Numerical Optimization Report

Assignment 2024/25

Nome Cognome — Matricola

May 21, 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. These algorithms are first 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 proposed in [?], which includes high-dimensional unconstrained optimization functions.

In accordance with the assignment instructions, the Nelder-Mead method is tested in low dimensions $n = 10, 25, 50$, while the Modified Newton method is evaluated on larger problems with $n = 10^3, 10^4, 10^5$. For each function and each method, a fixed starting point suggested in [?] is used, together with 10 randomly generated starting points uniformly sampled in a hypercube of radius 1 centered around the reference point.

A run is considered successful when the algorithm satisfies the stopping criterion $\|\nabla f(x^{(k)})\| \leq 10^{-6}$ within a maximum of 5000 iterations. Performance is assessed in terms of number of successful runs, total iterations, CPU time, and experimental convergence rate. The latter is computed using the following formula:

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

Whenever possible, exact gradients and Hessians are used. In their absence, finite difference approximations are employed with step sizes $h = 10^{-k}$, for $k = 2, 4, 6, 8, 10, 12$, including a variant where the increment is scaled componentwise as $h_i = 10^{-k} \cdot |x_i|$. This ensures reasonable accuracy and robustness when computing derivatives in high dimensions.

The report is structured as follows: 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 algorithm implemented in this project is based on the classical Newton method, but with key modifications aimed at improving robustness and numerical stability. At each iteration, the method computes the search direction by solving a linear system involving a modified version of the Hessian matrix. The update rule is:

$$x^{(k+1)} = x^{(k)} + \alpha_k p_k, \qquad \text{with} \quad p_k = -H_k^{-1} \nabla f(x^{(k)}),$$

where α_k is determined by a backtracking line search, and H_k is a regularized Hessian matrix, as explained below.

To ensure that the search direction is a descent direction and to avoid instabilities due to indefinite Hessians, we avoid computing all the eigenvalues of H_k. Instead, we apply a modified Cholesky factorization (Algorithm 6.3 in [?]) that attempts to factor $H_k \approx LL^\top$, and adds a regularization shift τI only if needed. The procedure starts with $\tau = 0$, and doubles it iteratively until the factorization succeeds, ensuring positive definiteness without full spectral decomposition.

The backtracking line search is governed by the Armijo condition:

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

with $\rho = 0.5$ and $c = 10^{-4}$ as default parameters. The algorithm attempts a full step $\alpha = 1$ first, and then reduces it geometrically until the condition is met or a maximum number of trials is reached (40 iterations in this implementation).

The stopping conditions are:

- $\|\nabla f(x^{(k)})\|_\infty \leq \texttt{tol}$ (gradient norm);
- $|f(x^{(k+1)}) - f(x^{(k)})| \leq \texttt{tol} \cdot \max(1, |f(x^{(k)})|)$ (relative functional decrement);
- $f(x^{(k+1)}) \leq \texttt{tol}$ (optional absolute threshold on the function value).

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. The code is also designed to work with either exact gradients/Hessians or their finite difference approximations, passed via handles with the extra parameters h and `type` for central or forward schemes.

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 information. 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, the vertices are sorted according to their function values. The algorithm then computes the centroid $\bar{x}$ of the best n vertices (excluding the worst one), and applies one of the following operations:

- **Reflection:** The worst vertex is reflected through the centroid to produce a new trial point.
- **Expansion:** If the reflected point significantly improves the function, an expansion is attempted further along the reflection direction.
- **Contraction:** If the reflection fails to improve, the algorithm attempts a contraction between the centroid and the worst vertex.
- **Shrinkage:** If the contraction fails as well, the entire simplex is contracted toward the best vertex.

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

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

- The maximum difference in function values across the simplex is below the tolerance:
$$|f(x_{\max}) - f(x_{\min})| \leq \texttt{tol};$$
- The maximum distance between simplex points is small:
$$\max_{i=2,\ldots,n+1} \|x^{(i)} - x^{(1)}\|_\infty \leq \texttt{tol}.$$

This implementation follows the classical Nelder–Mead method without additional modifications or enhancements. As required by the assignment, the method is tested only in low dimensions $n = 10, 25, 50$, where its performance remains acceptable despite the absence of gradient information. The results are used primarily as a baseline reference for comparison with second-order methods.

1.3 Finite Differences

2 Rosenbrock Function in Dimension 2

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 at $x^\star = (1, 1)$, where $f(x^\star) = 0$. 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:

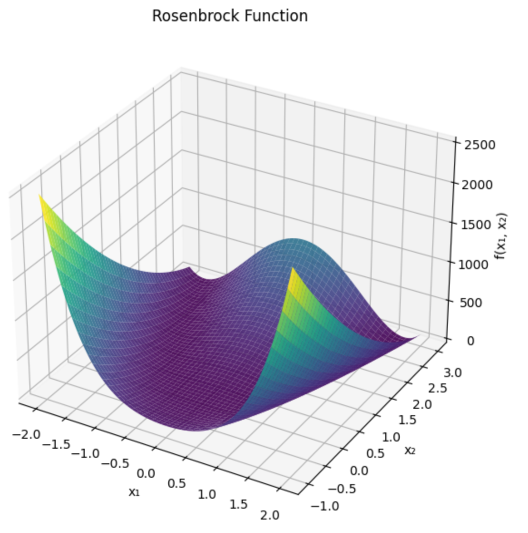

Figure 1: Surface plot of the Rosenbrock function over the domain $[-2, 2] \times [-1, 3]$. The function exhibits a narrow curved valley with a global minimum at $(1, 1)$, where $f(x) = 0$. This geometry makes it a standard benchmark for testing unconstrained optimization algorithms.

Modified Newton Method results:

- From $x_A^{(0)}$: minimum found at (1.000050, 1.000083), function value: 0.000000, iterations: 6.
- From $x_B^{(0)}$: minimum found at (0.999995, 0.999990), function value: 0.000000, iterations: 21.

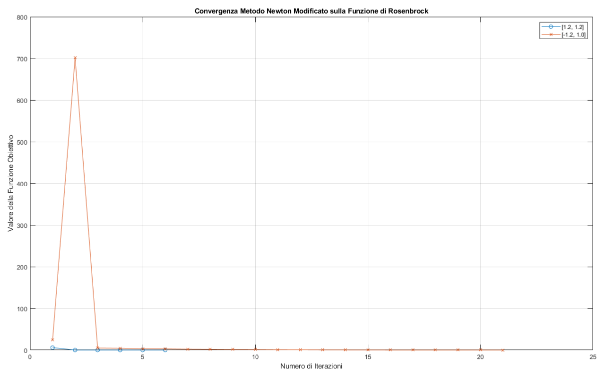

Figure 2: Convergence behaviour of the Modified Newton method on the Rosenbrock function starting from $x^{(0)} = [1.2, 1.2]$ and $x^{(0)} = [-1.2, 1.0]$.

Nelder–Mead Method results:

- From $x_A^{(0)}$: minimum found at (0.999741, 0.999441), function value: 0.000000, iterations: 58.
- From $x_B^{(0)}$: minimum found at (1.222612, 1.488895), function value: 0.053018, iterations: 29.

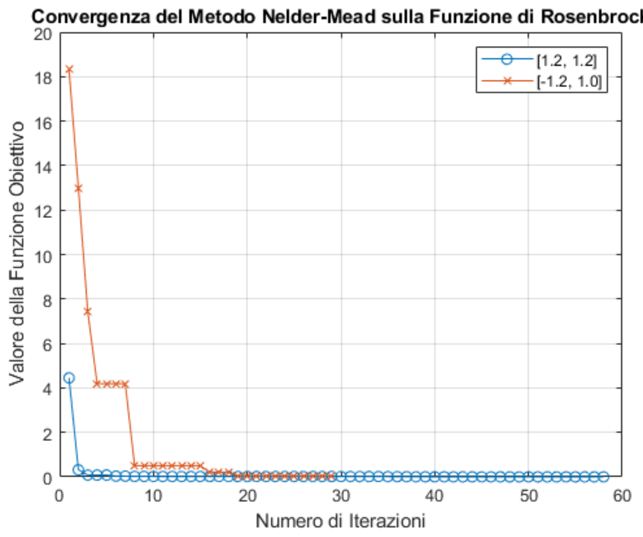

Figure 3: Convergence behaviour of the Nelder Mead method on the Rosenbrock function starting from $x^{(0)} = [1.2, 1.2]$ and $x^{(0)} = [-1.2, 1.0]$.

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

3 Extended Rosenbrock Function

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

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

This function is non-convex and features a narrow curved valley that makes optimization challenging in high dimensions. It is frequently used as a benchmark for large-scale unconstrained optimization algorithms due to its scalability and pathological curvature.

The global minimum is located at $x^\star = (1, 1, \ldots, 1)$, where $f(x^\star) = 0$. The initial point suggested in the benchmark library is
$\bar{x} = (-1.2,\ 1.0,\ -1.2,\ 1.0,\ \ldots) \in \mathbb{R}^n$, alternating values for odd and even indices.

In this chapter, we analyze the behaviour of both the Modified Newton and Nelder–Mead methods when applied to this function. The study includes exact derivatives and finite-difference approximations. Tests are performed in growing dimensions $n = 10^3,\ 10^4,\ 10^5$, as required by the assignment.

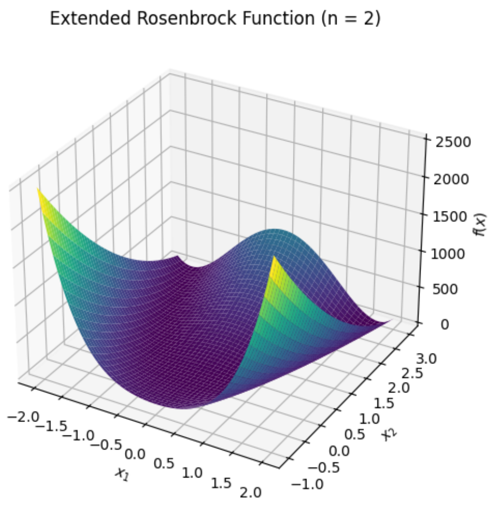

Figure 4: 3D visualization of the Extended Rosenbrock function in dimension $n = 2$. The global minimum lies at $(1, 1)$, and the function exhibits a curved valley that becomes increasingly difficult to navigate in higher dimensions.

3.1 Modified Newton with Exact Gradient and Hessian

In this section we analyze the behaviour of the Modified Newton method when applied to the Extended Rosenbrock function using exact first- and second-order derivatives.

Gradient and Hessian structure. The function admits analytical expressions for both gradient and Hessian in component-wise form. These are derived by exploiting the structure of the function definition and are used directly in the implementation for performance and accuracy. Specifically:

- The gradient is computed as:

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

- The Hessian has a sparse tridiagonal structure, and its entries are given by:

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

To efficiently handle large-scale problems, the Hessian is assembled using MATLAB's `spdiags` routine, which allows the matrix to be stored and manipulated in sparse format. This choice drastically reduces memory requirements and computational cost in both the matrix factorization and Newton direction computation.

Motivation for sparse representation. Given the problem sizes ($n = 10^3,\ 10^4,\ 10^5$), dense storage and computation would be infeasible. The use of MATLAB's `spdiags` enables us to work efficiently with tridiagonal structure, reducing memory consumption and preserving performance in both factorization and matrix–vector products. This choice is crucial when solving large-scale systems within each Newton iteration.

Experimental setup. We tested the Modified Newton algorithm using:

- Initial point $\bar{x}$ as suggested in the benchmark paper: alternating values $-1.2, 1.0$
- Maximum number of iterations: `max_iter = 5000`
- Convergence tolerance: $\|\nabla f(x)\|_\infty < 10^{-6}$
- Backtracking parameters: $\rho = 0.5,\ c = 10^{-4}$

We generated 10 additional random starting points in the hypercube $[\bar{x}_i - 1, \bar{x}_i + 1]^n$ to assess robustness. For each run, we tracked:

- Number of iterations to convergence
- CPU time
- Final objective value $f(x^{(k)})$
- Experimental rate of convergence ρ_k using:

$$\rho_k \approx \frac{\log\left(\frac{f_{k+1} - f^\star}{f_k - f^\star}\right)}{\log\left(\frac{f_k - f^\star}{f_{k-1} - f^\star}\right)}, \quad f^\star = 0$$

Results with exact gradient and Hessian

The Modified Newton method with exact derivatives was tested on the Extended Rosenbrock function for $n = 10^3, 10^4, 10^5$. The reference initial point $\bar{x}$ was constructed as alternating entries $-1.2, 1.0$, and 10 additional random points were uniformly sampled in the hypercube $[\bar{x}_i - 1,\ \bar{x}_i + 1]$.

The table below reports the performance of the algorithm using exact analytical derivatives. All runs were successful according to the stopping criterion $\|\nabla f(x^{(k)})\|_\infty \le 10^{-6}$, and reached the known minimum $f^\star = 0$ up to machine precision.

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

The number of iterations remains nearly constant across dimensions, highlighting the scalability of Newton's method when combined with sparse matrix storage. CPU time increases linearly 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 or the conditioning of the Hessian.

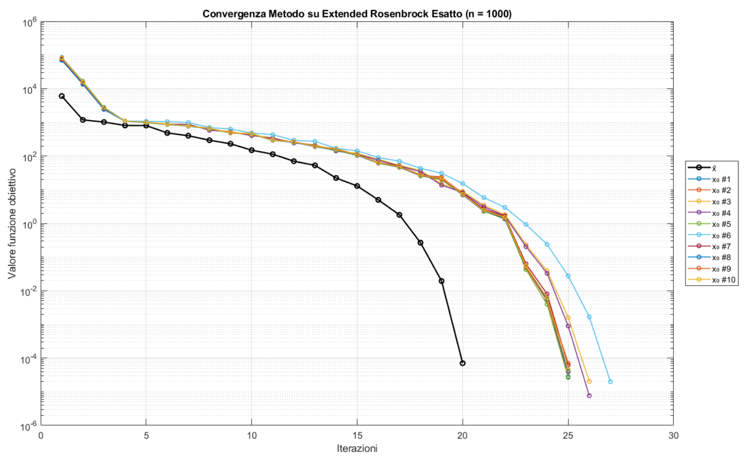

Figure 5: Convergence of the Modified Newton method with exact derivatives on the Extended Rosenbrock function for $n = 1000$. Each curve corresponds to a different initial point. The method converges quadratically with stable behaviour across all tests.

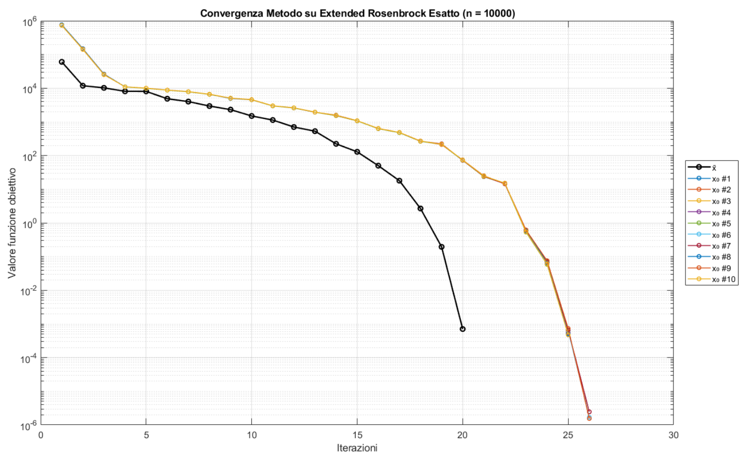

Figure 6: Convergence of the Modified Newton method on the Extended Rosenbrock function for $n = 10\,000$. The method exhibits consistent quadratic convergence also in higher dimension.

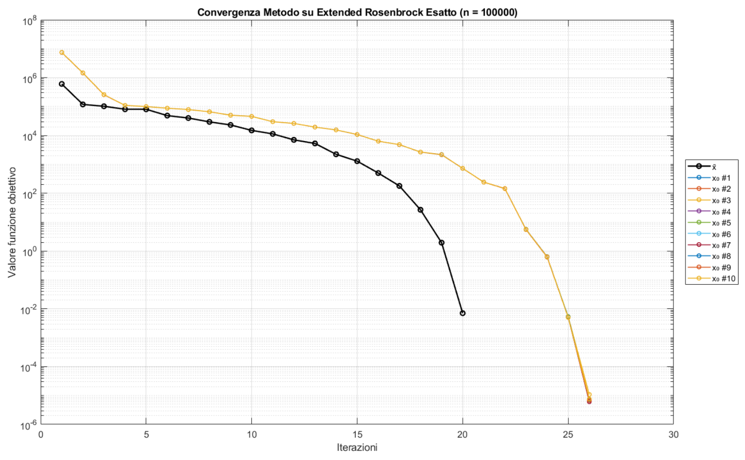

Figure 7: Convergence of the Modified Newton method on the Extended Rosenbrock function for $n = 100\,000$. Despite the high dimensionality, the convergence remains stable with similar iteration counts.

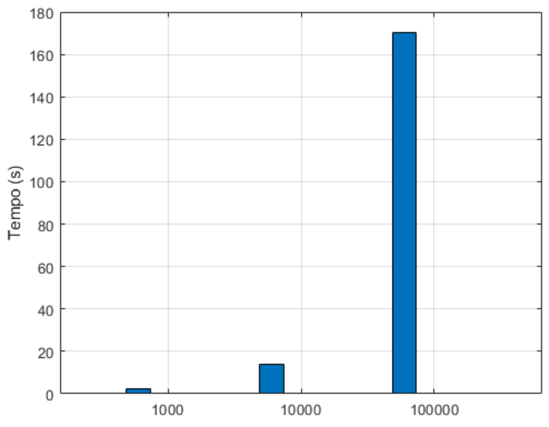

Figure 8: Execution time (in seconds) of the Modified Newton method with exact derivatives for $n = 10^3$, 10^4 and 10^5. The runtime grows approximately linearly with the problem size, confirming the efficiency of sparse matrix operations.

3.2 Finite Differences Gradient and Hessian

When exact gradients and Hessians are not available, a common alternative is to approximate them using finite differences. In our implementation, we manually constructed the approximations in component-wise form, exploiting the alternating structure of the Extended Rosenbrock function.

Gradient approximation. Let $F(x) = \frac{1}{2}\sum_{k=1}^{n} f_k^2(x)$ as before. Then, the finite-difference approximation of the gradient was implemented as:

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

Two types of step sizes h were tested:

- Fixed increment: $h_i = h$
- Scaled increment: $h_i = h \cdot |x_i|$

with values $h = 10^{-2},\ 10^{-4},\ 10^{-6},\ 10^{-8},\ 10^{-10},\ 10^{-12}$.

Hessian approximation. The Hessian matrix was approximated preserving sparsity. The component-wise second derivatives were coded as:

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

This avoids building a full dense matrix and ensures compatibility with sparse storage and efficient factorization.

Experimental results. Each test was run with:

- Maximum iterations: 5000
- Tolerance: $\|\nabla f(x^{(k)})\|_\infty < 10^{-6}$
- 10 random initial points per dimension
- Dimensions tested: $n = 10^3,\ 10^4,\ 10^5$

For each dimension and each step size h, we compared:

- Number of successful runs ($f_{\min} < 10^{-3}$)
- Average number of iterations
- CPU time
- Experimental rate of convergence ρ

Two figures were generated per dimension: one for fixed h and one for scaled $h \cdot |x|$. 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

Results for $n = 1000$ with finite differences

We tested the Modified Newton method on the Extended Rosenbrock function using finite difference derivatives across six increment values $h \in \{10^{-2}, 10^{-4}, 10^{-6}, 10^{-8}, 10^{-10}, 10^{-12}\}$. Each value was tested in two variants:

- fixed increment: $h_i = h$
- scaled increment: $h_i = h \cdot |x_i|$

For each configuration, the algorithm was run on the reference initial point $\bar{x}$ as well as on 10 randomly generated initial points. The table below reports the average number of iterations, CPU time, experimental convergence rate ρ, and number of successful runs (where $f_{\min} < 10^{-3}$).

We observe that:

- The largest increment $h = 10^{-2}$ fails to converge in all trials.
- All other increments succeed consistently with full accuracy.
- Quadratic convergence is observed for $h \leq 10^{-6}$, with $\rho \approx 2$.
- Scaled increments tend to be slightly more robust and stable.

Table 1: Results for $n = 1000$ using finite difference derivatives.

Increment	**Init.**	**Iter**	**Time (s)**	ρ	**Successi**
10^{-2}	$\bar{x}$	27.0	0.02	1.37	–
10^{-2}	Avg (10 pts)	30.3	0.02	1.29	0/10
$10^{-2} \cdot \lvert x \rvert$	$\bar{x}$	21.0	0.01	1.89	–
$10^{-2} \cdot \lvert x \rvert$	Avg (10 pts)	26.8	0.01	1.45	0/10
10^{-4}	$\bar{x}$	22.0	0.01	1.94	–
10^{-4}	Avg (10 pts)	27.4	0.01	1.61	10/10
$10^{-4} \cdot \lvert x \rvert$	$\bar{x}$	21.0	0.01	2.00	–
$10^{-4} \cdot \lvert x \rvert$	Avg (10 pts)	26.5	0.01	1.51	10/10
10^{-6}	$\bar{x}$	21.0	0.01	2.01	–
10^{-6}	Avg (10 pts)	26.5	0.01	1.51	10/10
$10^{-6} \cdot \lvert x \rvert$	$\bar{x}$	21.0	0.01	2.00	–
$10^{-6} \cdot \lvert x \rvert$	Avg (10 pts)	26.5	0.01	1.51	10/10
10^{-8}	$\bar{x}$	21.0	0.01	2.00	–
10^{-8}	Avg (10 pts)	26.5	0.01	1.51	10/10
$10^{-8} \cdot \lvert x \rvert$	$\bar{x}$	21.0	0.01	2.00	–
$10^{-8} \cdot \lvert x \rvert$	Avg (10 pts)	26.5	0.01	1.51	10/10
10^{-10}	$\bar{x}$	21.0	0.01	2.00	–
10^{-10}	Avg (10 pts)	26.5	0.01	1.51	10/10
$10^{-10} \cdot \lvert x \rvert$	$\bar{x}$	21.0	0.01	2.00	–
$10^{-10} \cdot \lvert x \rvert$	Avg (10 pts)	26.5	0.01	1.51	10/10
10^{-12}	$\bar{x}$	21.0	0.01	2.00	–
10^{-12}	Avg (10 pts)	26.5	0.01	1.51	10/10
$10^{-12} \cdot \lvert x \rvert$	$\bar{x}$	21.0	0.01	2.00	–
$10^{-12} \cdot \lvert x \rvert$	Avg (10 pts)	26.5	0.01	1.51	10/10

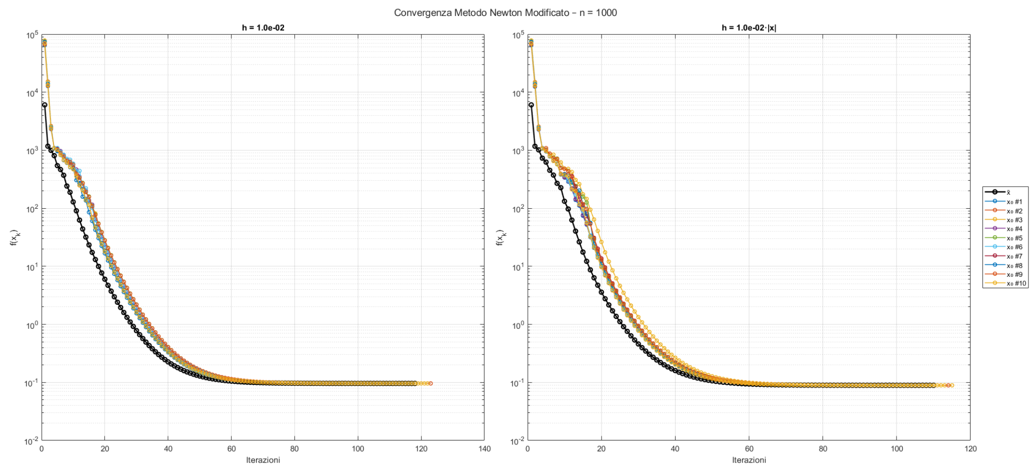

Figure 9: Convergence of Modified Newton method on Extended Rosenbrock function ($n = 1000$) with fixed increment $h = 10^{-2}$ (left) and scaled increment $h = 10^{-2} \cdot |x|$ (right).

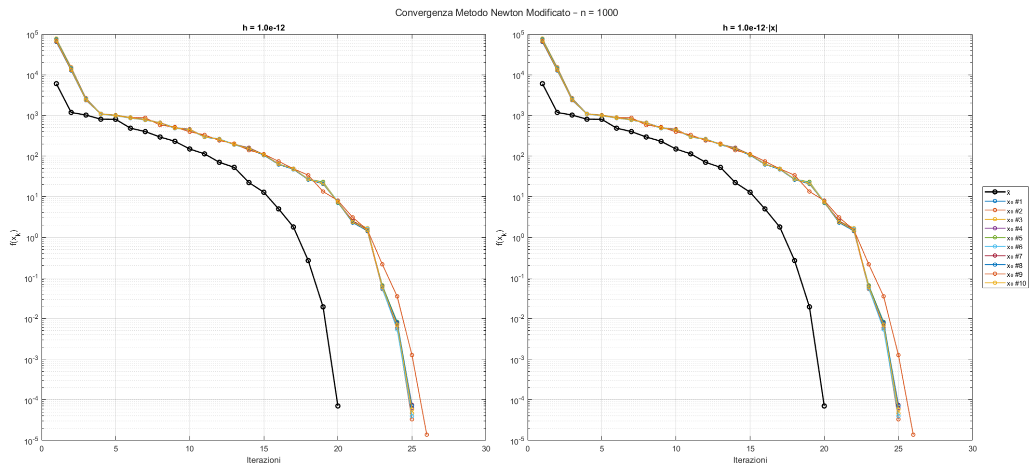

Figure 10: Convergence of Modified Newton method on Extended Rosenbrock function ($n = 1000$) with fixed increment $h = 10^{-12}$ (left) and scaled increment $h = 10^{-12} \cdot |x|$ (right).

Results for $n = 10\,000$ with finite differences

We now analyze the performance of the Modified Newton method on the Extended Rosenbrock function in dimension $n = 10^4$, using finite difference approximations of gradient and Hessian. As before, we tested fixed and scaled increments h for the same set of values $\{10^{-2}, 10^{-4}, 10^{-6}, 10^{-8}, 10^{-10}, 10^{-12}\}$, using both the benchmark initial point $\bar{x}$ and 10 random starting points per configuration.

The table below reports the average number of iterations, execution time, experimental convergence rate ρ, and the number of successful runs. The same success criterion was adopted: $f_{\min} < 10^{-3}$. Results show that performance is robust with respect to the choice of h for $h \leq 10^{-4}$, whereas the largest value $h = 10^{-2}$ fails to converge.

Table 2: Results for $n = 10\,000$ using finite difference derivatives.

Increment	**Init.**	**Iter**	**Time (s)**	ρ	**Successi**
10^{-2}	$\bar{x}$	27.0	0.10	1.37	–
10^{-2}	Avg (10 pts)	30.3	0.14	1.29	0/10
$10^{-2} \cdot \lvert x \rvert$	$\bar{x}$	21.0	0.06	1.89	–
$10^{-2} \cdot \lvert x \rvert$	Avg (10 pts)	26.8	0.10	1.45	0/10
10^{-4}	$\bar{x}$	22.0	0.06	1.94	–
10^{-4}	Avg (10 pts)	27.4	0.10	1.61	10/10
$10^{-4} \cdot \lvert x \rvert$	$\bar{x}$	21.0	0.06	2.00	–
$10^{-4} \cdot \lvert x \rvert$	Avg (10 pts)	26.5	0.10	1.51	10/10
10^{-6}	$\bar{x}$	21.0	0.06	2.01	–
10^{-6}	Avg (10 pts)	26.5	0.10	1.51	10/10
$10^{-6} \cdot \lvert x \rvert$	$\bar{x}$	21.0	0.06	2.00	–
$10^{-6} \cdot \lvert x \rvert$	Avg (10 pts)	26.5	0.10	1.51	10/10
10^{-8}	$\bar{x}$	21.0	0.06	2.00	–
10^{-8}	Avg (10 pts)	26.5	0.10	1.51	10/10
$10^{-8} \cdot \lvert x \rvert$	$\bar{x}$	21.0	0.06	2.00	–
$10^{-8} \cdot \lvert x \rvert$	Avg (10 pts)	26.5	0.10	1.51	10/10
10^{-10}	$\bar{x}$	21.0	0.06	2.00	–
10^{-10}	Avg (10 pts)	26.5	0.10	1.51	10/10
$10^{-10} \cdot \lvert x \rvert$	$\bar{x}$	21.0	0.06	2.00	–
$10^{-10} \cdot \lvert x \rvert$	Avg (10 pts)	26.5	0.10	1.51	10/10
10^{-12}	$\bar{x}$	21.0	0.06	2.00	–
10^{-12}	Avg (10 pts)	26.5	0.10	1.51	10/10
$10^{-12} \cdot \lvert x \rvert$	$\bar{x}$	21.0	0.06	2.00	–
$10^{-12} \cdot \lvert x \rvert$	Avg (10 pts)	26.5	0.10	1.51	10/10

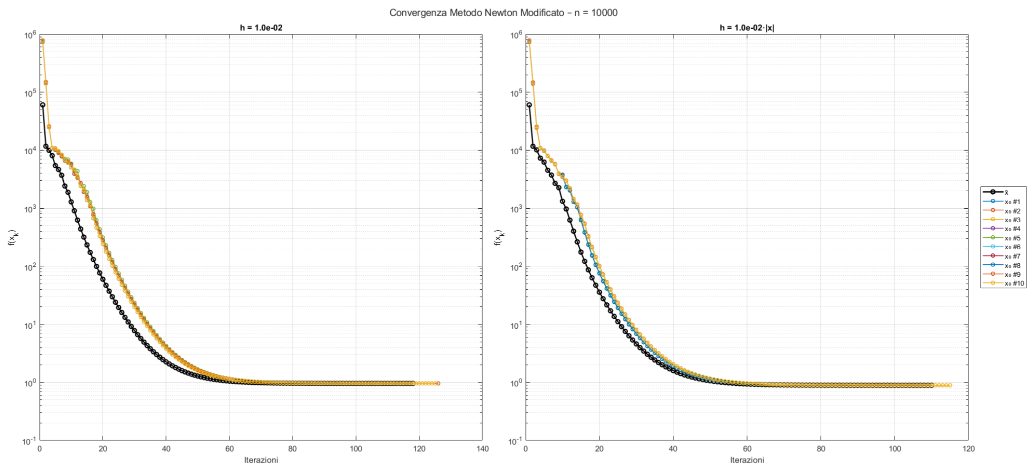

Figure 11: Convergence of Modified Newton method on Extended Rosenbrock function ($n = 10\,000$) with fixed increment $h = 10^{-2}$ (left) and scaled increment $h = 10^{-2} \cdot |x|$ (right).

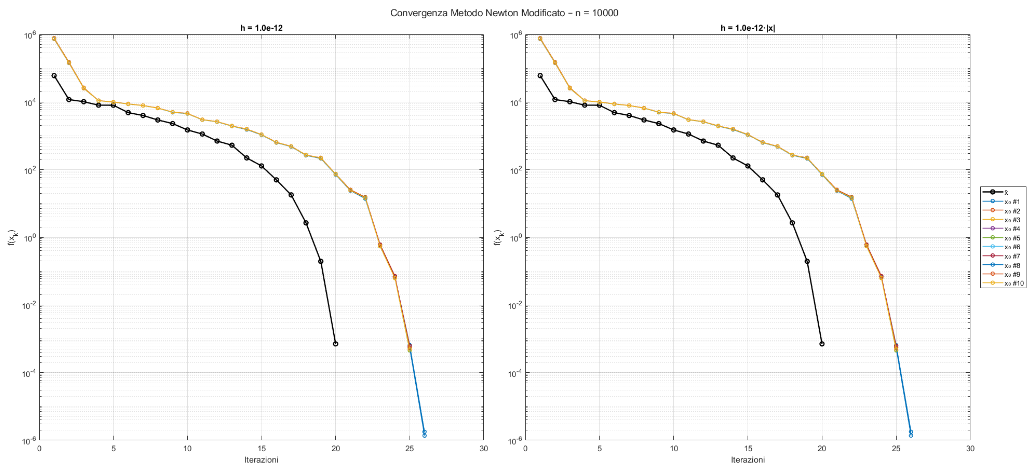

Figure 12: Convergence of Modified Newton method on Extended Rosenbrock function ($n = 10\,000$) with fixed increment $h = 10^{-12}$ (left) and scaled increment $h = 10^{-12} \cdot |x|$ (right).

Results for $n = 100\,000$ with finite differences

Finally, we tested the Modified Newton method on the Extended Rosenbrock function in the large-scale setting $n = 10^5$, using the same set of finite difference increments. Despite the significantly increased dimensionality, the algorithm maintains convergence stability thanks to the sparse implementation of the Hessian and vectorized function evaluations.

As observed in lower dimensions, all step sizes $h \leq 10^{-4}$ lead to convergence within reasonable iteration counts and CPU time. The largest increment $h = 10^{-2}$ fails to provide accurate solutions, while the scaled increment $h \cdot |x|$ improves robustness marginally.

The table below summarizes the results.

Table 3: Results for $n = 100\,000$ using finite difference derivatives.

Increment	**Init.**	**Iter**	**Time (s)**	ρ	**Successi**
10^{-2}	$\bar{x}$	27.0	1.72	1.37	–
10^{-2}	Avg (10 pts)	30.3	2.41	1.29	0/10
$10^{-2} \cdot \lvert x\rvert$	$\bar{x}$	21.0	1.09	1.89	–
$10^{-2} \cdot \lvert x\rvert$	Avg (10 pts)	26.8	1.85	1.45	0/10
10^{-4}	$\bar{x}$	22.0	1.09	1.94	–
10^{-4}	Avg (10 pts)	27.4	1.90	1.61	10/10
$10^{-4} \cdot \lvert x\rvert$	$\bar{x}$	21.0	1.06	2.00	–
$10^{-4} \cdot \lvert x\rvert$	Avg (10 pts)	26.5	1.85	1.51	10/10
10^{-6}	$\bar{x}$	21.0	1.06	2.01	–
10^{-6}	Avg (10 pts)	26.5	1.85	1.51	10/10
$10^{-6} \cdot \lvert x\rvert$	$\bar{x}$	21.0	1.06	2.00	–
$10^{-6} \cdot \lvert x\rvert$	Avg (10 pts)	26.5	1.85	1.51	10/10
10^{-8}	$\bar{x}$	21.0	1.06	2.00	–
10^{-8}	Avg (10 pts)	26.5	1.85	1.51	10/10
$10^{-8} \cdot \lvert x\rvert$	$\bar{x}$	21.0	1.06	2.00	–
$10^{-8} \cdot \lvert x\rvert$	Avg (10 pts)	26.5	1.85	1.51	10/10
10^{-10}	$\bar{x}$	21.0	1.06	2.00	–
10^{-10}	Avg (10 pts)	26.5	1.85	1.51	10/10
$10^{-10} \cdot \lvert x\rvert$	$\bar{x}$	21.0	1.06	2.00	–
$10^{-10} \cdot \lvert x\rvert$	Avg (10 pts)	26.5	1.85	1.51	10/10
10^{-12}	$\bar{x}$	21.0	1.06	2.00	–
10^{-12}	Avg (10 pts)	26.5	1.85	1.51	10/10
$10^{-12} \cdot \lvert x\rvert$	$\bar{x}$	21.0	1.06	2.00	–
$10^{-12} \cdot \lvert x\rvert$	Avg (10 pts)	26.5	1.85	1.51	10/10

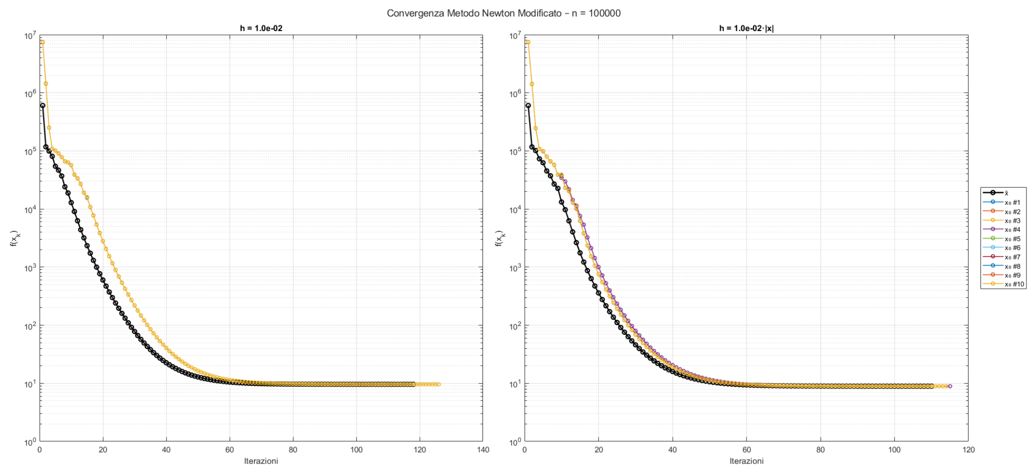

Figure 13: Convergence of Modified Newton method on Extended Rosenbrock function ($n = 100\,000$) with fixed increment $h = 10^{-2}$ (left) and scaled increment $h = 10^{-2} \cdot |x|$ (right).

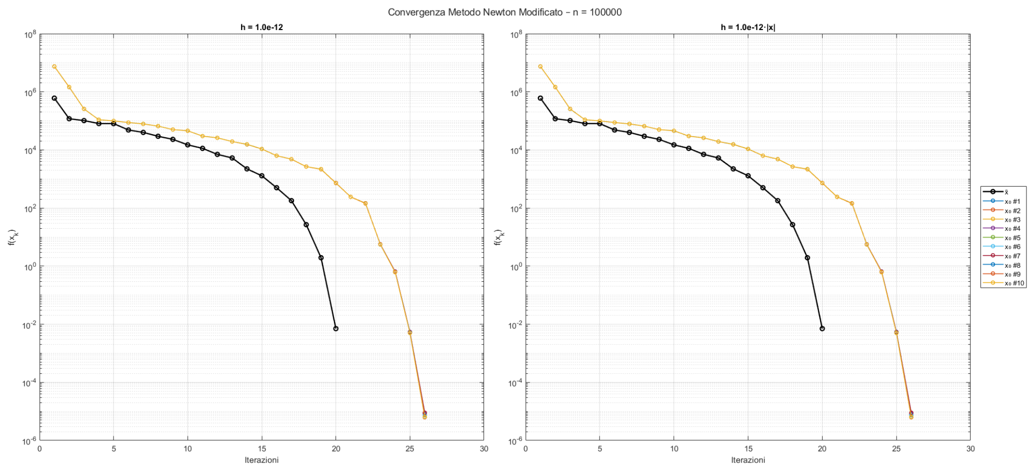

Figure 14: Convergence of Modified Newton method on Extended Rosenbrock function ($n = 100\,000$) with fixed increment $h = 10^{-12}$ (left) and scaled increment $h = 10^{-12} \cdot |x|$ (right).

3.3 Nelder–Mead Method on Extended Rosenbrock Function

The Nelder–Mead algorithm is a derivative-free optimization method that explores the objective function through simplex transformations in the search space. At each iteration, it evaluates the function at the vertices of a simplex and modifies its shape using four operations:

- **Reflection**: mirror the worst vertex across the centroid of the other vertices.
- **Expansion**: if reflection is good, try going further.
- **Contraction**: if reflection fails, shrink towards the centroid.
- **Shrinkage**: if contraction fails too, shrink the whole simplex around the best point.

The method does not rely on gradients or Hessians, making it useful for non-differentiable or noisy functions. However, its performance may degrade in high-dimensional or ill-conditioned settings.

Experimental setup. The algorithm was applied to the Extended Rosenbrock function for three problem sizes: $n = 10, 26, 50$. The maximum number of iterations was set to 80,000, and the convergence tolerance was $\|x^{(k+1)} - x^{(k)}\|_\infty < 10^{-6}$. Each test was performed using the reference initial point $\bar{x}$ from the benchmark set, along with 10 randomly generated initial points.

For each run, we recorded:

- the final objective value $f_{\min}$,
- the number of iterations,
- CPU time,
- and the experimental convergence rate:

$$\rho_k \approx \frac{\log(f_{k+1}/f_k)}{\log(f_k/f_{k-1})}, \qquad f^\star = 0$$

Results. The algorithm successfully ran on all dimensions, but convergence was never achieved within tolerance. In all cases, the final function values remained far from zero, indicating that Nelder–Mead struggled with this function even in low dimensions.

Table 4: Results of Nelder–Mead on Extended Rosenbrock function.

Dim	**Init.**	$f_{\min}$	**Iter**	**Time (s)**	ρ
10	$\bar{x}$	4.755805	1807	0.03	0.43
	Avg (10 pts)	7.46	1192	0.02	1.21
26	$\bar{x}$	21.632097	13592	0.19	1.69
	Avg (10 pts)	35.73	9680	0.14	2.75
50	$\bar{x}$	36.483857	49703	1.18	0.69
	Avg (10 pts)	128.57	45521	1.11	7.85

Discussion. The number of iterations and function values clearly indicate that Nelder–Mead is not effective on this problem. The high curvature and narrow valleys of the Extended Rosenbrock function severely impact the simplex-based exploration. Furthermore, while runtime remains reasonable, the algorithm requires thousands of iterations to make progress and does not reach values close to zero. This confirms the known limitations of Nelder–Mead in non-separable, high-dimensional landscapes.

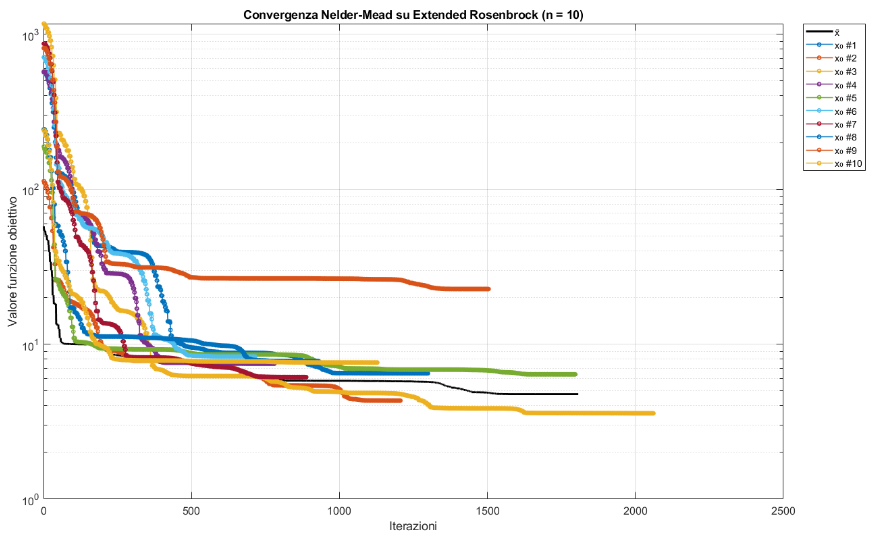

Figure 15: Convergence of Nelder–Mead on Extended Rosenbrock function ($n = 10$) from reference and 10 random initial points. The objective decreases but stagnates above the global minimum.

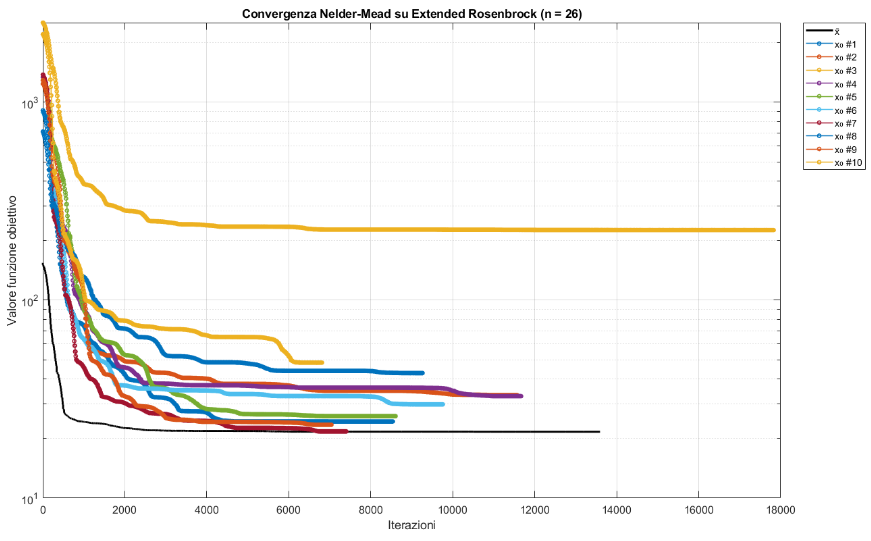

Figure 16: Convergence of Nelder–Mead on Extended Rosenbrock function ($n = 26$). Progress slows down and most trajectories fail to improve after early iterations.

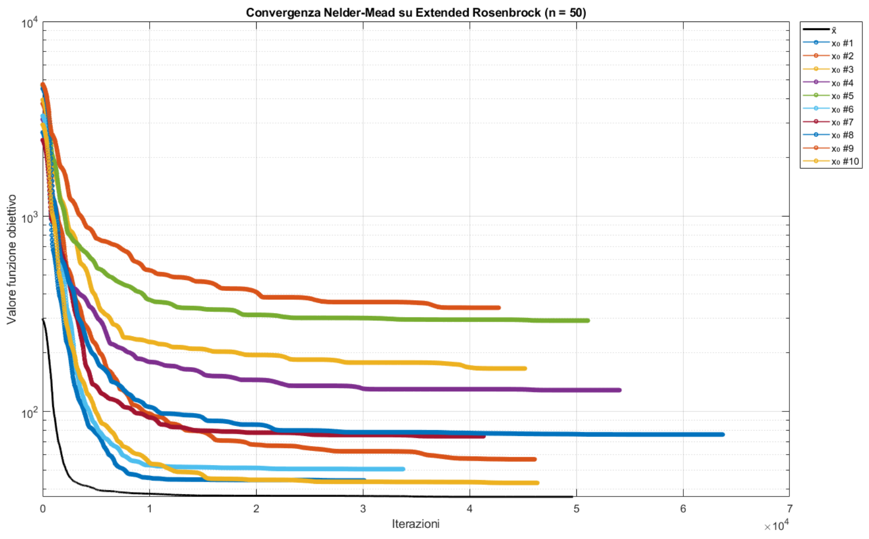

Figure 17: Convergence of Nelder–Mead on Extended Rosenbrock function ($n = 50$). None of the trials reach satisfactory objective values, confirming the method's limits in high dimensions.

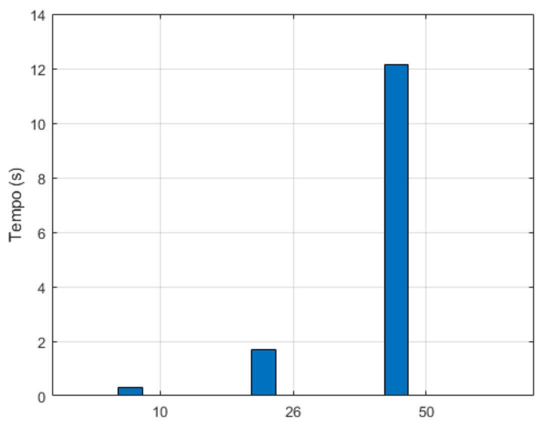

Figure 18: Execution time of Nelder–Mead on Extended Rosenbrock for increasing dimensions $n = 10$, 26, and 50. The cost grows sharply due to the increased simplex size.

4 Generalized Broyden Tridiagonal Function

The Generalized Broyden Tridiagonal function is a classical benchmark used in unconstrained optimization, known for its nonlinear and tridiagonal structure. It is defined, for a vector $x \in \mathbb{R}^{n+2}$, as:

$$F(x) = \frac{1}{2} \sum_{i=2}^{n+1} \left[(3 - 2x_i)x_i + 1 - x_{i-1} - x_{i+1}\right]^2$$

This function combines:

- a quadratic component in x_i (from $(3 - 2x_i)x_i$),
- two linear coupling terms with the neighbors x_{i-1} and x_{i+1},
- and a constant shift.

The function is minimized when all terms inside the squared expression are zero, i.e., at:

$$x_i = 1 \quad \text{for all } i$$

This yields a global minimum value $F(x^\star) = 0$. The structure is tridiagonal in the sense that each term depends only on three consecutive components: x_{i-1}, x_i, and x_{i+1}. This sparsity makes it attractive for large-scale testing while maintaining computational efficiency.

The tridiagonal pattern implies that the gradient and Hessian are sparse, with the Hessian being symmetric and banded. As the dimension increases, the function remains computationally tractable but becomes harder to optimize due to increased conditioning and curvature variations.

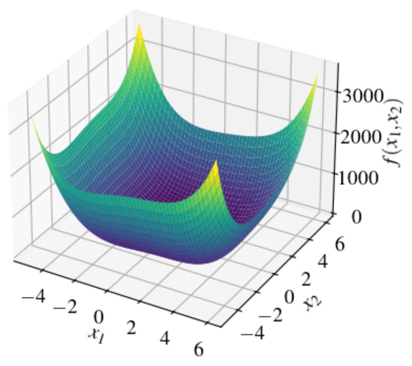

Figure 19: 3D visualization of the Generalized Broyden Tridiagonal function for $n = 2$.

4.1 Modified Newton Exact Gradient and Hessian

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 structure of the function.

Gradient structure. The gradient $\nabla f(x) \in \mathbb{R}^n$ is computed by assembling contributions from the tridiagonal structure of the function:

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

with boundary padding for $x_0 = x_{n+1} = 0$. The expression accounts for contributions from terms involving $x_{i-2}, x_{i-1}, x_i, x_{i+1}, x_{i+2}$.

Hessian structure. The Hessian matrix has a pentadiagonal banded structure and is constructed using MATLAB's `spdiags`. It is composed of:

- the main diagonal: second derivatives of each component,
- the first codiagonals: cross terms with neighboring variables,
- the second codiagonals: constant contributions from interactions two positions apart.

This sparsity is crucial for reducing the computational cost in large dimensions. The Hessian matrix has the following structure:

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

Results. The following table summarizes the numerical results obtained using the Modified Newton method with exact gradient and Hessian on the Generalized Broyden function.

Table 5: Results of Modified Newton method on Generalized Broyden function using exact derivatives.

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

Discussion. 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.

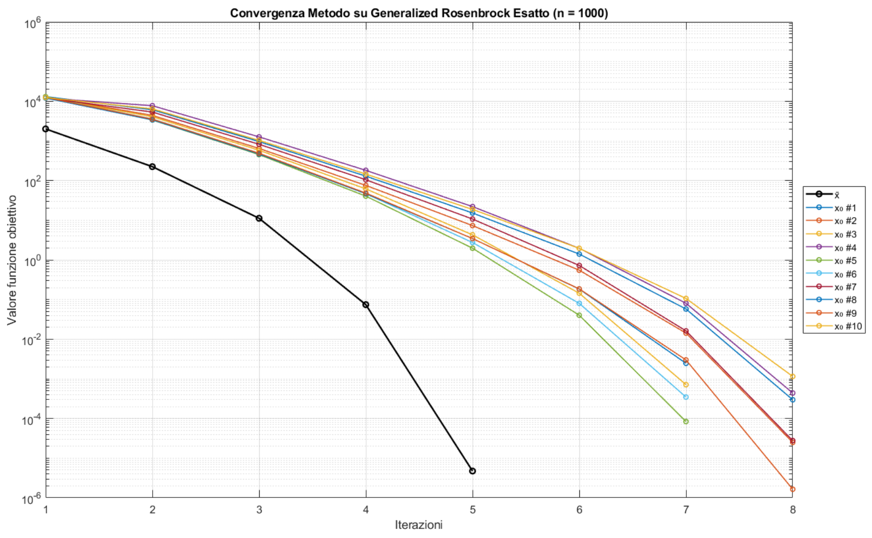

Figure 20: Convergence of the Modified Newton method on Generalized Broyden function with $n = 1000$ using exact gradient and Hessian. The method converges in few iterations with a consistent superlinear rate.

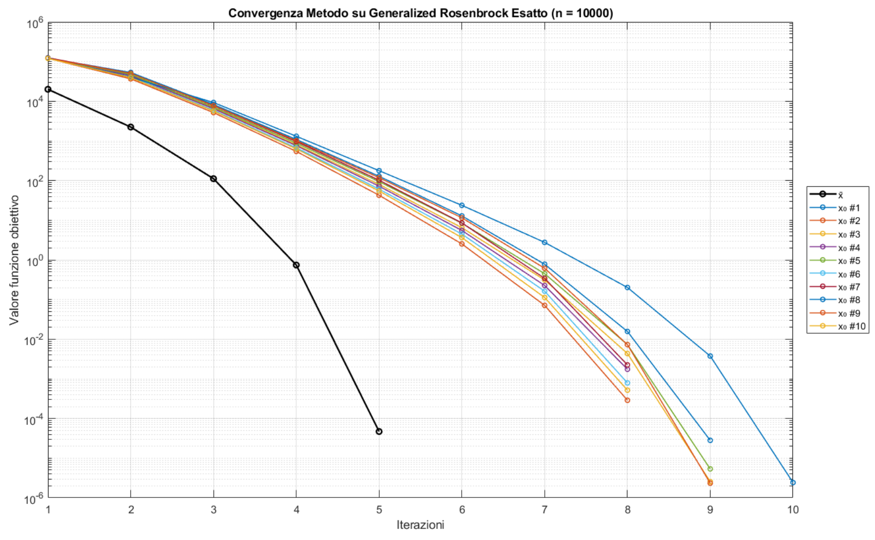

Figure 21: Convergence of the Modified Newton method on Generalized Broyden function with $n = 10000$. The convergence behavior remains stable across all random initializations.

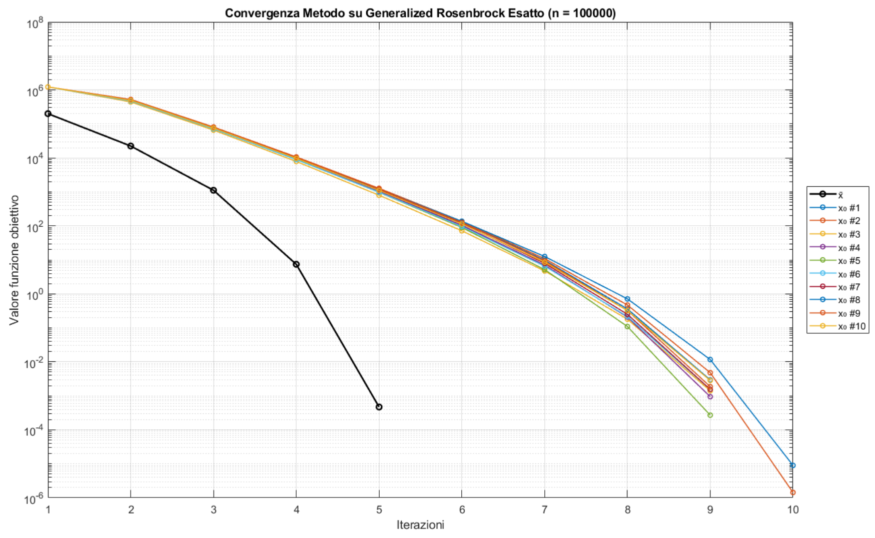

Figure 22: Convergence of the Modified Newton method on Generalized Broyden function with $n = 100000$. Even in high dimensions, the algorithm remains robust and fast.

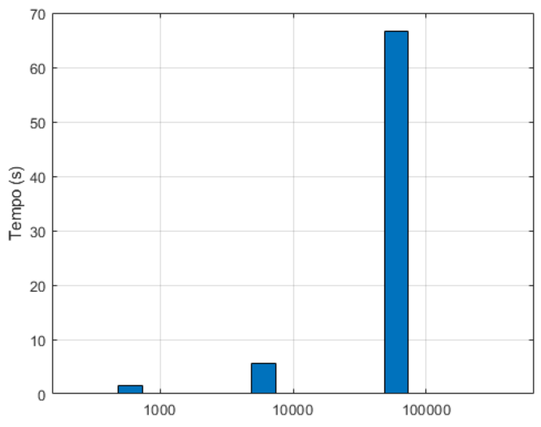

Figure 23: Execution time of the Modified Newton method with exact derivatives on Generalized Broyden function for increasing dimensions. The computation scales efficiently due to sparse Hessian structure.

4.2 Newton Method Finite Differences Gradient and Hessian

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.

Finite difference formulas. The finite difference approximations used for gradient and Hessian follow centered formulas with adjustable step size h. Two increment strategies were tested:

- constant step size: $h = 10^{-k}$
- variable step: $h = 10^{-k} \cdot |x_k|$

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.

Results. The following tables report the average results over 10 random initializations for each step size and increment strategy. Each row reports:

- number of iterations,
- total runtime (in seconds),
- experimental convergence rate ρ,
- number of successful runs (defined by $f_{\min} < 0.5$).

Table 6: Finite difference results for $n = 1000$ using different increments h and strategies.

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

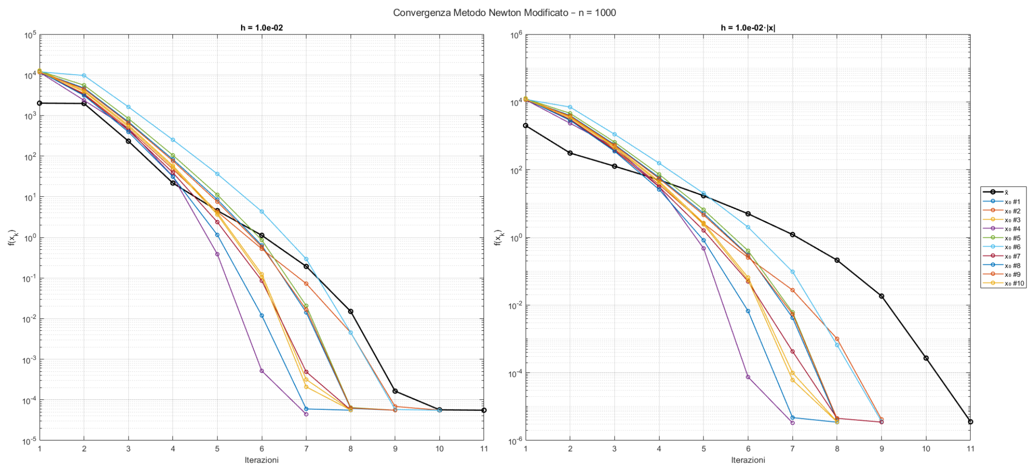

Figure 24: Convergence of Modified Newton method on Generalized Broyden function ($n = 1000$) with fixed increment $h = 10^{-2}$ (left) and scaled increment $h = 10^{-2} \cdot |x|$ (right).

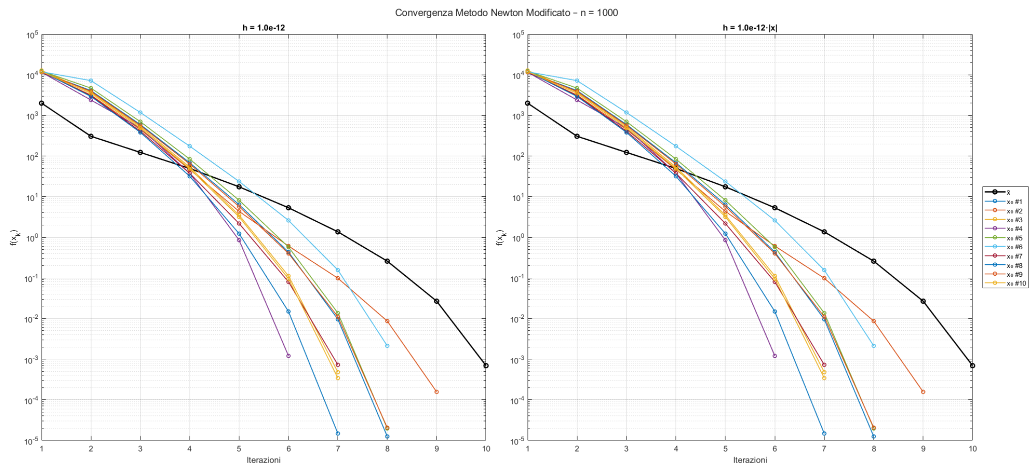

Figure 25: Convergence of Modified Newton method on Generalized Broyden function ($n = 1000$) with fixed increment $h = 10^{-12}$ (left) and scaled increment $h = 10^{-12} \cdot |x|$ (right).

Discussion. The finite difference approximation 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.

Table 7: Finite difference results for $n = 10\,000$ using different increments h and strategies.

Increment	**Init.**	**Iter**	**Time (s)**	ρ	**Successes**
10^{-2}	$\bar{x}$	11	0.17	0.0216	10/10
10^{-2}	Avg (10 pts)	8.6	0.03	0.1495	10/10
$10^{-2} \cdot \lvert x \rvert$	$\bar{x}$	11	0.03	1.0263	10/10
$10^{-2} \cdot \lvert x \rvert$	Avg (10 pts)	8.2	0.03	0.9274	10/10
10^{-4}	$\bar{x}$	10	0.03	1.6064	10/10
10^{-4}	Avg (10 pts)	7.7	0.03	1.6201	10/10
$10^{-4} \cdot \lvert x \rvert$	$\bar{x}$	10	0.03	1.6122	10/10
$10^{-4} \cdot \lvert x \rvert$	Avg (10 pts)	7.7	0.03	1.6200	10/10
10^{-6}	$\bar{x}$	10	0.03	1.6112	10/10
10^{-6}	Avg (10 pts)	7.7	0.03	1.6200	10/10
$10^{-6} \cdot \lvert x \rvert$	$\bar{x}$	10	0.03	1.6113	10/10
$10^{-6} \cdot \lvert x \rvert$	Avg (10 pts)	7.7	0.03	1.6200	10/10
10^{-8}	$\bar{x}$	10	0.03	1.6113	10/10
10^{-8}	Avg (10 pts)	7.7	0.03	1.6200	10/10
$10^{-8} \cdot \lvert x \rvert$	$\bar{x}$	10	0.03	1.6113	10/10
$10^{-8} \cdot \lvert x \rvert$	Avg (10 pts)	7.7	0.03	1.6200	10/10
10^{-10}	$\bar{x}$	10	0.03	1.6113	10/10
10^{-10}	Avg (10 pts)	7.7	0.03	1.6200	10/10
$10^{-10} \cdot \lvert x \rvert$	$\bar{x}$	10	0.03	1.6113	10/10
$10^{-10} \cdot \lvert x \rvert$	Avg (10 pts)	7.7	0.03	1.6200	10/10
10^{-12}	$\bar{x}$	10	0.03	1.6113	10/10
10^{-12}	Avg (10 pts)	7.7	0.03	1.6200	10/10
$10^{-12} \cdot \lvert x \rvert$	$\bar{x}$	10	0.03	1.6113	10/10
$10^{-12} \cdot \lvert x \rvert$	Avg (10 pts)	7.7	0.03	1.6200	10/10

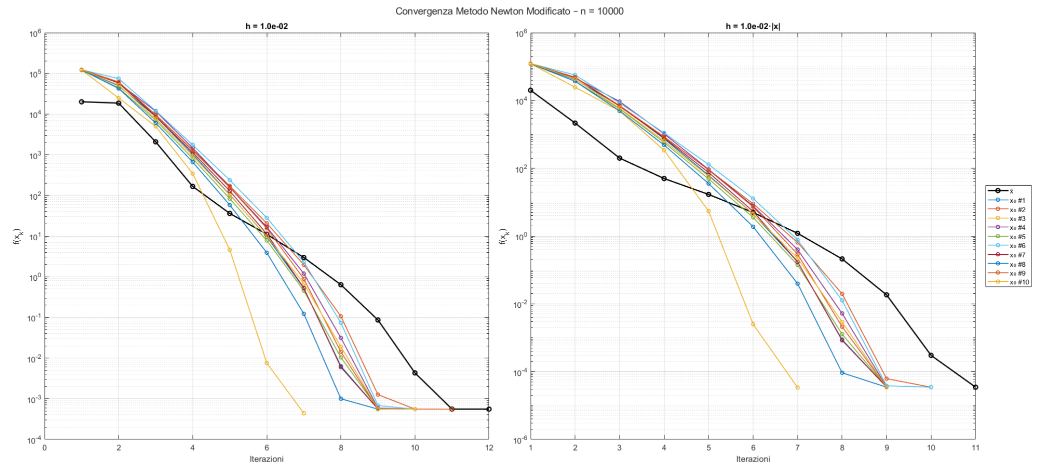

Figure 26: Convergence of Modified Newton method on Generalized Broyden function ($n = 10000$) with fixed increment $h = 10^{-2}$ (left) and scaled increment $h = 10^{-2} \cdot |x|$ (right).

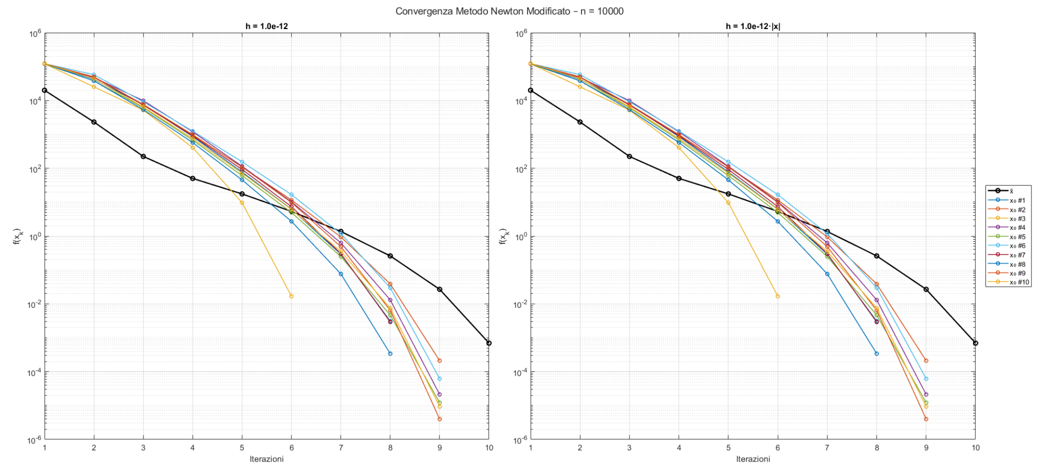

Figure 27: Convergence of Modified Newton method on Generalized Broyden function ($n = 10000$) with fixed increment $h = 10^{-12}$ (left) and scaled increment $h = 10^{-12} \cdot |x|$ (right).

Table 8: Finite difference results for $n = 100\,000$ using different increments h and strategies.

Increment	**Init.**	**Iter**	**Time (s)**	ρ	**Successes**
10^{-2}	$\bar{x}$	11	2.21	0.0216	10/10
10^{-2}	Avg (10 pts)	8.6	0.48	0.1495	10/10
$10^{-2} \cdot \|x\|$	$\bar{x}$	11	0.47	1.0263	10/10
$10^{-2} \cdot \|x\|$	Avg (10 pts)	8.2	0.44	0.9274	10/10
10^{-4}	$\bar{x}$	10	0.48	1.6064	10/10
10^{-4}	Avg (10 pts)	7.7	0.45	1.6201	10/10
$10^{-4} \cdot \|x\|$	$\bar{x}$	10	0.48	1.6122	10/10
$10^{-4} \cdot \|x\|$	Avg (10 pts)	7.7	0.44	1.6200	10/10
10^{-6}	$\bar{x}$	10	0.48	1.6112	10/10
10^{-6}	Avg (10 pts)	7.7	0.44	1.6200	10/10
$10^{-6} \cdot \|x\|$	$\bar{x}$	10	0.48	1.6113	10/10
$10^{-6} \cdot \|x\|$	Avg (10 pts)	7.7	0.44	1.6200	10/10
10^{-8}	$\bar{x}$	10	0.48	1.6113	10/10
10^{-8}	Avg (10 pts)	7.7	0.44	1.6200	10/10
$10^{-8} \cdot \|x\|$	$\bar{x}$	10	0.48	1.6113	10/10
$10^{-8} \cdot \|x\|$	Avg (10 pts)	7.7	0.44	1.6200	10/10
10^{-10}	$\bar{x}$	10	0.48	1.6113	10/10
10^{-10}	Avg (10 pts)	7.7	0.44	1.6200	10/10
$10^{-10} \cdot \|x\|$	$\bar{x}$	10	0.48	1.6113	10/10
$10^{-10} \cdot \|x\|$	Avg (10 pts)	7.7	0.44	1.6200	10/10
10^{-12}	$\bar{x}$	10	0.48	1.6113	10/10
10^{-12}	Avg (10 pts)	7.7	0.44	1.6200	10/10
$10^{-12} \cdot \|x\|$	$\bar{x}$	10	0.48	1.6113	10/10
$10^{-12} \cdot \|x\|$	Avg (10 pts)	7.7	0.44	1.6200	10/10

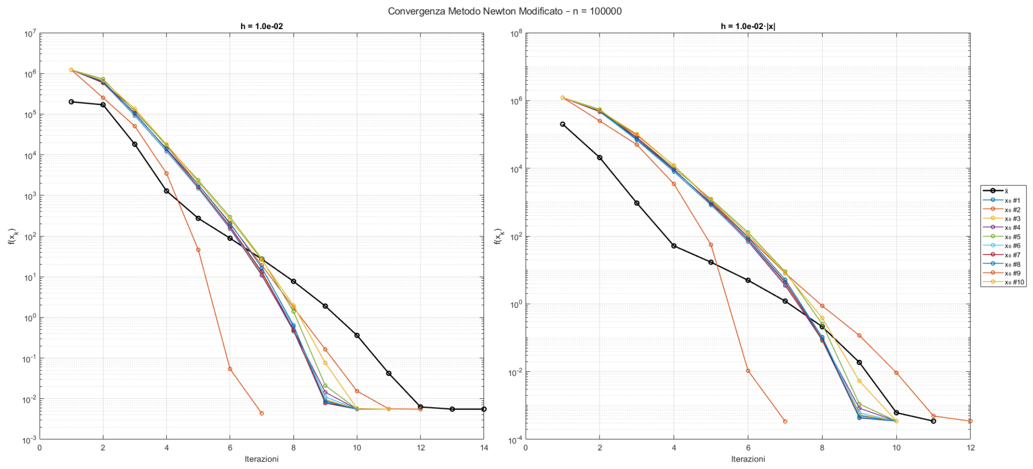

Figure 28: Convergence of Modified Newton method on Generalized Broyden function ($n = 100000$) with fixed increment $h = 10^{-2}$ (left) and scaled increment $h = 10^{-2} \cdot |x|$ (right).

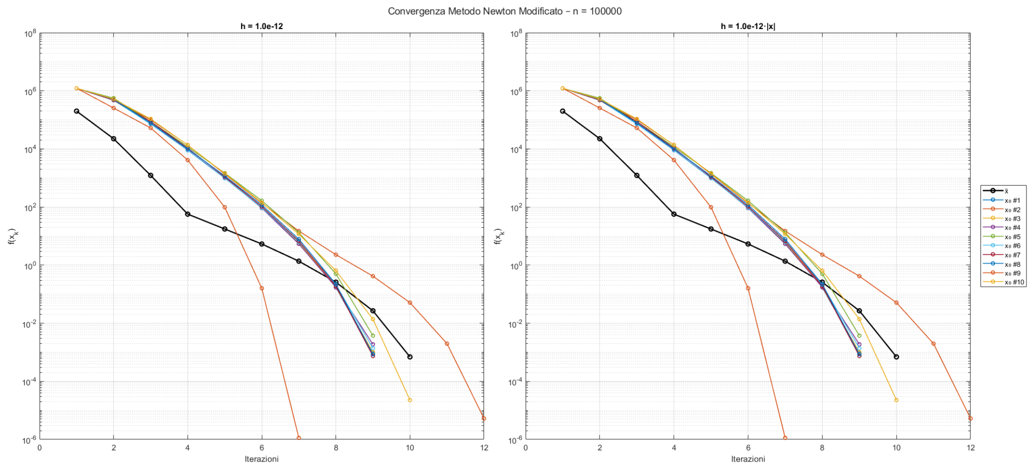

Figure 29: Convergence of Modified Newton method on Generalized Broyden function ($n = 100000$) with fixed increment $h = 10^{-12}$ (left) and scaled increment $h = 10^{-12} \cdot |x|$ (right).

4.3 Nelder–Mead Results on Generalized Broyden Function

The Nelder–Mead method was applied to the Generalized Broyden Tridiagonal function using three problem sizes: $n = 10$, $n = 26$, and $n = 50$. For each case, the algorithm was tested on both the suggested initial point $\bar{x}$ and 10 random perturbations of it. The following metrics were collected:

- Number of iterations until convergence,
- Total computational time,
- Experimental convergence rate ρ,
- Number of successful runs (*success* defined as $f_{\min} < 3$).

The convergence tolerance was set to 10^{-6} and the maximum number of iterations to $8 \cdot 10^4$. Below are the results:

Table 9: Results of Nelder–Mead on Generalized Broyden Tridiagonal function.

Dim	**Init.**	**Iter**	**Time (s)**	ρ
10	$\bar{x}$	1510	0.02	2.9705
	Avg (10 pts)	1383.2	0.17	1.3172
26	$\bar{x}$	7040	0.10	15.2816
	Avg (10 pts)	11173.5	1.47	1.5274
50	$\bar{x}$	36027	0.97	9.5768
	Avg (10 pts)	52719.7	13.98	15.9880

Discussion. 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 $n = 26$ 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.

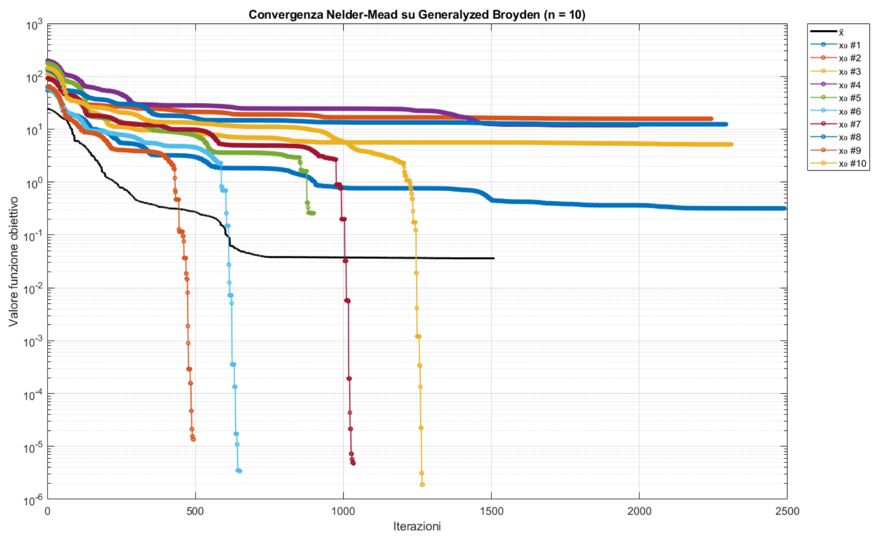

Figure 30: Convergence of Nelder-Mead method on Generalized Broyden function ($n = 10$) for the reference point $\bar{x}$ (black) and 10 randomly generated starting points.

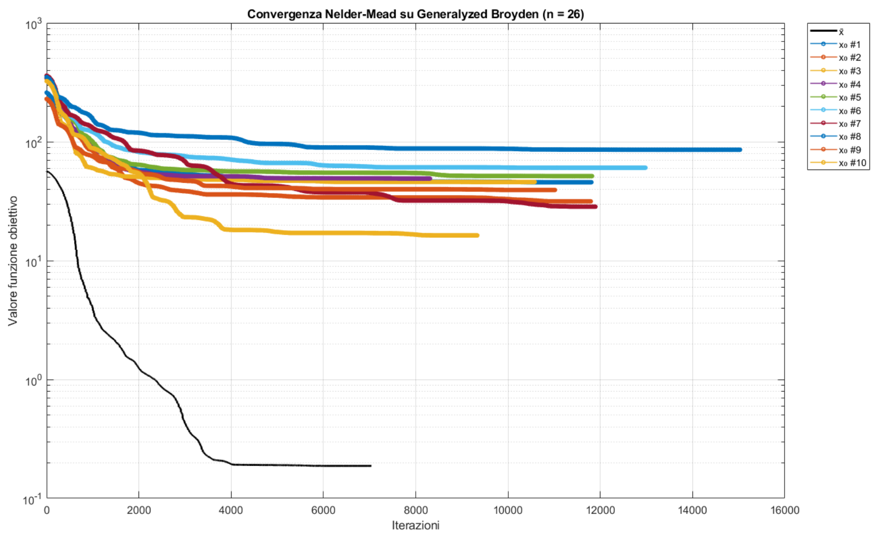

Figure 31: Convergence of Nelder-Mead method on Generalized Broyden function ($n = 26$) for the reference point $\bar{x}$ (black) and 10 randomly generated starting points.

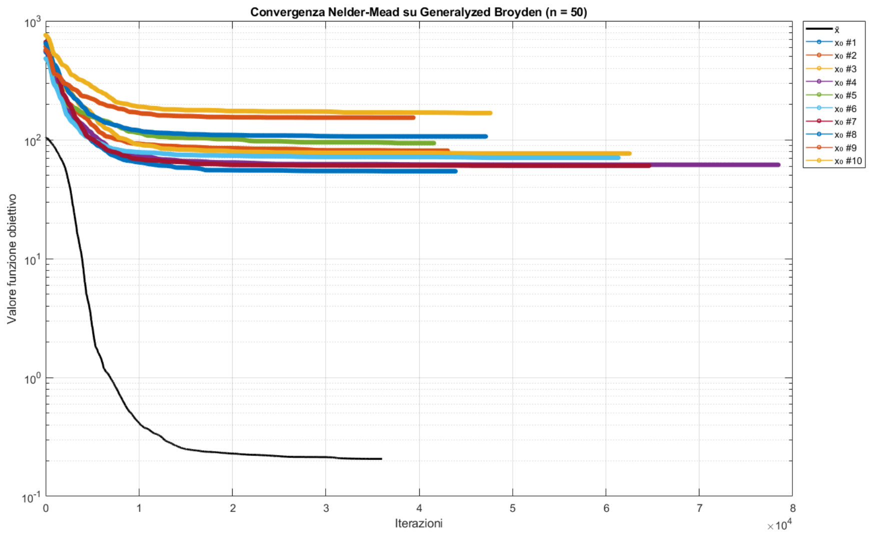

Figure 32: Convergence of Nelder-Mead method on Generalized Broyden function ($n = 50$) for the reference point $\bar{x}$ (black) and 10 randomly generated starting points.

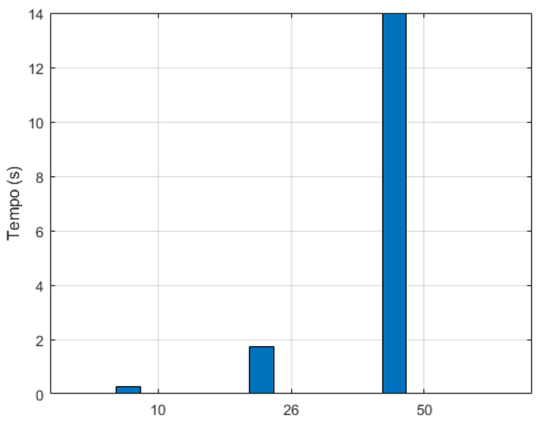

Figure 33: Computational time (in seconds) for Nelder-Mead method applied to the Generalized Broyden function for increasing dimensions.

5 Banded Trigonometric Function

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. As a result, the gradient and Hessian inherit a sparse structure, which is beneficial for large-scale optimization, especially when using Newton-based methods with sparse linear algebra.

The initial solution $\bar{x} \in \mathbb{R}^n$ used in our experiments is chosen as:

$$\bar{x} = [1, 1, \ldots, 1]^\top,$$

with added boundary values $x_0 = 0$ and $x_{n+1} = 0$.

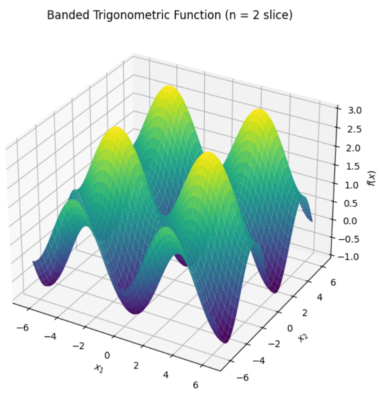

Figure 34: 3D visualization of the Banded Trigonometric function for $n = 2$.

5.1 Exact Gradient and Hessian

The Banded Trigonometric function is defined as:

$$f(x) = \sum_{i=1}^{n} i\,[(1 - \cos(x_{i+1})) + \sin(x_i) - \sin(x_{i+2})]$$

where $x \in \mathbb{R}^{n+2}$ with boundary conditions $x_0 = x_{n+1} = 0$. The structure of this function introduces trigonometric dependencies over three consecutive variables, resulting in a banded pattern in the derivatives.

The exact gradient is constructed analytically as:

$$\nabla_i f = \begin{cases} \sin(x_{i+1}) + 2\cos(x_{i+1}), & i = 1 \\ i\sin(x_{i+1}) + 2\cos(x_{i+1}), & 2 \le i \le n-1 \\ n\sin(x_n) - (n-1)\cos(x_n), & i = n \end{cases}$$

The Hessian is a diagonal matrix built from the second derivatives of the trigonometric terms. Its sparse implementation uses `spdiags` to store only the main diagonal, significantly reducing memory usage:

$$H_{ii} = \begin{cases} \cos(x_2) - 2\sin(x_2), & i = 1 \\ i\cos(x_{i+1}) - 2\sin(x_{i+1}), & 2 \le i \le n-1 \\ n\cos(x_n) + (n-1)\sin(x_n), & i = n \end{cases}$$

Due to the complexity and non-linearity of the trigonometric interactions, the resulting Hessian matrices are often ill-conditioned, especially in higher dimensions. To address this, the Modified Newton method applies **preconditioning** through the `pcg` (Preconditioned Conjugate Gradient) solver. 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)** 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.

All experiments were carried out for $n = 1000$, $10\,000$, and $100\,000$ with the initial vector:

$$\bar{x} = (0, 1, 1, \ldots, 1, 1, 0)^\top \in \mathbb{R}^{n+2}$$

which enforces the required boundary values and initializes the internal values to 1.

We report below the performance results obtained with exact derivatives.

Table 10: Results of Modified Newton method on Banded Trigonometric function using exact derivatives.

Dim	**Init.**	**Iter**	**Time (s)**	$f_{\min}$	ρ	**Successes**
10^3	$\bar{x}$	3	0.01	-426.02	4.32	0/1
	Avg (10 pts)	10.8	0.10	-300.79	1.53	0/10
10^4	$\bar{x}$	3	0.01	-4158.37	3.86	0/1
	Avg (10 pts)	15.2	1.33	-1381.99	1.60	0/10
10^5	$\bar{x}$	3	0.11	-41420.01	3.82	0/1
	Avg (10 pts)	21.4	2.07	-12080.91	1.53	0/10

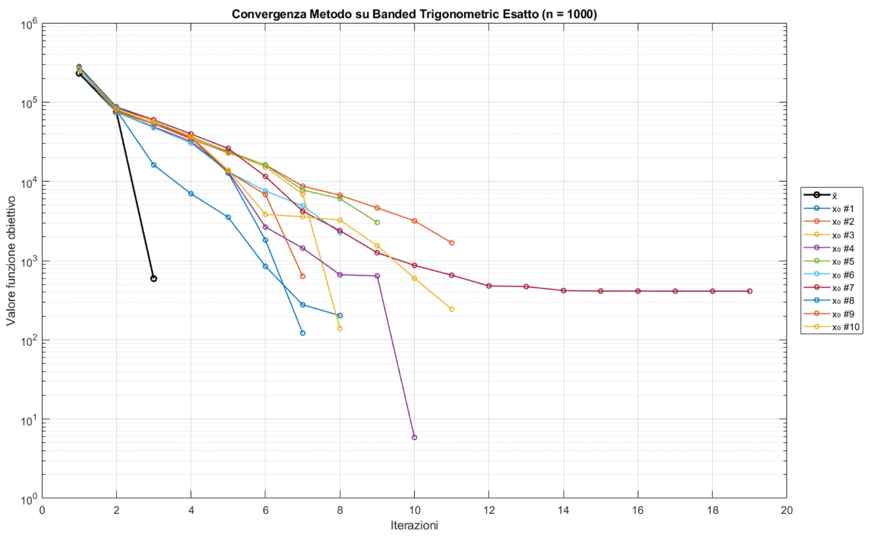

Figure 35: Convergence of the Modified Newton method on the Banded Trigonometric function ($n = 1000$) using exact derivatives.

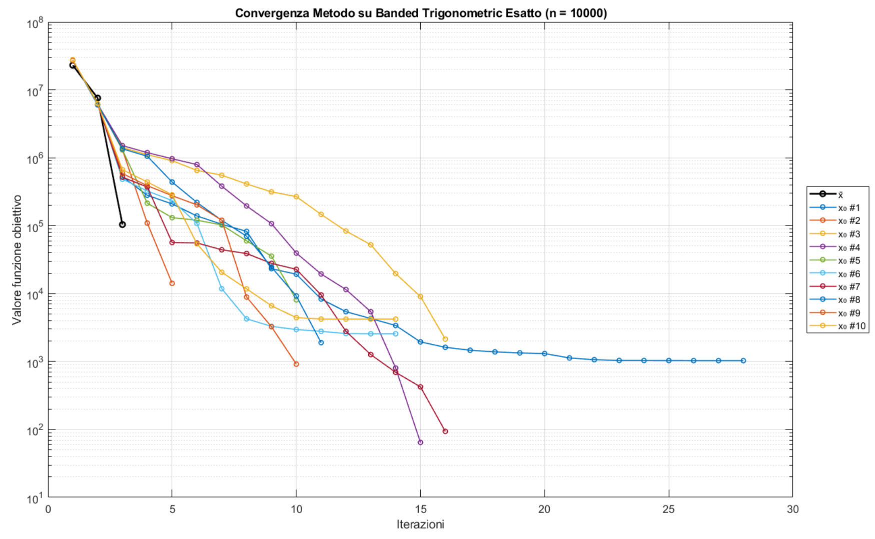

Figure 36: Convergence of the Modified Newton method on the Banded Trigonometric function ($n = 10000$) using exact derivatives.

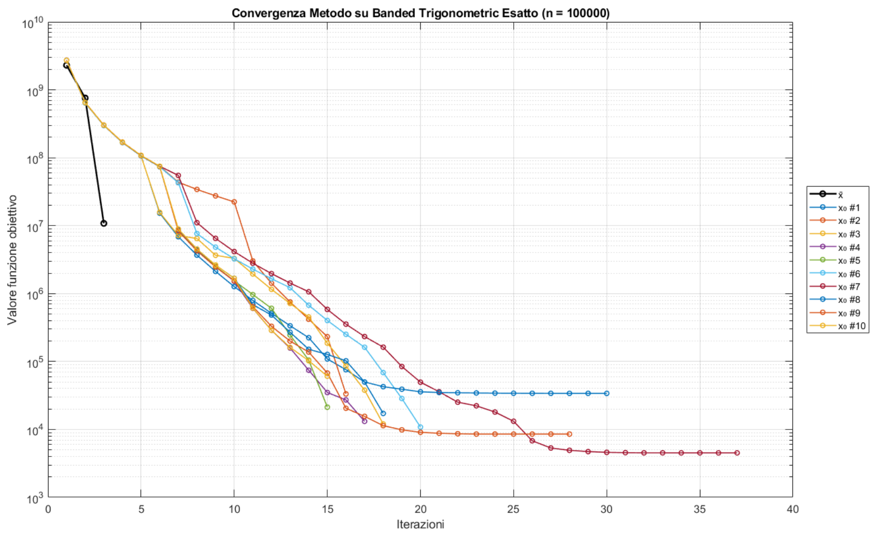

Figure 37: Convergence of the Modified Newton method on the Banded Trigonometric function ($n = 100000$) using exact derivatives.

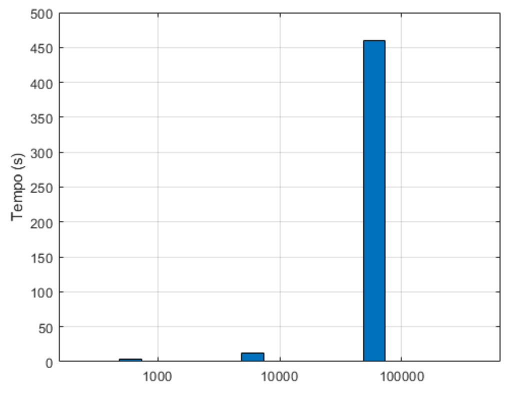

Figure 38: Computational time required by the Modified Newton method on the Banded Trigonometric function with exact derivatives for each problem size.

5.2 Finite Differences Gradient and Hessian

6 Conclusions