increment h
214         if type
215         hs = h*abs(x);
216         else
217         hs = h*ones(n, 1);
218         end
219
220         % construction of the tridiagonal matrix
221         diag(1:2:n) = 1200*hs(1:2:n).*x(1:2:n) - 200.*x(2:2:n
                ) + 1 + 700*hs(1:2:n).^2 + 600*x(1:2:n).^2;
222         diag(2:2:n) = 100;
223         codiag(1:2:n) = -100*hs(1:2:n) - 200*x(1:2:n);
```

```
224
225         D = sparse(1:n,1:n,diag,n,n);
226         E = sparse(2:n,1:n-1,codiag,n,n);
227
228         H = E + D + E';
229         end
230
231         matricole = [295706, 302689]; % ora 295706 e
                diventato 349152
232         rng(min(matricole));
233
234         max_iter = 5000;  % Maximum number of iterations
235         tol = 1e-6;
236         num_points = 10;
237         k = 2:2:12;
238         h = power(10,-k); % increment of the finite
                differences
239         n_NewtonModified = [1000, 10000, 100000];
240         time_dim  = zeros(3);
241         a = 1;
242
243         extended_rosenbrock = @(x) 0.5*sum([10*(x(1:2:end).^2
                - x(2:2:end)); x(1:2:end-1)-1].^2);
244
245         t_total = tic;
246
247         for j=n_NewtonModified
248
249         t0 = tic;
250
251         fprintf('\n
                ================================================\
                n');
252         fprintf(' TEST SU EXTENDED ROSENBROCK IN DIMENSIONE %
                d \n', j);
253         fprintf('
                ================================================\
                n\n');
254
255         x_bar = initial_solution_er(j);
256
257         X0 = generate_initial_points(x_bar, num_points);
258
259         fd = 1;
260
261         if fd == 1
262         grad_f = @extended_rosenbrock_gradf_fd;
263         hess_f = @extended_rosenbrock_Hessf_fd;
264         end
```

```
265
266         if fd == 1
267         fprintf('\n
                ================================================\
                n');
268         fprintf(' TEST SU DERIVATE APPROSSIMATE CON
                DIFFERENZE FINITE \n');
269         fprintf('
                ================================================\
                n\n');
270
271         for increment = h
272
273         fprintf('\n
                ------------------------------------------------
                ');
274         fprintf('\nDefault increment h = %d \n',increment);
275         fprintf('
                ------------------------------------------------\
                n');
276
277         % === TEST SU x_bar === %
278         fprintf('\n--- TEST SU VALORE X_BAR ---\n');
279         tic;
280         [~, f_min, iter_bar, min_hist_bar, grad_norm_bar,
                e_rate_bar] = modified_newton(extended_rosenbrock,
                grad_f,hess_f,x_bar,tol,max_iter,fd,increment,0);
281         t = toc;
282         fprintf('f_min = %.6f\n | iter = %d | tempo = %.3fs\n
                 | grad_norm = %.6f\n', f_min, iter_bar, t,
                grad_norm_bar);
283         rho = compute_ecr(e_rate_bar);
284         fprintf('rho      %.4f\n\n', rho);
285
286         % === TEST SU 10 PUNTI CASUALI === %
287         fprintf('\n--- TEST SU 10 PUNTI CASUALI ---\n');
288         min_hist_all = cell(num_points, 1);
289         successi = 0;
290         for i = 1:num_points
291         x0 = X0(:,i);
292         fprintf('\n--- Test %d (x0 #%d) ---\n', i, i);
293         tic;
294         [~, f_min, iter, min_hist, grad_norm, e_rate] =
                modified_newton(extended_rosenbrock,grad_f,hess_f,
                x0,tol,max_iter,fd,increment,0);
295         t = toc;
296         fprintf('f_min = %.6f\n | iter = %d | tempo = %.3fs\n
                 | grad_norm = %.6f\n', f_min, iter, t, grad_norm)
                ;
```

```
297         rho = compute_ecr(e_rate);
298         fprintf('rho      %.4f\n\n', rho);
299         min_hist_all{i} = min_hist;
300         successi = successi + is_success(f_min, 0.5);
301         end
302         fprintf('\nSuccessi: %d su %d\n', successi,
                num_points);
303
304         % Absolute value increment
305         fprintf('\n
                ----------------------------------------------------
                ');
306         fprintf('\nAbsolute value increment h = %d*|x| \n',
                increment);
307         fprintf('
                ----------------------------------------------------\
                n');
308
309         % === TEST SU x_bar === %
310         fprintf('\n--- TEST SU VALORE X_BAR ---\n');
311         tic;
312         [x_min, f_min, iter_bar_abs, min_hist_bar_abs,
                grad_norm, e_rate_abs] = modified_newton(
                extended_rosenbrock,grad_f,hess_f,x_bar,tol,
                max_iter,fd,increment,1);
313         t = toc;
314         fprintf('f_min = %.6f\n | iter = %d | tempo = %.3fs\n
                 | grad_norm = %.6f\n', f_min, iter_bar_abs, t,
                grad_norm);
315         rho = compute_ecr(e_rate_abs);
316         fprintf('rho      %.4f\n\n', rho);
317
318         % === TEST SU 10 PUNTI CASUALI === %
319         fprintf('\n--- TEST SU 10 PUNTI CASUALI ---\n');
320         min_hist_all_abs = cell(num_points, 1);
321         successi = 0;
322         for i = 1:num_points
323         x0 = X0(:,i);
324         fprintf('\n--- Test %d (x0 #%d) ---\n', i, i);
325         tic;
326         [x_min, f_min, iter, min_hist, grad_norm, e_rate] =
                modified_newton(extended_rosenbrock,grad_f,hess_f,
                x0,tol,max_iter,fd,increment,1);
327         t = toc;
328         fprintf('f_min = %.6f\n | iter = %d | tempo = %.3fs\n
                 | grad_norm = %.6f\n', f_min, iter, t, grad_norm)
                ;
329         rho = compute_ecr(e_rate);
330         fprintf('rho      %.4f\n\n', rho);
```

```
331         min_hist_all_abs{i} = min_hist;
332         successi = successi + is_success(f_min, 0.5);
333         end
334         fprintf('\nSuccessi: %d su %d\n', successi,
                num_points);
335
336         % === FIGURE ===
337         fig = figure('Units','normalized','Position',[0.12
                0.12 0.78 0.62]);
338         tl = tiledlayout(fig,1,2,'TileSpacing','compact','
                Padding','compact');
339         colors = lines(num_points);
340
341         % -- LEFT: h
342         nexttile(tl,1); hold on;
343         plot(1:iter_bar, min_hist_bar, '-o', 'LineWidth',
                1.8, 'Color', 'k', 'DisplayName', 'x  ');
344         for i = 1:num_points
345         mh = min_hist_all{i};
346         plot(1:length(mh), mh, '-o', 'LineWidth', 1.2, '
                MarkerSize', 5, 'Color', colors(i,:), 'DisplayName
                ', sprintf(' x   #%d', i));
347         end
348         title(sprintf('h = %.1e', increment), 'FontSize', 12)
                ;
349         xlabel('Iterazioni'); ylabel('f(x_k)');
350         set(gca, 'YScale', 'log'); grid on; box on; set(gca,
                'FontSize', 11);
351
352         % -- RIGHT: h * |x|
353         nexttile(tl,2); hold on;
354         plot(1:iter_bar_abs, min_hist_bar_abs, '-o', '
                LineWidth', 1.8, 'Color', 'k', 'DisplayName', 'x
                ');
355         for i = 1:num_points
356         mh = min_hist_all_abs{i};
357         plot(1:length(mh), mh, '-o', 'LineWidth', 1.2, '
                MarkerSize', 5, 'Color', colors(i,:), 'DisplayName
                ', sprintf(' x   #%d', i));
358         end
359         title(sprintf('h = %.1 e  |x|', increment), 'FontSize'
                , 12);
360         xlabel('Iterazioni'); ylabel('f(x_k)');
361         set(gca, 'YScale', 'log'); grid on; box on; set(gca,
                'FontSize', 11);
362
363         title(tl, sprintf('Convergenza Metodo Newton
                Modificato     n = %d', j), 'FontSize', 14);
364         legend('show', 'Location', 'eastoutside');
```

```
365
366         end
367         end
368
369         fd = 0;
370
371         if fd == 0
372         grad_f = @grad_extended_rosenbrock;
373         hess_f = @extended_rosenbrock_Hess;
374         end
375
376         if fd == 0
377
378         fprintf('\n
                ================================================\
                n');
379         fprintf(' TEST SU DERIVATE ESATTE \n');
380         fprintf('
                ================================================\
                n\n');
381
382         x_bar = initial_solution_er(j);
383
384         X0 = generate_initial_points(x_bar, num_points);
385
386         % === TEST SU x_bar ===
387         fprintf('\n--- TEST SU VALORE X_BAR ---\n');
388         tic;
389         [x_min, f_min, iter_bar, min_hist_bar, grad_norm,
                e_rate] = modified_newton(extended_rosenbrock,
                grad_f,hess_f,x_bar,tol,max_iter,fd,[],[]);
390         t = toc;
391         fprintf('f_min = %.6f\n | iter = %d | tempo = %.3fs\n
                 | grad_norm = %.6f\n', f_min, iter_bar, t,
                grad_norm);
392         rho = compute_ecr(e_rate);
393         fprintf('rho      %.4f\n', rho);
394
395         % === TEST SU 10 PUNTI CASUALI ===
396         min_hist_all = cell(num_points, 1);
397         successi = 0;
398         for i = 1:num_points
399         x0 = X0(:,i);
400         fprintf('\n--- Test %d (x0 #%d) ---\n', i, i);
401         tic;
402         [x_min, f_min, iter, min_hist, grad_norm, e_rate] =
                modified_newton(extended_rosenbrock,grad_f,hess_f,
                x0,tol,max_iter,fd,[],[]);
403         t = toc;
```

```
404         fprintf('f_min = %.6f\n | iter = %d | tempo = %.3fs\n
                | grad_norm = %.6f\n', f_min, iter, t, grad_norm)
                ;
405         rho = compute_ecr(e_rate); % se f* = 0
406         fprintf('rho      %.4f\n', rho);
407         min_hist_all{i} = min_hist;
408         successi = successi + is_success(f_min, 0.5);
409         end
410
411         fprintf('\nSuccessi: %d su %d\n', successi,
                num_points);
412
413         % === PLOT CONVERGENZA ===
414         figure('Units', 'normalized', 'Position', [0.2 0.2
                0.6 0.6]);
415         hold on;
416
417         % Plot x
418         plot(1:iter_bar, min_hist_bar, '-o', 'LineWidth',
                1.8, ...
419         'DisplayName', 'x  ', 'Color', 'k');
420
421         % Colori per i 10 test random
422         colors = lines(num_points);
423
424         % Plot dei 10 punti iniziali random
425         for i = 1:num_points
426         mh = min_hist_all{i};
427         plot(1:length(mh), mh, '-o', ...
428         'LineWidth', 1.2, ...
429         'MarkerSize', 5, ...
430         'Color', colors(i,:), ...
431         'DisplayName', sprintf(' x    #%d', i));
432         end
433
434         % Titoli e assi
435         xlabel('Iterazioni', 'FontSize', 12);
436         ylabel('Valore funzione obiettivo', 'FontSize', 12);
437         title(sprintf('Convergenza Metodo su Extended
                Rosenbrock Esatto (n = %d)', j), 'FontSize', 14);
438
439         % STYLE
440         legend('show', 'Location', 'eastoutside');
441         grid on;
442         set(gca, 'YScale', 'log');
443
444         box on;
445         set(gca, 'FontSize', 12);
446         hold off;
```

```
447
448         end
449
450         time_dim(a) = toc(t0);
451         fprintf('\nTempo (incl. plotting) per n = %-7d  :
                %.2f  s\n', j, time_dim(a));
452         a = a + 1;
453         end
454
455         % ---------------------- TEMPO TOTALE SCRIPT
                ----------------------
456         fprintf('\n
                ==================================================\
                n');
457         fprintf(' TABELLA TEMPISTICHE ALGORITMO EXTENDED
                ROSENBROCK \n');
458         fprintf('
                ==================================================\
                n\n');
459
460         time_total = toc(t_total);
461
462         fprintf('\nTempo (incl. plotting) per n = %-7d  :
                %.2f  s\n', ...
463         n_NewtonModified(1), time_dim(1));
464         fprintf('\nTempo (incl. plotting) per n = %-7d  :
                %.2f  s\n', ...
465         n_NewtonModified(2), time_dim(2));
466         fprintf('\nTempo (incl. plotting) per n = %-7d  :
                %.2f  s\n', ...
467         n_NewtonModified(3), time_dim(3));
468         fprintf('\nTempo TOTALE (tutte le dimensioni) :  %.2f
                 s\n', time_total);
469
470         %---  bar chart  ---
471         figure;
472         bar(categorical(string(n_NewtonModified)), time_dim);
473         ylabel('Tempo (s)'); grid on;
```

Listing 2: Full script: Modified Newton method on Extended Rosenbrock

Modified Newton Method on Generalized Broyden

```
clc; clear; close all;
format long e

function [x_min, f_min, iter, min_history, grad_norm,
    e_rate] = modified_newton(f,grad_f,hess_f,x0,tol,
   max_iter,fd,h,type)

x = x0;
n = length(x);
min_history = zeros(1, max_iter);
e_rate = zeros(n,4);

rho = 0.5;      % Reduction factor for backtracking
c = 1e-4;       % Armijo condition constant
iter = 1;

while iter <= max_iter

if fd
g = grad_f(x,h,type);
H = hess_f(x,h,type);
else
g = grad_f(x);
H = hess_f(x);
end

% Store current function value
min_history(iter) = f(x);

% Modified Hessian (ensure positive definiteness)
%tao = max(0, sqrt(1) - min(eig(H)));
%H_mod = H + tao * eye(n);  % Adds diagonal damping
    if needed

[L, ~] = alg63_cholesky(H,100);

% Compute Newton direction
%p = -H_mod \ g;

p = - L'\(L\g);

if fd == 0
p=[0;p;0];
g=[0;g;0];
end

% Backtracking line search (Armijo rule)
```

```
45          alpha = 1;
46          f_curr = f(x);
47          max_backtracking_iter = 40;
48          backtracking_iter = 0;
49
50          while f(x + alpha * p) > f_curr + c * alpha * g' * p
               && backtracking_iter < max_backtracking_iter
51          alpha = rho * alpha;
52          backtracking_iter = backtracking_iter + 1;
53          end
54
55          % Update iterate
56          x = x + alpha * p;
57
58          f_old = f_curr;
59          f_curr = f(x);
60
61          if iter < 5
62          e_rate(:,iter) = x;
63          end
64
65          if iter >= 5
66          e_rate = [e_rate(:, 2:4), x];
67          end
68
69          if fd
70          g = grad_f(x,h,type);
71          else
72          g = grad_f(x);
73          end
74          % Check stopping criterion
75          if norm(g,inf) <= tol
76          break
77          end
78
79          if abs(f_curr - f_old) <= tol*max(1,abs(f_old))
80          break
81          end
82
83          if f(x) <= tol
84          break;
85          end
86
87          iter = iter + 1;
88          end
89
90          % Output final results
91          x_min = x;
92          f_min = f(x);
```

```
93          min_history = min_history(1:iter);
94          grad_norm = norm(g,inf);
95
96          end
97
98          function [L, tau] = alg63_cholesky(A, maxIter)
99
100         n     = size(A,1);
101
102         % Step 1:     = ||A||_F
103         beta = norm(A, 'fro');
104
105         % % Step 2:  0
106         % if min(diag(A)) > 0
107         %     tau = 0;
108         % else
109         %     tau = min(beta/2, 1e-1);   % non partire oltre
                0.1
110         % end
111
112         % Step 2:    iniziale
113         tau0 = 1e-3;
114         tau  = 0;
115
116         I = speye(n);
117
118         % Step 3
119         for k = 0:maxIter
120         [L,flag] = chol(A + tau*I,'lower');    % L*L' = A+ I
121         if flag == 0                           % OK
122         return
123         end
124         %tau = max(2*tau, beta/2);
125         if tau == 0
126         tau = max(tau0, min(beta/2, 1e-1));
127         else
128         tau = 2 * tau;
129         end
130
131         end
132
133         error('Alg63: fallito dopo %d tentativi', maxIter);
134         end
135
136         function xbar = initial_solution_gb(n)
137
138         xbar = -ones(n+2, 1);
139         xbar(1) = 0;
140         xbar(n+2) = 0;
```

```
end

function q = compute_ecr(X)

d = zeros(1,3);
for k = 1:3
d(k) = norm(X(:,k+1) - X(:, k), 2);
end

q = log(d(3) / d(2)) / log(d(2) / d(1));
end

function X0 = generate_initial_points(x_bar, num_points)
n = length(x_bar);
X0 = repmat(x_bar, 1, num_points) + 2*rand(n, num_points) - 1;
end

function esito = is_success(f_min, tol_success)

if f_min > 0 && f_min < tol_success
esito = 1;
elseif f_min < 0 && f_min > -tol_success
esito = 1;
else
esito = 0;
end
end

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       TEST DELL'ALGORITMO SULLA FUNZIONE GENERALIZED BROYDEN
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Exact gradient
grad_generalized_broyden = @(x) [(3-4*x(2))*((3-2*x(2))*x(2)+1-x(1)-x(3))-((3-2*x(3))*x(3)-x(2)-x(4)+1);
(3 - 4*x(3:end-2)) .* ((3 - 2*x(3:end-2)) .* x(3:end-2) + 1 - x(2:end-3) - x(4:end-1) ) - ((3 - 2*x(2:end-3)) .* x(2:end-3) ...
+ 1 - x(1:end-4) - x(3:end-2) ) - ((3 - 2*x(4:end-1)) .* x(4:end-1) + 1 - x(3:end-2) - x(5:end) );
```

```
178         (3-4*x(length(x)-1))*((3-2*x(length(x)-1))*x(length(x
                )-1)+1-x(length(x)-2)-x(length(x)))-((3-2*x(length
                (x)-2))*x(length(x)-2)-x(length(x)-3)-x(length(x)
                -1)+1)];
179
180         % Exact hessian
181         function H = generalized_broyden_Hessf(x)
182
183         n_full = length(x);
184         n = n_full - 2;
185         x_in = x(2:end-1);
186
187         fk = (3 - 2*x_in) .* x_in + 1 - x(1:end-2) - x(3:end)
                ;
188
189         d0 = zeros(n, 1);    % diagonale principale
190         d1 = zeros(n-1,1);   % codiagonali  1
191         d2 = ones(n-2,1);    % codiagonali  2  (costanti = 1)
192
193         d0(1)   = (3 - 4*x(2))^2 - 4*fk(1) + 1;
194         d0(end) = (3 - 4*x(end-1))^2 - 4*fk(end) + 1;
195         d0(2:end-1) = (3 - 4*x(3:end-2)).^2 - 4*fk(2:end-1) +
                2;
196
197         d1(:) = 4 * x(2:end-2) + 4 * x(3:end-1) - 6;
198
199         H = spdiags([ ...
200         [d2; 0; 0], ...    % diagonale -2
201         [d1; 0], ...    % diagonale -1
202         d0, ...            % diagonale  0
203         [0; d1], ...    % diagonale +1
204         [0; 0; d2] ...     % diagonale +2
205         ], -2:2, n, n);
206
207         end
208
209         % Finite differences gradient
210         function grad_fd = generalized_broyden_gradf_fd(x, h,
                type)
211         n = length(x);
212         if type
213         hs = h*abs(x);
214         else
215         hs = h*ones(n, 1);
216         end
217         xm2 = [0; 0; x(1:end-2)];
218         xm1 = [0; x(1:end-1)];
219         xp1 = [x(2:end); 0];
220         xp2 = [x(3:end); 0; 0];
```

```
221         grad_fd = 2*xm1.^2 + 4*xm1.*x - 6*xm1 + 8*x.^3 - 18*x
                .^2 + 4*x.*xp1 + 8*x.*hs.^2 + 7.*x ...
222         + 2*xp1.^2 - 6*xp1 - 6*hs.^2 + xm2 + xp2 + 1;
223         grad_fd(1) =  6*x(1) - 6*x(2) + x(3)+ 4*x(1)*x(2) +
                8*x(1)*hs(1)^2 ...
224         - 18*x(1)^2 + 8*x(1)^3 + 2*x(2)^2 - 6*hs(1)^2 + 2;
225         grad_fd(n) = x(n-2) - 6*x(n-1) + 6*x(n) + 4*x(n-1)*x(
                n) + 8*x(n)*hs(n)^2 ...
226         + 2*x(n-1)^2 - 18*x(n)^2 + 8*x(n)^3 - 6*hs(n)^2 + 2;
227         end
228
229         % Finite differences hessian
230         function H = generalized_broyden_Hessf_fd(x, h, type)
231         n = length(x);
232         if type
233         hs = h*abs(x);
234         else
235         hs = h*ones(n, 1);
236         end
237
238         diag = zeros(n,1);
239         codiag1 = zeros(n-1,1);
240         codiag2 = zeros(n-2,1);
241         xm1 = [0; x(1:n-1)];
242         xp1 = [x(2:n); 0];
243
244         % Diagonal
245         diag(1:n) = 24*x.^2 + 48*x.*hs - 36*x + 28*hs.^2 -
                36*hs + 4*xm1 + 4*xp1 + 7;
246         diag(1) = 24*x(1)^2 + 48*x(1)*hs(1) - 36*x(1) + 28*hs
                (1)^2 - 36*hs(1) + 4*x(2) + 6;
247         diag(n) = 24*x(n)^2 + 48*x(n)*hs(n) - 36*x(n) + 28*hs
                (n)^2 - 36*hs(n) + 4*x(n-1) + 6;
248         % First codiagonal
249         codiag1(1:n-1) = 4*x(1:n-1) + 4*x(2:n) + 2*hs(1:n-1)
                + 2*hs(2:n) - 6;
250         % Second codiagonal
251         codiag2(1:n-2) = 1;
252
253         % construction of the pentadiagonal matrix
254         D = sparse(1:n,1:n,diag,n,n);
255         E = sparse(2:n,1:n-1,codiag1,n,n);
256         F = sparse(3:n,1:n-2,codiag2,n,n);
257         H = D + E + E' + F + F';
258         end
259
260         matricole = [295706, 302689]; %ora 295706
                diventato 349152
261         rng(min(matricole));
```

```
262
263         max_iter = 5000;  % Maximum number of iterations
264         tol = 1e-6;
265         num_points = 10;
266         fd = 1;
267         k = 2:2:12;
268         h = power(10,-k); % increment of the finite
                differences
269         n_NewtonModified = [1000,10000, 100000];
270         time_dim  = zeros(3);
271         a = 1;
272
273         generalized_broyden = @(x) 0.5*sum(((3-2*x(2:end-1))
                .*x(2:end-1)+1-x(1:end-2)-x(3:end)).^2);
274
275         t_total = tic;
276
277         for j=n_NewtonModified
278
279         t0 = tic;
280
281         fprintf('\n
                ==================================================\
                n');
282         fprintf(' TEST SU GENERALIZED BROYDEN IN DIMENSIONE %
                d \n', j);
283         fprintf('
                ==================================================\
                n\n');
284
285         x_bar = initial_solution_gb(j);
286
287         X0 = generate_initial_points(x_bar, num_points);
288
289         fd = 1;
290
291         if fd == 1
292         grad_f = @generalized_broyden_gradf_fd;
293         hess_f = @generalized_broyden_Hessf_fd;
294         end
295
296         if fd == 1
297
298         fprintf('\n
                ==================================================\
                n');
299         fprintf(' TEST SU DERIVATE APPROSSIMATE CON
                DIFFERENZE FINITE \n');
```

```
300         fprintf('
                ==================================================\
                n\n');
301
302         for increment = h
303
304         fprintf('\n
                --------------------------------------------------
                ');
305         fprintf('\nDefault increment h = %d \n',increment);
306         fprintf('
                --------------------------------------------------\
                n');
307
308         % === TEST SU x_bar === %
309         fprintf('\n--- TEST SU VALORE X_BAR ---\n');
310         tic;
311         [~, f_min, iter_bar, min_hist_bar,grad_norm_bar,
                e_rate] = modified_newton(generalized_broyden,
                grad_f,hess_f,x_bar,tol,max_iter,fd,increment,0);
312         t = toc;
313         fprintf('f_min = %.6f\n | iter = %d | tempo = %.3fs\n
                | grad_norm = %.6f\n', f_min, iter_bar, t,
                grad_norm_bar);
314         rho = compute_ecr(e_rate);
315         fprintf('rho      %.4f\n\n', rho);
316
317         % === TEST SU 10 PUNTI CASUALI === %
318         fprintf('\n--- TEST SU 10 PUNTI CASUALI ---\n');
319         min_hist_all = cell(num_points, 1);
320         successi = 0;
321         for i = 1:num_points
322         x0 = X0(:,i);
323         fprintf('\n--- Test %d (x0 #%d) ---\n', i, i);
324         tic;
325         [~, f_min, iter, min_hist, grad_norm,e_rate] =
                modified_newton(generalized_broyden,grad_f,hess_f,
                x0,tol,max_iter,fd,increment,0);
326         t = toc;
327         fprintf('f_min = %.6f\n | iter = %d | tempo = %.3fs\n
                | grad_norm = %.6f\n', f_min, iter, t,grad_norm);
328         rho = compute_ecr(e_rate); % se f* = 0
329         fprintf('rho      %.4f\n\n', rho);
330         min_hist_all{i} = min_hist;
331         successi = successi + is_success(f_min, 0.5);
332         end
333         fprintf('\nSuccessi: %d su %d\n', successi,
                num_points);
334
```

```
335         % Absolute value increment
336         fprintf('\n
                ----------------------------------------------------
                ');
337         fprintf('\nAbsolute value increment h = %d*|x| \n',
                increment);
338         fprintf('
                ----------------------------------------------------\
                n');
339
340         % === TEST SU x_bar === %
341         fprintf('\n--- TEST SU VALORE X_BAR ---\n');
342         tic;
343         [x_min, f_min, iter_bar_abs, min_hist_bar_abs,
                grad_norm, e_rate] = modified_newton(
                generalized_broyden,grad_f,hess_f,x_bar,tol,
                max_iter,fd,increment,1);
344         t = toc;
345         fprintf('f_min = %.6f\n | iter = %d | tempo = %.3fs\n
                 | grad_norm = %.6f\n', f_min, iter_bar_abs, t,
                grad_norm);
346         rho = compute_ecr(e_rate); % se f* = 0
347         fprintf('rho      %.4f\n\n', rho);
348
349         % === TEST SU 10 PUNTI CASUALI === %
350         fprintf('\n--- TEST SU 10 PUNTI CASUALI ---\n');
351         min_hist_all_abs = cell(num_points, 1);
352         successi = 0;
353         for i = 1:num_points
354         x0 = X0(:,i);
355         fprintf('\n--- Test %d (x0 #%d) ---\n', i, i);
356         tic;
357         [x_min, f_min, iter, min_hist, grad_norm,e_rate] =
                modified_newton(generalized_broyden,grad_f,hess_f,
                x0,tol,max_iter,fd,increment,1);
358         t = toc;
359         fprintf('f_min = %.6f\n | iter = %d | tempo = %.3fs\n
                 | grad_norm = %.6f\n', f_min, iter, t,grad_norm);
360         rho = compute_ecr(e_rate); % se f* = 0
361         fprintf('rho      %.4f\n\n', rho);
362         min_hist_all_abs{i} = min_hist;
363         successi = successi + is_success(f_min, 0.5);
364         end
365         fprintf('\nSuccessi: %d su %d\n', successi,
                num_points);
366
367         % === FIGURE ===
368         fig = figure('Units','normalized','Position',[0.12
                0.12 0.78 0.62]);
```

```
369         tl = tiledlayout(fig,1,2,'TileSpacing','compact','
                Padding','compact');
370         colors = lines(num_points);
371
372         % -- LEFT: h fisso
373         nexttile(tl,1); hold on;
374         plot(1:iter_bar, min_hist_bar, '-o', 'LineWidth',
                1.8, 'Color', 'k', 'DisplayName', 'x  ');
375         for i = 1:num_points
376         mh = min_hist_all{i};
377         plot(1:length(mh), mh, '-o', 'LineWidth', 1.2, '
                MarkerSize', 5, 'Color', colors(i,:), 'DisplayName
                ', sprintf(' x   #%d', i));
378         end
379         title(sprintf('h = %.1e', increment), 'FontSize', 12)
                ;
380         xlabel('Iterazioni'); ylabel('f(x_k)');
381         set(gca, 'YScale', 'log'); grid on; box on; set(gca,
                'FontSize', 11);
382
383         % -- RIGHT: h * |x|
384         nexttile(tl,2); hold on;
385         plot(1:iter_bar_abs, min_hist_bar_abs, '-o', '
                LineWidth', 1.8, 'Color', 'k', 'DisplayName', 'x
                ');
386         for i = 1:num_points
387         mh = min_hist_all_abs{i};
388         plot(1:length(mh), mh, '-o', 'LineWidth', 1.2, '
                MarkerSize', 5, 'Color', colors(i,:), 'DisplayName
                ', sprintf(' x   #%d', i));
389         end
390         title(sprintf('h = %.1 e  |x|', increment), 'FontSize'
                , 12);
391         xlabel('Iterazioni'); ylabel('f(x_k)');
392         set(gca, 'YScale', 'log'); grid on; box on; set(gca,
                'FontSize', 11);
393
394         % -- Titolo generale e legenda unica
395         title(tl, sprintf('Convergenza Metodo Newton
                Modificato     n = %d', j), 'FontSize', 14);
396         legend('show', 'Location', 'eastoutside');
397
398         end
399         end
400
401         fd = 0;
402
403         if fd == 0
404         grad_f = grad_generalized_broyden;
```

```
405         hess_f = @generalized_broyden_Hessf;
406         end
407
408         if fd == 0
409
410         fprintf('\n
                ================================================\
                n');
411         fprintf(' TEST SU DERIVATE ESATTE \n');
412         fprintf('
                ================================================\
                n\n');
413
414         x_bar = initial_solution_gb(j);
415
416         X0 = generate_initial_points(x_bar, num_points);
417
418         % === TEST SU x_bar ===
419         fprintf('\n--- TEST SU VALORE X_BAR ---\n');
420         tic;
421         [x_min, f_min, iter_bar, min_hist_bar, grad_norm,
                e_rate] = modified_newton(generalized_broyden,
                grad_f,hess_f,x_bar,tol,max_iter,fd,[],[]);
422         t = toc;
423         fprintf('f_min = %.6f\n | iter = %d | tempo = %.3fs\n
                 | grad_norm = %.6f\n', f_min, iter_bar, t,
                grad_norm);
424         rho = compute_ecr(e_rate); % se f* = 0
425         fprintf('rho      %.4f\n', rho);
426
427         % === TEST SU 10 PUNTI CASUALI ===
428         min_hist_all = cell(num_points, 1);
429         successi = 0;
430         for i = 1:num_points
431         x0 = X0(:,i);
432         fprintf('\n--- Test %d (x0 #%d) ---\n', i, i);
433         tic;
434         [x_min, f_min, iter, min_hist, grad_norm,e_rate] =
                modified_newton(generalized_broyden,grad_f,hess_f,
                x0,tol,max_iter,fd,[],[]);
435         t = toc;
436         fprintf('f_min = %.6f\n | iter = %d | tempo = %.3fs\n
                 | grad_norm = %.6f\n', f_min, iter, t,grad_norm);
437         rho = compute_ecr(e_rate); % se f* = 0
438         fprintf('rho      %.4f\n', rho);
439         min_hist_all{i} = min_hist;
440         successi = successi + is_success(f_min, 0.5);
441         end
442
```

```
443         fprintf('\nSuccessi: %d su %d\n', successi,
                num_points);
444
445         % === PLOT CONVERGENZA ===
446         figure('Units', 'normalized', 'Position', [0.2 0.2
                0.6 0.6]);
447         hold on;
448
449         % Plot x
450         plot(1:iter_bar, min_hist_bar, '-o', 'LineWidth',
                1.8, ...
451         'DisplayName', 'x  ', 'Color', 'k');
452
453         % Colori per i 10 test random
454         colors = lines(num_points);
455
456         % Plot dei 10 punti iniziali random
457         for i = 1:num_points
458         mh = min_hist_all{i};
459         plot(1:length(mh), mh, '-o', ...
460         'LineWidth', 1.2, ...
461         'MarkerSize', 5, ...
462         'Color', colors(i,:), ...
463         'DisplayName', sprintf(' x   #%d', i));
464         end
465
466         % Titoli e assi
467         xlabel('Iterazioni', 'FontSize', 12);
468         ylabel('Valore funzione obiettivo', 'FontSize', 12);
469         title(sprintf('Convergenza Metodo su Generalized
                Broyden Esatto (n = %d)', j), 'FontSize', 14);
470
471         % Legenda e stile
472         legend('show', 'Location', 'eastoutside');
473         grid on;
474         set(gca, 'YScale', 'log');
475
476         box on;
477         set(gca, 'FontSize', 12);
478         hold off;
479
480         end
481
482         time_dim(a) = toc(t0);
483         fprintf('\nTempo (incl. plotting) per n = %-7d  :
                %.2f  s\n', j, time_dim(a));
484         a = a + 1;
485         end
486
```

```
487         % ---------------------- TEMPO TOTALE SCRIPT
                ----------------------
488         fprintf('\n
                ================================================\
                n');
489         fprintf(' TABELLA TEMPISTICHE ALGORITMO GENERALIZED
                ROSENBROCK \n');
490         fprintf('
                ================================================\
                n\n');
491
492         time_total = toc(t_total);
493
494         fprintf('\nTempo (incl. plotting) per n = %-7d  :
                %.2f  s\n', ...
495         n_NewtonModified(1), time_dim(1));
496         fprintf('\nTempo (incl. plotting) per n = %-7d  :
                %.2f  s\n', ...
497         n_NewtonModified(2), time_dim(2));
498         fprintf('\nTempo (incl. plotting) per n = %-7d  :
                %.2f  s\n', ...
499         n_NewtonModified(3), time_dim(3));
500         fprintf('\nTempo TOTALE (tutte le dimensioni) :  %.2f
                 s\n', time_total);
501
502         %---  bar chart  ---
503         figure;
504         bar(categorical(string(n_NewtonModified)), time_dim);
505         ylabel('Tempo (s)'); grid on;
```

Listing 3: Full script: Modified Newton method on Generalyzed Broyden

Modified Newton Method on Banded Trigonometric

```
clc; clear; close all;
format long e

function [x_min, f_min, iter, min_history, grad_norm,
    e_rate] = modified_newton(f,grad_f,hess_f,x0,tol,
   max_iter,fd,h,type)

x = x0;
n = length(x);
min_history = zeros(1, max_iter);
e_rate = zeros(n,4);

rho = 0.5;      % Reduction factor for backtracking
c = 1e-4;       % Armijo condition constant
iter = 1;

while iter < max_iter

if fd
g = grad_f(x,h,type);
H = hess_f(x,h,type);
else
g = grad_f(x);
H = hess_f(x);
end

% Store current function value
min_history(iter) = f(x);

% Modified Hessian (ensure positive definiteness)
%tao = max(0, sqrt(1) - min(eig(H)));
%H_mod = H + tao * eye(n);  % Adds diagonal damping
    if needed
% Compute Newton direction
% p = -H_mod \ g;

[~, tau] = alg63_cholesky(H,100);

H = H + tau*speye(size(H,1));
L = ichol(H);
[p, ~, ~, ~, ~] = pcg(H, -g, 1e-6, 50, L, L');

% beta = 10^-3;
% coeffient = 2;
% max_iter = 100;

```

```
44          % [B, tau] = chol_with_addition(H, beta, coeffient,
                max_iter);
45
46          % H = H + B;
47
48          % Compute Newton direction
49          % p = - L'\(L\g);
50
51          if fd == 0
52          p=[0;p;0];
53          g=[0;g;0];
54          end
55
56          % Backtracking line search (Armijo rule)
57          alpha = 1;
58          f_curr = f(x);
59          max_backtracking_iter = 10;
60          backtracking_iter = 0;
61
62          while f(x + alpha * p) > f_curr + c * alpha * g' * p
                && backtracking_iter < max_backtracking_iter
63          alpha = rho * alpha;
64          backtracking_iter = backtracking_iter + 1;
65          end
66
67          % Update iterate
68          x = x + alpha * p;
69
70          f_old = f_curr;
71          f_curr = f(x);
72
73          if iter < 5
74          e_rate(:,iter) = x;
75          end
76
77          if iter >= 5
78          e_rate = [e_rate(:, 2:4), x];
79          end
80
81
82          % New gradient
83          if fd
84          g = grad_f(x,h,type);
85          else
86          g = grad_f(x);
87          end
88
89          % Check stopping criterion
90          if norm(g,inf) <= tol
```

```
        break
        end

        if abs(f_curr - f_old) <= tol*max(1,abs(f_old))
        break
        end


        iter = iter + 1;
        end

        % Output final results
        x_min = x;
        f_min = f(x);
        min_history = min_history(1:iter);
        grad_norm = norm(g,inf);


        end


        function [L, tau] = alg63_cholesky(A, maxIter)

        n    = size(A,1);

        % Step 1:    = ||A||_F
        beta = norm(A, 'fro');

        % % Step 2:  0
        % if min(diag(A)) > 0
        %     tau = 0;
        % else
        %     tau = min(beta/2, 1e-1);   % non partire oltre
            0.1
        % end

        % Step 2:    iniziale
        tau0 = 1e-3;
        tau  = 0;

        I = speye(n);

        % Step 3
        for k = 0:maxIter
        [L,flag] = chol(A + tau*I,'lower');   % L*L' = A+ I
        if flag == 0                          %  OK
        return
        end
        %tau = max(2*tau, beta/2);
```

```
        if tau == 0
        tau = max(tau0, min(beta/2, 1e-1));
        else
        tau = 2 * tau;
        end

        end

        error('Alg63: fallito dopo %d tentativi', maxIter);
        end

        function xbar = initial_solution_bt(n)

        xbar = ones(n+2, 1);
        xbar(1) = 0;
        xbar(n+2) = 0;

        end

        function q = compute_ecr(X)


        d = zeros(1,3);
        for k = 1:3
        d(k) = norm(X(:,k+1) - X(:, k), 2);
        end


        q = log(d(3) / d(2)) / log(d(2) / d(1));
        end

        %10 random points
        function X0 = generate_initial_points(x_bar,
            num_points)

        n = length(x_bar);
        X0 = repmat(x_bar, 1, num_points) + 2*rand(n,
            num_points) - 1;

        end

        %success function
        function esito = is_success(grad_norm, tol_success)

        if grad_norm > 0 && grad_norm < tol_success
        esito = 1;
        else
        esito = 0;
        end
```

```
end

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       TEST DELL'ALGORITMO SULLA FUNZIONE DI BANDED TRIGONOMETRIC
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Exact gradient
grad_banded_tr = @(x) [sin(x(2)) + 2*cos(x(2));
(2:length(x)-3)'.*sin(x(3:end-2)) + 2*cos(x(3:end-2))
    ;
(length(x)-2) * sin(x(end-1)) - (length(x)-3) * cos(x
    (end-1))];

% Exact hessian
function H = banded_trig_hess(x)
n = length(x);
i = (2:n-3)';
diag_princ = zeros(n-2, 1);
diag_princ(2:n-3) = i.*cos(x(3:end-2)) - 2*sin(x(3:
    end-2));
diag_princ(1) = cos(x(2)) - 2*sin(x(2));
diag_princ(n-2) = (n-2)*cos(x(end-1)) + (n-3)*sin(x(
    end-1));
H = spdiags(diag_princ, 0, n-2, n-2);
end

% Finite differences gradient
function grad_fd = banded_trigonometric_gradf_fd(x, h
    , type)
n = length(x);
if type
hs = h*abs(x);
else
hs = h*ones(n, 1);
end

i = (1:n-1)';

grad_fd = [
2*i.*sin(x(1:n-1)).*sin(hs(1:n-1)) + 4*cos(x(1:n-1))
    .*sin(hs(1:n-1));
2*n.*sin(x(n)).*sin(hs(n)) - 2*(n-1)*cos(x(n)).*sin(
    hs(n));
];
```

```
223         grad_fd = grad_fd ./ (2*hs);
224         end
225
226         % Finite differences hessian
227         function H = banded_trigonometric_hessf_fd(x, h, type
                )
228         n = length(x);
229         if type
230         hs = h*abs(x);
231         else
232         hs = h*ones(n, 1);
233         end
234         i = (1:n)';
235         diag = (-i.*cos(x) + 2*sin(x)).*(-1) + (i.*sin(x) +
                2*cos(x)).*(-hs);
236         diag(n) = (-n.*cos(x(n)) - (n-1)*sin(x(n))).*(-1) + (
                n.*sin(x(n)) - (n-1)*cos(x(n))).*(-hs(n));
237         H = sparse(1:n,1:n,diag,n,n);
238         end
239
240         matricole = [295706, 302689]; %ora 295706
                diventato 349152
241         rng(min(matricole));
242
243         max_iter = 5000;  % Maximum number of iterations
244         tol = 1e-6;
245         num_points = 10;
246         k = 2:2:12;
247         h = power(10,-k); % increment of the finite
                differences
248         n_NewtonModified = [1000, 10000, 100000];
249         time_dim  = zeros(3);    % times that we collect
250         a = 1;
251
252         banded_tr = @(x) sum((1:length(x)-2)' .* ((1 - cos(x
                (2:end-1))) + sin(x(1:end-2)) - sin(x(3:end))));
253
254         t_total = tic;
255
256         for j=n_NewtonModified
257
258         t0 = tic;
259
260         fprintf('\n
                ==================================================\
                n');
261         fprintf(' TEST SU BANDED TRIGONOMETRIC IN DIMENSIONE
                %d \n', j);
```

```
262         fprintf('
                ==================================================\
                n\n');
263
264         % Start point
265         x_bar = initial_solution_bt(j);
266
267         % 10 starting points random
268         X0 = generate_initial_points(x_bar, num_points);
269
270         fd = 1;
271
272         if fd == 1
273         grad_f = @banded_trigonometric_gradf_fd;
274         hess_f = @banded_trigonometric_hessf_fd;
275         end
276
277         if fd == 1
278
279         fprintf('\n
                ==================================================\
                n');
280         fprintf(' TEST SU DERIVATE APPROSSIMATE CON
                DIFFERENZE FINITE \n');
281         fprintf('
                ==================================================\
                n\n');
282
283         for increment = h
284
285         fprintf('\n
                --------------------------------------------------
                ');
286         fprintf('\nDefault increment h = %d \n',increment);
287         fprintf('
                --------------------------------------------------\
                n');
288
289         % === TEST SU x_bar === %
290         fprintf('\n--- TEST SU VALORE X_BAR ---\n');
291         tic;
292         [~, f_min, iter_bar, min_hist_bar,grad_norm_bar,
                e_rate_bar] = modified_newton(banded_tr,grad_f,
                hess_f,x_bar,tol,max_iter,fd,increment,0);
293         t = toc;
294         fprintf('f_min = %.6f\n | iter = %d | tempo = %.3fs\n
                 | grad_norm = %.6f\n', f_min, iter_bar, t,
                grad_norm_bar);
295         rho = compute_ecr(e_rate_bar);
```

```
296         fprintf('rho      %.4f\n\n', rho);
297
298         % === TEST SU 10 PUNTI CASUALI === %
299         fprintf('\n--- TEST SU 10 PUNTI CASUALI ---\n');
300         min_hist_all = cell(num_points, 1);
301         successi = 0;
302         for i = 1:num_points
303         x0 = X0(:,i);
304         fprintf('\n--- Test %d (x0 #%d) ---\n', i, i);
305         tic;
306         [~, f_min, iter, min_hist, grad_norm, e_rate] =
               modified_newton(banded_tr,grad_f,hess_f,x0,tol,
               max_iter,fd,increment,0);
307         t = toc;
308         fprintf('f_min = %.6f\n | iter = %d | tempo = %.3fs\n
               | grad_norm = %.6f\n', f_min, iter, t, grad_norm)
               ;
309         rho = compute_ecr(e_rate);
310         fprintf('rho      %.4f\n\n', rho);
311         min_hist_all{i} = min_hist;
312         successi = successi + is_success(grad_norm, 0.5);
313         end
314         fprintf('\nSuccessi: %d su %d\n', successi,
               num_points);
315
316         % Absolute value increment
317         fprintf('\n
               ----------------------------------------------------
               ');
318         fprintf('\nAbsolute value increment h = %d*|x| \n',
               increment);
319         fprintf('
               ----------------------------------------------------\
               n');
320
321         % === TEST SU x_bar === %
322         fprintf('\n--- TEST SU VALORE X_BAR ---\n');
323         tic;
324         [x_min, f_min, iter_bar_abs, min_hist_bar_abs,
               grad_norm, e_rate_abs] = modified_newton(banded_tr
               ,grad_f,hess_f,x_bar,tol,max_iter,fd,increment,1);
325         t = toc;
326         fprintf('f_min = %.6f\n | iter = %d | tempo = %.3fs\n
               | grad_norm = %.6f\n', f_min, iter_bar_abs, t,
               grad_norm);
327         rho = compute_ecr(e_rate_abs);
328         fprintf('rho      %.4f\n\n', rho);
329
330         % === TEST SU 10 PUNTI CASUALI === %
```

```
331         fprintf('\n--- TEST SU 10 PUNTI CASUALI ---\n');
332         min_hist_all_abs = cell(num_points, 1);
333         successi = 0;
334         for i = 1:num_points
335         if i==3 && j==100000
336         i=2; %perche il test 3 in dim 100k impiega 4 minuti
337         end
338         x0 = X0(:,i);
339         fprintf('\n--- Test %d (x0 #%d) ---\n', i, i);
340         tic;
341         [x_min, f_min, iter, min_hist, grad_norm, e_rate] =
               modified_newton(banded_tr,grad_f,hess_f,x0,tol,
               max_iter,fd,increment,1);
342         t = toc;
343         fprintf('f_min = %.6f\n | iter = %d | tempo = %.3fs\n
               | grad_norm = %.6f\n', f_min, iter, t, grad_norm)
               ;
344         rho = compute_ecr(e_rate);
345         fprintf('rho      %.4f\n\n', rho);
346         min_hist_all_abs{i} = min_hist;
347         successi = successi + is_success(grad_norm, 0.5);
348         end
349         fprintf('\nSuccessi: %d su %d\n', successi,
               num_points);
350
351         % === FIGURE ===
352         fig = figure('Units','normalized','Position',[0.12
               0.12 0.78 0.62]);
353         tl = tiledlayout(fig,1,2,'TileSpacing','compact','
               Padding','compact');
354         colors = lines(num_points);
355
356         % -- LEFT: h
357         nexttile(tl,1); hold on;
358         plot(1:iter_bar, min_hist_bar, '-o', 'LineWidth',
               1.8, 'Color', 'k', 'DisplayName', 'x  ');
359         for i = 1:num_points
360         mh = min_hist_all{i};
361         plot(1:length(mh), mh, '-o', 'LineWidth', 1.2, '
               MarkerSize', 5, 'Color', colors(i,:), 'DisplayName
               ', sprintf(' x   #%d', i));
362         end
363         title(sprintf('h = %.1e', increment), 'FontSize', 12)
               ;
364         xlabel('Iterazioni'); ylabel('f(x_k)');
365         set(gca, 'YScale', 'linear');
366         ylim('auto');
367         grid on; box on; set(gca, 'FontSize', 11);
368
```

```
369         % -- RIGHT: h * |x|
370         nexttile(tl,2); hold on;
371         plot(1:iter_bar_abs, min_hist_bar_abs, '-o', '
                LineWidth', 1.8, 'Color', 'k', 'DisplayName', 'x
                ');
372         for i = 1:num_points
373         mh = min_hist_all_abs{i};
374         plot(1:length(mh), mh, '-o', 'LineWidth', 1.2, '
                MarkerSize', 5, 'Color', colors(i,:), 'DisplayName
                ', sprintf(' x   #%d', i));
375         end
376         title(sprintf('h = %.1e  |x|', increment), 'FontSize'
                , 12);
377         xlabel('Iterazioni'); ylabel('f(x_k)');
378         set(gca, 'YScale', 'linear');
379         ylim('auto');
380         grid on; box on; set(gca, 'FontSize', 11);
381
382         title(tl, sprintf('Convergenza Metodo Newton
                Modificato     n = %d', j), 'FontSize', 14);
383         legend('show', 'Location', 'eastoutside');
384
385         end
386         end
387
388         fd = 0;
389
390         if fd == 0
391         grad_f = grad_banded_tr;
392         hess_f = @banded_trig_hess;
393         end
394
395         if fd == 0
396
397         fprintf('\n
                ================================================\
                n');
398         fprintf(' TEST SU DERIVATE ESATTE \n');
399         fprintf('
                ================================================\
                n\n');
400
401         x_bar = initial_solution_bt(j);
402
403         X0 = generate_initial_points(x_bar, num_points);
404
405         % === TEST SU x_bar ===
406         fprintf('\n--- TEST SU VALORE X_BAR ---\n');
407         tic;
```

```
408         [x_min, f_min, iter_bar, min_hist_bar, grad_norm,
                e_rate] = modified_newton(banded_tr,grad_f,hess_f,
                x_bar,tol,max_iter,fd,[],[]);
409         t = toc;
410         fprintf('f_min = %.6f\n | iter = %d | tempo = %.3fs\n
                | grad_norm = %.6f\n', f_min, iter_bar, t,
                grad_norm);
411         rho = compute_ecr(e_rate);
412         fprintf('rho      %.4f\n', rho);
413
414         % === TEST SU 10 PUNTI CASUALI ===
415         min_hist_all = cell(num_points, 1);
416         successi = 0;
417         for i = 1:num_points
418         x0 = X0(:,i);
419         fprintf('\n--- Test %d (x0 #%d) ---\n', i, i);
420         tic;
421         [x_min, f_min, iter, min_hist, grad_norm, e_rate] =
                modified_newton(banded_tr,grad_f,hess_f,x0,tol,
                max_iter,fd,[],[]);
422         t = toc;
423         fprintf('f_min = %.6f\n | iter = %d | tempo = %.3fs\n
                | grad_norm = %.6f\n', f_min, iter, t, grad_norm)
                ;
424         rho = compute_ecr(e_rate);
425         fprintf('rho      %.4f\n', rho);
426         min_hist_all{i} = min_hist;
427         successi = successi + is_success(grad_norm, 0.5);
428         end
429
430         fprintf('\nSuccessi: %d su %d\n', successi,
                num_points);
431
432         % === PLOT CONVERGENZA ===
433         figure('Units', 'normalized', 'Position', [0.2 0.2
                0.6 0.6]);  % finestra grande
434         hold on;
435
436         % Plot x
437         plot(1:iter_bar, min_hist_bar, '-o', 'LineWidth',
                1.8, ...
438         'DisplayName', 'x  ', 'Color', 'k');
439
440         colors = lines(num_points);
441
442         % Plot dei 10 punti iniziali random
443         for i = 1:num_points
444         mh = min_hist_all{i};
445         plot(1:length(mh), mh, '-o', ...
```

```
446         'LineWidth', 1.2, ...
447         'MarkerSize', 5, ...
448         'Color', colors(i,:), ...
449         'DisplayName', sprintf(' x   #%d', i));
450         end
451
452         % Titles
453         xlabel('Iterazioni', 'FontSize', 12);
454         ylabel('Valore funzione obiettivo', 'FontSize', 12);
455         title(sprintf('Convergenza Metodo su Banded
               Trigonometric Esatto (n = %d)', j), 'FontSize',
               14);
456
457         % Style
458         legend('show', 'Location', 'eastoutside');
459         grid on;
460         set(gca, 'YScale', 'linear');
461
462         box on;
463         set(gca, 'FontSize', 12);
464         hold off;
465
466         end
467
468         time_dim(a) = toc(t0);
469         fprintf('\nTempo (incl. plotting) per n = %-7d  :
               %.2f  s\n',j, time_dim(a));
470         a = a + 1;
471         end
472
473         % ---------------------- TOTAL TIME SCRIPT
               -----------------------
474         fprintf('\n
               ================================================\
               n');
475         fprintf(' TABELLA TEMPISTICHE ALGORITMO BANDED
               TRIGONOMETRIC \n');
476         fprintf('
               ================================================\
               n\n');
477
478         time_total = toc(t_total);
479
480         fprintf('\nTempo (incl. plotting) per n = %-7d  :
               %.2f  s\n', ...
481         n_NewtonModified(1), time_dim(1));
482         fprintf('\nTempo (incl. plotting) per n = %-7d  :
               %.2f  s\n', ...
483         n_NewtonModified(2), time_dim(2));
```

```
484         fprintf('\nTempo (incl. plotting) per n = %-7d  :
                %.2f  s\n', ...
485         n_NewtonModified(3), time_dim(3));
486         fprintf('\nTempo TOTALE (tutte le dimensioni) :  %.2f
                s\n', time_total);
487
488         %---  bar chart  ---
489         figure;
490         bar(categorical(string(n_NewtonModified)), time_dim);
491         ylabel('Tempo (s)'); grid on;
```

Listing 4: Full script: Modified Newton method on Banded Trigonometric

Nelder Mead Method on all function

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PARTE 1 - IMPLEMENTAZIONE DELL'ALGORITMO NELDER-MEAD
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [x_min, f_min, iter, min_history, e_rate] = nelder_mead(f, x0, tol, max_iter)

n = length(x0); % Dimension of the problem
rho = 1;      % Reflection coefficient
chi = 2;      % Expansion coefficient
gamma = 0.5;      % Contraction coefficient
sigma = 0.5;    % Shrinkage coefficient

% Initialize simplex
simplex = zeros(n + 1, n);
simplex(1, :) = x0';
e_rate = zeros(n,4);
identity_matrix = eye(n);

for i = 2:n+1
simplex(i, :) = x0' + 0.03 * identity_matrix(:, i-1)'; % create new points similar to the first one
end

% Evaluate function at simplex points
f_vals = arrayfun(@(i) f(simplex(i, :)'), 1:n+1); % Compute the correspondent value of the function for each row (the same of sapply() in R)
vectorNorm = zeros(n, 1);
min_history = zeros(1, max_iter);
iter = 1;

while iter < max_iter

% Sort vertices by function values
[f_vals, idx] = sort(f_vals);
simplex = simplex(idx, :);
min_history(iter) = f_vals(1);

% Compute centroid of all points except worst
centroid = mean(simplex(1:n, :));

```

```
39          % Reflection
40          x_r = centroid + rho * (centroid - simplex(n+1, :));
41          f_r = f(x_r');
42
43          if f_r < f_vals(1) % Expansion
44          x_e = centroid + chi * (x_r - centroid);
45          f_e = f(x_e');
46          if f_e < f_r
47          simplex(n+1, :) = x_e;
48          f_vals(n+1) = f_e;
49          else
50          simplex(n+1, :) = x_r;
51          f_vals(n+1) = f_r;
52          end
53          elseif f_r < f_vals(n) % Accept reflection
54          simplex(n+1, :) = x_r;
55          f_vals(n+1) = f_r;
56          else % Contraction
57          if f_vals(n+1) < f_r
58          x_c = centroid - gamma * (centroid - f_vals(n+1));
59          else
60          x_c = centroid - gamma * (centroid - f_r);
61          end
62          f_c = f(x_c');
63          if f_c < f_vals(n+1)
64          simplex(n+1, :) = x_c;
65          f_vals(n+1) = f_c;
66          else % Shrink
67          for i = 2:n+1
68          simplex(i, :) = simplex(1, :) + sigma * (simplex(i,
                :) - simplex(1, :));
69          f_vals(i) = f(simplex(i, :)');
70          end
71          end
72          end
73
74          [f_vals, idx] = sort(f_vals);
75          simplex = simplex(idx, :);
76          term_f = abs(f_vals(n+1) - f_vals(1));
77
78          for i = 2:n+1
79          vectorNorm(i-1) = norm(simplex(i,:) - simplex(1,:),
                inf);
80          end
81
82          term_x = max(vectorNorm);
83
84          if iter < 5 && norm(e_rate(:,iter) - simplex(1,:)',
                2) >= tol
```

```
        e_rate(:,iter) = simplex(1,:)';
        end

        if iter >= 5 && norm(e_rate(:,4) - simplex(1,:)', 2) >= tol
        e_rate = [e_rate(:, 2:4), simplex(1,:)'];
        end


        % Check convergence
        if term_f <= tol || term_x <= tol
        break;
        end

        iter = iter + 1;
        end

        x_min = simplex(1, :);
        f_min = f_vals(1);
        min_history = min_history(1:iter);
        end

        % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % PARTE 2 - TEST DELL'ALGORITMO SULLA FUNZIONE DI ROSENBROCK
        % Test con due punti iniziali richiesti dall'assignment:
        % [1.2, 1.2] e [-1.2, 1.0]
        % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        % Nelder-Mead test on Rosenbrock function
        clc; clear; close all;

        % Define Rosenbrock function
        rosenbrock = @(x) 100 * (x(2) - x(1)^2)^2 + (1 - x(1))^2;

        % Initial points
        x0_1 = [1.2, 1.2]';
        x0_2 = [-1.2, 1.0]';

        % Parameters
        tol = 1e-6;         % Tolerance for convergence
        max_iter = 500;   % Maximum number of iterations

```

```matlab
    % Run Nelder-Mead for both starting points
    [x_min1, f_min1, iter1, min_history1] = nelder_mead(rosenbrock, x0_1, tol, max_iter);
    [x_min2, f_min2, iter2, min_history2] = nelder_mead(rosenbrock, x0_2, tol, max_iter);

    % Display results
    fprintf('\n==================================================\n');
    fprintf(' TEST SU ROSENBROCK IN DUE DIMENSIONI      \n');
    fprintf('==================================================\n\n');

    fprintf('Starting point: [1.2, 1.2]\n');
    fprintf('Minimum found: [%f, %f]\n', x_min1(1), x_min1(2));
    fprintf('Function value: %f\n', f_min1);
    fprintf('Iterations: %d\n\n', iter1);

    fprintf('Starting point: [-1.2, 1.0]\n');
    fprintf('Minimum found: [%f, %f]\n', x_min2(1), x_min2(2));
    fprintf('Function value: %f\n', f_min2);
    fprintf('Iterations: %d\n\n', iter2);

    % Plot figures
    iterations_1= 1:iter1;
    iterations_2= 1:iter2;

    figure;
    plot(iterations_1, min_history1(1:iter1), '-o', 'DisplayName', '[1.2, 1.2]');
    hold on;
    plot(iterations_2, min_history2(1:iter2), '-x', 'DisplayName', '[-1.2, 1.0]');
    hold off;
    xlabel('Numero di Iterazioni');
    ylabel('Valore della Funzione Obiettivo');
    title('Convergenza del Metodo Nelder-Mead sulla Funzione di Rosenbrock');
    legend show;
    grid on;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% PARTE 3 - TEST DELL'ALGORITMO SULLA FUNZIONE DI EXTENDED ROSENBROCK
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%intro
matricole = [295706, 302689]; %ora 295706 diventato 349152
rng(min(matricole));

n_NelderMead = [10, 26, 50];

extended_rosenbrock = @(x) 0.5*sum([10*(x(1:2:end).^2 - x(2:2:end)); x(1:2:end-1)-1].^2);

function xbar = initial_solution_er(n)

xbar = ones(n, 1);
xbar(1:2:end) = -1.2;

end

function X0 = generate_initial_points_er(x_bar, num_points)
n = length(x_bar);
X0 = repmat(x_bar, 1, num_points) + 2*rand(n, num_points) - 1;
end

function esito = is_success(f_min, tol_success)

if f_min > 0 && f_min < tol_success
esito = 1;
elseif f_min < 0 && f_min > -tol_success
esito = 1;
else
esito = 0;
end
end

function esito = is_success_banded(f_min, tol_success, dim)
if dim==10
if f_min > -10.24 && f_min < -10.24 + tol_success
esito = 1;
elseif f_min < -10.24 && f_min > -10.24 - tol_success
esito = 1;
else
```

```
202         esito = 0;
203         end
204         end
205 
206         if dim==26
207         if f_min > -23.34 && f_min < -23.34 + tol_success
208         esito = 1;
209         elseif f_min < -23.34 && f_min > -23.34 - tol_success
210         esito = 1;
211         else
212         esito = 0;
213         end
214         end
215 
216         if dim==50
217         if f_min > -41.58 && f_min < -41.58 + tol_success
218         esito = 1;
219         elseif f_min < -41.58 && f_min > -41.58 - tol_success
220         esito = 1;
221         else
222         esito = 0;
223         end
224         end
225 
226         end
227 
228         function q = compute_ecr(X)
229 
230         d = zeros(1,3);
231         for k = 1:3
232         d(k) = norm(X(:,k+1) - X(:, k), 2);
233         end
234 
235         q = log(d(3) / d(2)) / log(d(2) / d(1));
236         end
237 
238         %INIZIO TEST
239         max_iter = 80000;  % Maximum number of iterations
240         tol = 1e-6;
241         num_points = 10;
242         time_dim  = zeros(3);    %     times collected
243         a = 1;
244 
245         t_total = tic;
246 
247         for j=n_NelderMead
248 
249         t0 = tic;
250 
```

```
251         fprintf('\n
                ==============================================\n')
                ;
252         fprintf(' TEST SU EXTENDED ROSENBROCK IN DIMENSIONE %
                d \n', j);
253         fprintf('
                ==============================================\n\n
                ');
254
255         x_bar = initial_solution_er(j);
256
257         X0 = generate_initial_points_er(x_bar, num_points);
258
259         % === TEST SU x_bar ===
260         fprintf('\n--- TEST SU VALORE X_BAR ---\n');
261         tic;
262         [x_min, f_min, iter, min_hist_bar,e_rate] =
                nelder_mead(extended_rosenbrock, x_bar, tol,
                max_iter);
263         t = toc;
264         fprintf('f_min = %.6f\n | iter = %d | tempo = %.2fs\n
                ', f_min, iter, t);
265         rho = compute_ecr(e_rate);
266         fprintf('rho      %.4f\n', rho);
267
268
269         % === TEST SU 10 PUNTI CASUALI ===
270         min_hist_all = cell(num_points, 1);
271         successi = 0;
272         for i = 1:num_points
273         x0 = X0(:,i);
274         fprintf('\n--- Test %d (x0 #%d) ---\n', i, i);
275         tic;
276         [x_min, f_min, iter, min_hist, e_rate] = nelder_mead(
                extended_rosenbrock, x0, tol, max_iter);
277         t = toc;
278         fprintf('f_min = %.6f\n | iter = %d | tempo = %.2fs\n
                ', f_min, iter, t);
279         rho = compute_ecr(e_rate);
280         fprintf('rho      %.4f\n', rho);
281         min_hist_all{i} = min_hist;
282         successi = successi + is_success(f_min, 2);
283         end
284
285         fprintf('\nSuccessi: %d su %d\n', successi,
                num_points);
286
287         % === PLOT CONVERGENZA ===
```

```
288         figure('Units', 'normalized', 'Position', [0.2 0.2
                0.6 0.6]);  % finestra ampia
289         hold on;
290
291         % --- Plot x   ---
292         plot(1:length(min_hist_bar), min_hist_bar, '-k', ...
293         'LineWidth', 2.2, 'DisplayName', 'x  ');
294
295         % --- Automatic colours ---
296         colors = lines(num_points);
297
298         % --- Plot 10 casual points ---
299         for i = 1:num_points
300         mh = min_hist_all{i};
301         plot(1:length(mh), mh, '-o', ...
302         'LineWidth', 1.2, ...
303         'MarkerSize', 4, ...
304         'Color', colors(i,:), ...
305         'DisplayName', sprintf(' x   #%d', i));
306         end
307
308         % --- Labels and titles ---
309         xlabel('Iterazioni', 'FontSize', 13);
310         ylabel('Valore funzione obiettivo', 'FontSize', 13);
311         title(sprintf('Convergenza Nelder-Mead su Extended
                Rosenbrock (n = %d)', j), 'FontSize', 14);
312
313         % --- Legenda e stile ---
314         legend('show', 'Location', 'northeastoutside');
315         grid on;
316         set(gca, 'YScale', 'log');
317
318         box on;
319         set(gca, 'FontSize', 12);
320         hold off;
321
322         time_dim(a) = toc(t0);
323         fprintf('\nTempo (incl. plotting) per n = %-7d  :
                %.2f  s\n', ...
324         j, time_dim(a));
325         a = a + 1;
326
327         end
328
329         % ---------------------- TEMPO TOTALE SCRIPT
                ----------------------
330         fprintf('\n
                ================================================\
                n');
```

```
        fprintf(' TABELLA TEMPISTICHE ALGORITMO EXTENDED ROSENBROCK \n');
        fprintf('=================================================\n\n');

        time_total = toc(t_total);

        fprintf('\nTempo (incl. plotting) per n = %-7d  : %.2f  s\n', ...
        n_NelderMead(1), time_dim(1));
        fprintf('\nTempo (incl. plotting) per n = %-7d  : %.2f  s\n', ...
        n_NelderMead(2), time_dim(2));
        fprintf('\nTempo (incl. plotting) per n = %-7d  : %.2f  s\n', ...
        n_NelderMead(3), time_dim(3));
        fprintf('\nTempo TOTALE (tutte le dimensioni) :  %.2f  s\n', time_total);

        %---  bar chart  ---
        figure;
        bar(categorical(string(n_NelderMead)), time_dim);
        ylabel('Tempo (s)'); grid on;

        % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % PARTE 3 - TEST DELL'ALGORITMO SULLA FUNZIONE Generalized Broyden tridiagonal
        % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        generalized_broyden = @(x) 0.5*sum(((3-2*x(2:end-1)).*x(2:end-1)+1-x(1:end-2)-x(3:end)).^2);

        function xbar = initial_solution_gb(n)

        xbar = -ones(n+2, 1);
        xbar(1) = 0;
        xbar(n+2) = 0;

        end

        function X0 = generate_initial_points_gb(x_bar, num_points)
        n = length(x_bar);
```

```
365         X0 = repmat(x_bar, 1, num_points) + 2*rand(n,
                num_points) - 1;
366         end
367
368         %INIZIO TEST
369         max_iter = 80000;  % Maximum number of iterations
370         tol = 1e-6;
371         num_points = 10;
372         time_dim  = zeros(3);    %     times collected
373         a = 1;
374
375         t_total = tic;
376
377         for j=n_NelderMead
378
379         t0 = tic;
380
381         fprintf('\n
                ==============================================\n')
                ;
382         fprintf(' TEST SU GENERALYZED BROYDEN TRIDIAGONAL IN
                DIMENSIONE %d \n', j);
383         fprintf('
                ==============================================\n\n
                ');
384
385         x_bar = initial_solution_gb(j);
386
387         X0 = generate_initial_points_gb(x_bar, num_points);
388
389         % === TEST SU x_bar ===
390         fprintf('\n--- TEST SU VALORE X_BAR ---\n');
391         tic;
392         [x_min, f_min, iter, min_hist_bar, e_rate] =
                nelder_mead(generalized_broyden, x_bar, tol,
                max_iter);
393         t = toc;
394         fprintf('f_min = %.6f\n | iter = %d | tempo = %.2fs\n
                ', f_min, iter, t);
395         rho = compute_ecr(e_rate);
396         fprintf('rho     %.4f\n', rho);
397
398         % === TEST SU 10 PUNTI CASUALI ===
399         min_hist_all = cell(num_points, 1);
400         successi = 0;
401         for i = 1:num_points
402         x0 = X0(:,i);
403         fprintf('\n--- Test %d (x0 #%d) ---\n', i, i);
404         tic;
```

```
405         [x_min, f_min, iter, min_hist, e_rate] = nelder_mead(
                generalized_broyden, x0, tol, max_iter);
406         t = toc;
407         fprintf('f_min = %.6f\n | iter = %d | tempo = %.2fs\n
                ', f_min, iter, t);
408         rho = compute_ecr(e_rate);
409         fprintf('rho      %.4f\n', rho);
410         min_hist_all{i} = min_hist;
411         successi = successi + is_success(f_min, 2);
412         end
413
414         fprintf('\nSuccessi: %d su %d\n', successi,
                num_points);
415
416         % === PLOT CONVERGENZA ===
417         figure('Units', 'normalized', 'Position', [0.2 0.2
                0.6 0.6]);  % finestra ampia
418         hold on;
419
420         % --- Plot  x    ---
421         plot(1:length(min_hist_bar), min_hist_bar, '-k', ...
422         'LineWidth', 2.2, 'DisplayName', 'x  ');
423
424         % --- Colours---
425         colors = lines(num_points);
426
427         % --- Plot 10 casual points ---
428         for i = 1:num_points
429         mh = min_hist_all{i};
430         plot(1:length(mh), mh, '-o', ...
431         'LineWidth', 1.2, ...
432         'MarkerSize', 4, ...
433         'Color', colors(i,:), ...
434         'DisplayName', sprintf(' x   #%d', i));
435         end
436
437         % --- Labels ---
438         xlabel('Iterazioni', 'FontSize', 13);
439         ylabel('Valore funzione obiettivo', 'FontSize', 13);
440         title(sprintf('Convergenza Nelder-Mead su Generalyzed
                Broyden (n = %d)', j), 'FontSize', 14);
441
442         % --- Legenda e stile ---
443         legend('show', 'Location', 'northeastoutside');
444         grid on;
445         set(gca, 'YScale', 'log');
446
447         box on;
448         set(gca, 'FontSize', 12);
```

```
        hold off;

        time_dim(a) = toc(t0);
        fprintf('\nTempo (incl. plotting) per n = %-7d  :  %.2f  s\n', ...
        j, time_dim(a));
        a = a + 1;

        end

        % --------------------- TEMPO TOTALE SCRIPT ----------------------
        fprintf('\n =================================================\n');
        fprintf(' TABELLA TEMPISTICHE ALGORITMO GENERALIZED BROYDEN \n');
        fprintf(' =================================================\n\n');

        time_total = toc(t_total);

        fprintf('\nTempo (incl. plotting) per n = %-7d  :  %.2f  s\n', ...
        n_NelderMead(1), time_dim(1));
        fprintf('\nTempo (incl. plotting) per n = %-7d  :  %.2f  s\n', ...
        n_NelderMead(2), time_dim(2));
        fprintf('\nTempo (incl. plotting) per n = %-7d  :  %.2f  s\n', ...
        n_NelderMead(3), time_dim(3));
        fprintf('\nTempo TOTALE (tutte le dimensioni) :  %.2f  s\n', time_total);

        %---  bar chart  ---
        figure;
        bar(categorical(string(n_NelderMead)), time_dim);
        ylabel('Tempo (s)'); grid on;

        % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % PARTE 3 - TEST DELL'ALGORITMO SULLA FUNZIONE BANDED TRIGONOMETRIC
        % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```
482         banded_trigonometric = @(x) sum((1:length(x)-2)' .*
                ((1 - cos(x(2:end-1))) + sin(x(1:end-2)) - sin(x
                (3:end))));
483
484         function xbar = initial_solution_bt(n)
485
486         xbar = ones(n+2, 1);
487         xbar(1) = 0;
488         xbar(n+2) = 0;
489
490         end
491
492         function X0 = generate_initial_points_bt(x_bar,
                num_points)
493         n = length(x_bar);
494         X0 = repmat(x_bar, 1, num_points) + 2*rand(n,
                num_points) - 1;
495         end
496
497         %INIZIO TEST
498         max_iter = 800000;  % Maximum number of iterations
499         tol = 1e-6;
500         num_points = 10;
501         time_dim  = zeros(3);
502         a = 1;
503
504         t_total = tic;
505
506         for j=n_NelderMead
507
508         t0 = tic;
509
510         fprintf('\n
                ==============================================\n')
                ;
511         fprintf(' TEST SU BANDED TRIGONOMETRIC IN DIMENSIONE
                %d \n', j);
512         fprintf('
                ==============================================\n\n
                ');
513
514         x_bar = initial_solution_bt(j);
515
516         X0 = generate_initial_points_bt(x_bar, num_points);
517
518         % === TEST SU x_bar ===
519         fprintf('\n--- TEST SU VALORE X_BAR ---\n');
520         tic;
```

```
521         [x_min, f_min, iter, min_hist_bar, e_rate] =
                nelder_mead(banded_trigonometric, x_bar, tol,
                max_iter);
522         t = toc;
523         fprintf('f_min = %.6f\n | iter = %d | tempo = %.2fs\n
                ', f_min, iter, t);
524         min_hist_bar = min_hist_bar(1:iter);
525         rho = compute_ecr(e_rate);
526         fprintf('rho      %.4f\n', rho);
527
528         % === TEST SU 10 PUNTI CASUALI ===
529         min_hist_all = cell(num_points, 1);
530         successi=0;
531         for i = 1:num_points
532         x0 = X0(:, i);
533         fprintf('\n--- Test %d (x0 #%d) ---\n', i, i);
534         tic;
535         [x_min, f_min, iter, min_hist, e_rate] = nelder_mead(
                banded_trigonometric, x0, tol, max_iter);
536         t = toc;
537         fprintf('f_min = %.6f\n | iter = %d | tempo = %.2fs\n
                ', f_min, iter, t);
538         min_hist = min_hist(1:iter-1);
539         rho = compute_ecr(e_rate);
540         fprintf('rho      %.4f\n', rho);
541         min_hist_all{i} = min_hist;
542         successi = successi + is_success_banded(f_min, 2, j);
543
544         end
545
546         fprintf('\nSuccessi: %d su %d\n', successi,
                num_points);
547
548         % === PLOT CONVERGENZA ===
549         figure('Units', 'normalized', 'Position', [0.2 0.2
                0.6 0.6]);
550         hold on;
551
552         % --- Plot x     ---
553         plot(1:length(min_hist_bar), min_hist_bar, '-k', ...
554         'LineWidth', 2.2, 'DisplayName', 'x  ');
555
556         % --- Colours ---
557         colors = lines(num_points);
558
559         % --- Plot 10 casual points ---
560         for i = 1:num_points
561         mh = min_hist_all{i};
562         plot(1:length(mh), mh, '-o', ...
```

```
563         'LineWidth', 1.2, ...
564         'MarkerSize', 4, ...
565         'Color', colors(i,:), ...
566         'DisplayName', sprintf(' x   #%d', i));
567         end
568
569         % ---  labels ---
570         xlabel('Iterazioni', 'FontSize', 13);
571         ylabel('Valore funzione obiettivo', 'FontSize', 13);
572         title(sprintf('Convergenza Nelder-Mead su Banded
                Trigonometric (n = %d)', j), 'FontSize', 14);
573
574         % --- Legenda e stile ---
575         legend('show', 'Location', 'northeastoutside');
576         grid on;
577         set(gca, 'YScale', 'linear');
578
579         box on;
580         set(gca, 'FontSize', 12);
581         hold off;
582
583         time_dim(a) = toc(t0);
584         fprintf('\nTempo (incl. plotting) per n = %-7d  :
                %.2f  s\n', ...
585         j, time_dim(a));
586         a = a + 1;
587
588         end
589
590         % ---------------------- TEMPO TOTALE SCRIPT
                ----------------------
591         fprintf('\n
                ================================================\
                n');
592         fprintf(' TABELLA TEMPISTICHE ALGORITMO BANDED
                TRIDIAGONAL \n');
593         fprintf('
                ================================================\
                n\n');
594
595         time_total = toc(t_total);
596
597         fprintf('\nTempo (incl. plotting) per n = %-7d  :
                %.2f  s\n', ...
598         n_NelderMead(1), time_dim(1));
599         fprintf('\nTempo (incl. plotting) per n = %-7d  :
                %.2f  s\n', ...
600         n_NelderMead(2), time_dim(2));
```

```
601         fprintf('\nTempo (incl. plotting) per n = %-7d  :
               %.2f  s\n', ...
602         n_NelderMead(3), time_dim(3));
603         fprintf('\nTempo TOTALE (tutte le dimensioni) :  %.2f
                 s\n', time_total);
604
605         %---  bar chart  ---
606         figure;
607         bar(categorical(string(n_NelderMead)), time_dim);
608         ylabel('Tempo (s)'); grid on;
```

Listing 5: Full script: Nelder Mead Method on all function