

# Numerical Optimization for Large Scale Problems

## Assignment Report

Giacomo Maino s338682, Mattia Molinari s337194, Alessandro Perlo s337131

February 3<sup>rd</sup>, 2025

### Abstract

In this report, we explore optimization techniques for large-scale unconstrained problems, focusing on variations of Newton method. Specifically, we implement and analyze the Modified Newton Method and the Truncated Newton Method, comparing their performance on several test functions. Both exact and finite difference-based Hessian and gradient computations are considered. Our experiments evaluate convergence rates, computational efficiency, and the impact of preconditioning. Additionally, we discuss the challenges posed by finite difference approximations and the sensitivity of each method to different problem structures.

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Modified Newton Method	2
1.2	Truncated Newton method	2
1.3	Finite differences	3
<b>2</b>	<b>Rosenbrock function</b>	<b>3</b>
2.1	Exact gradient and Hessian	3
2.2	Finite differences gradient and Hessian	3
<b>3</b>	<b>Extended Rosenbrock function</b>	<b>3</b>
3.1	Exact gradient and Hessian	3
3.2	Finite differences gradient and Hessian	4
<b>4</b>	<b>Generalized Broyden tridiagonal function</b>	<b>6</b>
4.1	Exact gradient and Hessian	9
4.2	Finite differences gradient and Hessian	9
<b>5</b>	<b>Banded trigonometric function</b>	<b>12</b>
5.1	Exact gradient and Hessian	13
5.2	Finite differences gradient and Hessian	15
<b>6</b>	<b>Conclusions</b>	<b>15</b>
<b>A</b>	<b>Code snippets</b>	<b>18</b>
A.1	Code for method implementations	18
A.1.1	Modified Newton method	18
A.1.2	Truncated Newton method	21
A.2	Code for objective functions	25
A.2.1	Extended Rosenbrock function	25
A.2.2	Generalized Broyden tridiagonal function	27
A.2.3	Banded trigonometric problem	28
A.3	Utility code for running experiments	29

# 1 Introduction

In this section we will describe the implementation details of the algorithms used to solve the optimization problems, namely modified Newton method and truncated Newton method, focusing on the differences with respect to the standard Newton method. These methods will be tested against the Rosenbrock function and three test problems from [1]. The chosen test problems are the extended Rosenbrock function (problem 25), the generalized Broyden tridiagonal function (problem 32) and the banded trigonometric function (problem 16) and results are contained in sections 3, 4 and 5 respectively.

The experiments were conducted using 11 points: a predefined starting point and 10 additional randomly generated points uniformly distributed in a hypercube around the initial guess. For each test function, we performed optimizations at problem dimensions  $n = 10^3, 10^4, 10^5$ . We implemented backtracking line search with a sufficient decrease condition, using standard parameters  $\rho = 0.5$  and  $c = 10^{-4}$ , further tuning has not been necessary in any case. Each method was evaluated in terms of success rate, number of iterations to convergence, execution time, and experimental convergence rate. The experimental convergence rate was computed using the formula:

$$q = \frac{\log(\|e_{k+1}\|/\|e_k\|)}{\log(\|e_k\|/\|e_{k-1}\|)} \quad (1)$$

where  $e_k$  denotes the error at iteration  $k$ . Error at iteration  $k$  is approximated as the norm of the difference between the point at current iteration and the point at previous iteration, i.e.  $\hat{e}_k = x_k - x_{k-1}$ . For each experiment, we report average metrics over the successful runs, where a run is considered successful if the method converges within a maximum of  $10^3$  iterations and Armijo condition is satisfied within  $bt_{max} = 50$  backtracking attempts.

## 1.1 Modified Newton Method

The modified Newton method aims to enhance robustness of the standard Newton method by ensuring positive-definiteness of the Hessian matrix. At iteration  $k$ , it is necessary to check whether the Hessian matrix  $H_k$  is positive definite: in case it is not, the matrix is modified by adding a matrix  $B_k$  in order to ensure positive definiteness. A common choice for  $B_k$  is a multiple of the identity matrix, i.e.  $B_k = \tau_k I$  so that the whole spectrum of  $H_k$  is shifted by  $\tau_k$ . Then we want to find the smallest  $\tau_k$  such that  $H_k + \tau_k I$  is positive definite, which is  $-\lambda_{k,min} + \beta$  where  $\lambda_{k,min}$  is the negative eigenvalue with the largest module.

To avoid to have to compute  $\lambda_{k,min}$ , we adopted the *Cholesky with Added Multiple of the Identity* algorithm outlined in [2] that consists in building a sequence of  $\tau_k$  until the modified matrix is positive definite. The sequence is built starting from  $\tau_k = \min_i h_{ii} + \beta$  where  $\min_i h_{ii}$  is the smallest diagonal element of  $H_k$ . Then, at each iteration:

1. positive-definiteness is assessed trying to perform a Cholesky factorization of  $H_k + \tau_k I$ ;
2. if the factorization is not successful,  $\tau_k$  is increased by a factor  $c$  and the process is repeated for a limited number of times  $k_{chol,max}$ .

In all the experiments we choose  $\beta = 10^{-3}$ ,  $k_{chol,max} = 100$ . A good value for the constant factor is  $c = 2$ , but as we will discuss in section 3 for the extended Rosenbrock function a larger value  $c = 5$  is beneficial. The method is endowed with a line search strategy with backtracking. We carry on experiments both with and without preconditioning, using the incomplete Cholesky factorization as preconditioner.

## 1.2 Truncated Newton method

The truncated Newton method aims to reduce the computational cost of the Newton method by adopting the following strategies:

- the newton system  $H_k p_k = -\nabla f(x_k)$  is solved approximately by means of an iterative method (i.e. conjugate gradient method), with a tolerance that depends on  $\|\nabla f(x_k)\|$ ;
- whenever a direction of negative curvature is found in the execution of the iterative method, the method is stopped and the direction is used as the search direction to prevent a non-negative curvature direction to be chosen in case of a non-positive definite  $H_k$ .

In all the experiments, we choose the relative tolerance for the iterative method at iteration  $k$  to be

$$\eta_k = \min\{0.5, \sqrt{\|\nabla f(x_k)\|}\}$$

that is a forcing term that is proven to yield a superlinear convergence rate. The method is endowed with a line search strategy with backtracking. We carry on experiments both with and without preconditioning, using the incomplete Cholesky factorization as preconditioner for the Newton system whenever the Hessian matrix is positive definite.

### 1.3 Finite differences

Experiments in subsequent sections will adopt both exact and finite differences gradient and Hessian to perform the optimization. When finite differences are adopted, the gradient will be estimated using centered finite differences

$$\frac{\partial f}{\partial x_k} \approx \frac{f(x + he_k) - f(x - he_k)}{2h} \quad (2)$$

while the Hessian will be estimated using forward finite differences, using the following formula

$$\frac{\partial^2 f}{\partial x_k \partial x_j} \approx \frac{f(x + he_k + he_j) - f(x + he_k) - f(x + he_j) + f(x)}{h^2} \quad (3)$$

where  $e_k$  and  $e_j$  are the  $k$ -th and the  $j$ -th canonical basis vectors respectively. Moreover, two different approaches will be adopted to choose the step size  $h$ : the first one will use a fixed step size while the second one will use a step size that depends on the current point  $x$  and that is different for each component, defined as follows

$$h_{k,i} = h|x_{k,i}|$$

where  $h_{k,i}$  is the increment for component  $i$  at step  $k$ ,  $h$  is a relative step size and  $x_{k,i}$  is the  $i$ -th component of the point at step  $k$ . Due to the large scale nature of the problems, the finite differences method is expected to be slower than the exact method, so ad-hoc implementations that will exploit the sparsity of the Hessian matrix and the separability of the specific functions will be used.

## 2 Rosenbrock function

### 2.1 Exact gradient and Hessian

AP: [TODO]

### 2.2 Finite differences gradient and Hessian

AP: [TODO]

## 3 Extended Rosenbrock function

The extended Rosenbrock function is a generalization of the Rosenbrock function to  $n$  dimensions, defined as follows. Figure 1 shows the surface plot of the 2-dimensional extended Rosenbrock function: notice that for  $n = 2$  it is identical to the standard Rosenbrock function, except for the  $\frac{1}{2}$  term.

$$F(x) = \frac{1}{2} \sum_{k=1}^n f_k^2(x), \quad f_k(x) = \begin{cases} 10(x_k^2 - x_{k+1}), & k \bmod 2 = 1 \\ x_{k-1} - 1, & k \bmod 2 = 0 \end{cases} \quad (4)$$

The minimum of the function is in a very flat valley which is easy to reach, but in practice it's harder to converge to a minimum, which makes the extended Rosenbrock function a challenging optimization problem. AP: [Add something to justify the choice of the constant for modified newton method].

### 3.1 Exact gradient and Hessian

The gradient of the extended Rosenbrock function is given by the following expression,

$$\frac{\partial F}{\partial x_k} = \begin{cases} 200(x_k^3 - x_k x_{k+1}) + (x_k - 1), & k \bmod 2 = 1 \\ -100(x_{k-1}^2 - x_k), & k \bmod 2 = 0 \end{cases} \quad (5)$$

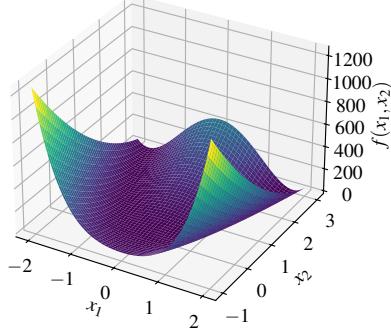


Figure 1: Surface plot of the 2-dimensional extended Rosenbrock function

computation can be eased considering that component  $k$  depends only on  $f_k$  and  $f_{k+1}$  when  $k$  is odd, and only on  $f_{k-1}$  when  $k$  is even. The Hessian of the extended Rosenbrock function is given by the following expression.

$$\frac{\partial^2 F}{\partial x_k \partial x_j} = \begin{cases} 200(3x_k^2 - x_{k+1}) + 1, & j = k, k \bmod 2 = 1 \\ 100, & j = k, k \bmod 2 = 0 \\ -200x_k, & |k - j| = 1, k \bmod 2 = 1 \\ 0, & \text{otherwise} \end{cases} \quad (6)$$

Notice that the Hessian is a sparse matrix, with only  $n$  non-zero elements on the diagonal and  $n/2$  non-zero elements on the first co-diagonal.

Table 1 shows the results for the *Modified Newton method* applied to the extended Rosenbrock function with exact gradient and Hessian. All attempts were successful, and the method converged in a small number of iterations, with a convergence rate that is close to 2 for all dimensions but  $10^5$ , where the convergence rate is smaller and close to 1. However, the time required to converge is significantly higher for the  $10^5$ -dimensional problem, which is expected due to the increased number of function evaluations required to compute the gradient and Hessian. We observe that being the number of iterations necessary to converge to the minimum of the function very low, the experimental convergence rate is not very reliable, as it is computed as the ratio between the number of iterations and the logarithm of the relative error.

Table 2 shows the results for the *Truncated Newton method* applied to the extended Rosenbrock function with exact gradient and Hessian. All attempts were successful, and the method converged in a small number of iterations. The computed experimental convergence rate is not reliable at all, presumably due to the small number of iterations and truncations of the iterative solver used to solve the Newton system. Moreover, when preconditioning is not adopted we get a negative convergence rate since  $\log \|e_k\|$  where  $e_k$  is the error at iteration  $k$  is not monotonic with respect to  $k$ , as shown in figure 2. However, when preconditioning is adopted and  $k$  is larger, i.e. for  $n = 10^5$ , the convergence rate is superlinear as expected from the truncated Newton method with a superlinear forcing term.

Figure 2 shows the estimate of the error for the Modified Newton method and for the Truncated Newton method applied to the Extended Rosenbrock function with exact gradient and Hessian for  $n = 10^5$  when starting from random point 1:

- for the *Modified Newton method*, in early iterates the estimated error is not monotonic, but it becomes monotonic after a few iterations regardless of preconditioning;
- for the *Truncated Newton method*, in early iterates the estimated error is not monotonic, but it becomes monotonic after a few iterations, but only if preconditioning is adopted.

Both in the case of the Modified Newton method and the Truncated Newton method, preconditioning improves performance of the optimization algorithms both in terms of number of iterations and time required to converge.

### 3.2 Finite differences gradient and Hessian

When applying 2, one can notice that the terms  $F(x + he_k)$  and  $F(x - he_k)$  only differ by terms  $f_k$  and  $f_{k+1}$  for  $k$  odd, by terms  $f_{k-1}$  for  $k$  even. Then to make function evaluations less expensive, we can define the following

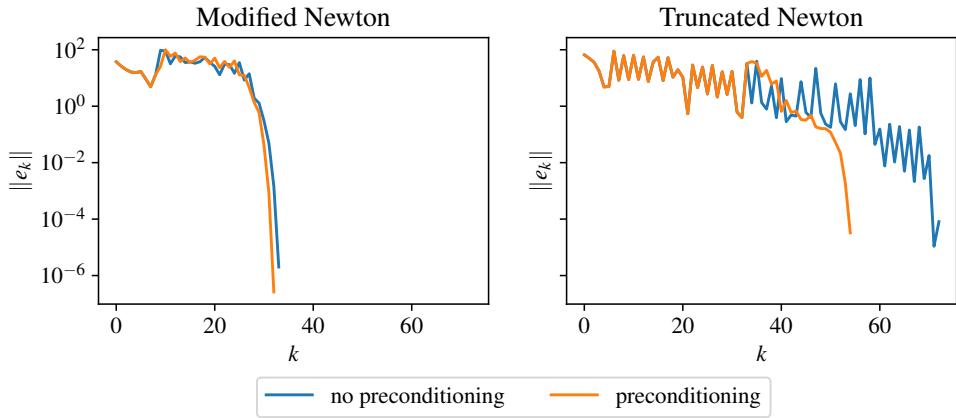


Figure 2: Estimate of the error for Modified Newton method and for the Truncated Newton method applied to the Extended Rosenbrock function with exact gradient and Hessian for  $n = 10^5$ , random point 1.

Table 1: Results for Modified Newton method applied to Extended Rosenbrock with exact gradient and hessian, metrics are average metrics for successful attempts.

preconditioning dimension	iterations		convergence rate		time		success rate	
	False	True	False	True	False	True	False	True
3	31.91	28.91	1.99	1.51	0.05	0.02	1.00	1.00
4	32.36	29.18	2.10	2.04	0.11	0.08	1.00	1.00
5	26.50	26.00	1.10	1.00	0.68	0.53	1.00	1.00

Table 2: Results for Truncated Newton method applied to Extended Rosenbrock with exact gradient and hessian, metrics are average metrics for successful attempts.

preconditioning dimension	iterations		convergence rate		time		success rate	
	False	True	False	True	False	True	False	True
3	50.09	31.00	-2.66	11.55	0.01	0.01	1.00	1.00
4	57.27	34.73	-3.06	4.33	0.04	0.03	1.00	1.00
5	64.00	28.50	-5.58	1.62	0.45	0.23	1.00	1.00

function  $F_{fd,k}$ , which can be plugged in 2 in place of  $F$  yielding the same result.

$$F_{fd,k}(x) = \begin{cases} \frac{1}{2}f_k^2(x) + \frac{1}{2}f_{k+1}^2(x), & k \bmod 2 = 1 \\ \frac{1}{2}f_{k-1}^2(x), & k \bmod 2 = 0 \end{cases}$$

The same procedure can be applied for the Hessian, considering that:

- function evaluations to compute entry  $h_{k,k}$  differ only by  $f_k$  and  $f_{k+1}$  for  $k$  odd, and only on  $f_{k-1}$  for  $k$  even;
- function evaluations to compute entry  $h_{k,k+1}$  differ only by  $f_k$  and  $f_{k+1}$  for  $k$  odd.

Then to make function evaluations less expensive, we can define the functions  $F_{fd,k,k}$  and  $F_{fd,k,k+1}$ , which can be plugged in 3 in place of  $F$  yielding the same result to compute entries  $h_{k,k}$  and  $h_{k,k+1}$  respectively.

$$\begin{aligned} F_{fd,k,k}(x) &= \begin{cases} \frac{1}{2}f_k^2(x) + \frac{1}{2}f_{k+1}^2(x), & k \bmod 2 = 1 \\ \frac{1}{2}f_{k-1}^2(x), & k \bmod 2 = 0 \end{cases} \\ F_{fd,k,k+1}(x) &= \begin{cases} \frac{1}{2}f_k^2(x) + \frac{1}{2}f_{k+1}^2(x), & k \bmod 2 = 1 \\ 0, & k \bmod 2 = 0 \end{cases} \end{aligned}$$

When plugging the functions  $F_{fd,k}$ ,  $F_{fd,k,k}$  and  $F_{fd,k,k+1}$  into 2 and 3 it's convenient to expand them so that the computation of the gradient and Hessian is not subject to numerical cancellation. After expanding the functions, the gradient and Hessian can be approximated as follows.

$$\begin{aligned} \frac{\partial F}{\partial x_k} &\approx \begin{cases} 600h^2x_k - 100hx_{k+1} + \frac{1}{2}h + 350h^3 + 300hx_k^2, & k \bmod 2 = 1 \\ -100x_{k-1}^2 + 100x_k, & k \bmod 2 = 0 \end{cases} \\ \frac{\partial^2 F}{\partial x_k \partial x_j} &\approx \begin{cases} 1200h_kx_k - 200x_{k+1} + 1 + 700h_k^2 + 600x_k^2, & j = k, k \bmod 2 = 1 \\ 100, & j = k, k \bmod 2 = 0 \\ -100h_kh_{k+1} - 200x_k, & |k - j| = 1, k \bmod 2 = 1 \\ 0, & \text{otherwise} \end{cases} \end{aligned}$$

Tables 3 and 4 show the results for the *Modified Newton method* applied to the extended Rosenbrock function with absolute and specific finite differences respectively. All attempts were successful, and the method converged in a small number of iterations, with a convergence rate that is close to 2 for all dimensions, when a suitable choice of  $h$  is made. On the contrary, when a poor choice of  $h$  is made, the convergence rate is close to 1, which is expected since the convergence rate of the Newton method is 2: this happens when  $h = 10^{-2}$  or  $h = 10^{-4}$ , both for the constant increment and the specific increment methodologies. We notice that for a fixed dimension, performance improves as the stepsize  $h$  (be it constant or specific) decreases, as expected, up to a certain point where reducing the stepsize does not result in any improvement (i.e. time and iterations do not significantly decrease). This plateau is reached sooner when preconditioning is adopted and when the dimension is lower.

Tables 5 and 6 show the results for the *Truncated Newton method* applied to the extended Rosenbrock function with absolute and specific finite differences respectively. All attempts were successful, and the method converged in a small number of iterations, when a suitable choice of  $h$  is made. A larger number of iterations is required when a poor choice of  $h$  is made, namely when  $h = 10^{-2}$  or  $h = 10^{-4}$ , both for the constant increment and the specific increment methodologies, yielding a convergence rate of 1. For smaller values of  $h$ , the estimate of the convergence rate is not reliable at all and considerations made in subsection 3.1 for the Truncated Newton method applied to the extended Rosenbrock function with exact gradient and Hessian apply here as well. The plateau in performance as the stepsize  $h$  decreases is reached slower than in the case of the Modified Newton method, hinting that the Truncated Newton method highly benefits from a finer approximation of the derivatives.

## 4 Generalized Broyden tridiagonal function

The generalized Broyden tridiagonal function is defined as follows.

$$F(x) = \frac{1}{2} \sum_{i=1}^n f_i^2(x) \quad f_k(x) = (3 - 2x_k)x_k + 1 - x_{k-1} - x_{k+1} \quad (7)$$

Figure 3 shows the surface plot of the 2-dimensional generalized Broyden tridiagonal function. Notice that the area where the minimum lies is very flat, which makes it hard to converge to the minimum.

Table 3: Results for Modified Newton method applied to Extended Rosenbrock with absolute finite differences, metrics are average metrics for successful attempts.

dimension	preconditioning h	iterations		convergence rate		time		success rate	
		False	True	False	True	False	True	False	True
3	1e-02	149.91	150.82	1.00	1.00	0.07	0.07	1.00	1.00
	1e-04	33.18	32.09	1.01	1.00	0.02	0.02	1.00	1.00
	1e-06	32.09	30.18	1.96	2.09	0.03	0.02	1.00	1.00
	1e-08	32.00	30.18	2.04	2.23	0.02	0.02	1.00	1.00
	1e-10	32.00	30.18	2.00	2.23	0.02	0.02	1.00	1.00
	1e-12	32.00	30.18	2.00	2.23	0.02	0.02	1.00	1.00
4	1e-02	160.09	160.55	1.00	1.00	0.34	0.36	1.00	1.00
	1e-04	34.18	32.55	1.00	1.00	0.12	0.09	1.00	1.00
	1e-06	32.55	30.27	1.89	1.96	0.12	0.08	1.00	1.00
	1e-08	32.55	30.27	1.95	1.98	0.12	0.08	1.00	1.00
	1e-10	32.55	30.27	1.95	1.98	0.11	0.08	1.00	1.00
	1e-12	32.55	30.27	1.95	1.98	0.11	0.08	1.00	1.00
5	1e-02	169.36	169.82	1.00	1.00	3.26	3.36	1.00	1.00
	1e-04	34.82	33.73	1.00	1.00	1.08	0.79	1.00	1.00
	1e-06	34.00	31.73	2.13	2.00	1.17	0.75	1.00	1.00
	1e-08	33.27	31.73	2.60	2.10	1.17	0.73	1.00	1.00
	1e-10	33.45	31.73	1.93	2.10	1.13	0.73	1.00	1.00
	1e-12	33.45	31.73	1.90	2.10	1.15	0.73	1.00	1.00

Table 4: Results for Modified Newton method applied to Extended Rosenbrock with specific finite differences, metrics are average metrics for successful attempts.

dimension	preconditioning h	iterations		convergence rate		time		success rate	
		False	True	False	True	False	True	False	True
3	1e-02	139.82	141.45	1.00	1.00	0.06	0.06	1.00	1.00
	1e-04	33.73	32.27	1.00	1.00	0.02	0.02	1.00	1.00
	1e-06	32.09	30.18	1.89	2.06	0.02	0.02	1.00	1.00
	1e-08	32.00	30.18	2.04	2.23	0.02	0.02	1.00	1.00
	1e-10	32.00	30.18	2.00	2.23	0.02	0.02	1.00	1.00
	1e-12	32.00	30.18	2.00	2.23	0.02	0.02	1.00	1.00
4	1e-02	149.55	150.55	1.00	1.00	0.32	0.32	1.00	1.00
	1e-04	34.45	32.45	1.00	1.00	0.12	0.09	1.00	1.00
	1e-06	32.55	30.27	1.81	1.96	0.12	0.09	1.00	1.00
	1e-08	32.55	30.27	1.95	1.98	0.12	0.08	1.00	1.00
	1e-10	32.55	30.27	1.95	1.98	0.12	0.08	1.00	1.00
	1e-12	32.55	30.27	1.95	1.98	0.11	0.08	1.00	1.00
5	1e-02	159.18	160.27	1.00	1.00	3.05	3.17	1.00	1.00
	1e-04	34.64	34.45	1.00	1.00	1.09	0.80	1.00	1.00
	1e-06	33.45	31.73	2.56	2.00	1.16	0.74	1.00	1.00
	1e-08	33.09	31.73	1.84	2.10	1.13	0.73	1.00	1.00
	1e-10	33.45	31.73	1.90	2.10	1.12	0.74	1.00	1.00
	1e-12	33.45	31.73	1.90	2.10	1.14	0.77	1.00	1.00

Table 5: Results for Truncated Newton method applied to Extended Rosenbrock with absolute finite differences, metrics are average metrics for successful attempts.

dimension	preconditioning h	iterations		convergence rate		time		success rate	
		False	True	False	True	False	True	False	True
3	1e-02	182.00	179.27	1.00	1.00	0.02	0.04	1.00	1.00
	1e-04	54.36	37.91	1.00	1.00	0.01	0.01	1.00	1.00
	1e-06	52.82	36.36	-1.48	2.39	0.01	0.01	1.00	1.00
	1e-08	52.91	35.91	-2.12	2.86	0.01	0.01	1.00	1.00
	1e-10	53.36	35.45	-4.10	2.21	0.01	0.01	1.00	1.00
	1e-12	55.64	36.45	-2.49	3.01	0.01	0.01	1.00	1.00
4	1e-02	203.91	192.18	1.00	1.00	0.16	0.17	1.00	1.00
	1e-04	61.64	44.45	1.00	1.00	0.06	0.05	1.00	1.00
	1e-06	59.18	42.55	-1.35	4.36	0.06	0.05	1.00	1.00
	1e-08	61.64	42.09	-1.78	2.34	0.06	0.05	1.00	1.00
	1e-10	62.27	41.91	-2.46	3.14	0.06	0.05	1.00	1.00
	1e-12	61.73	43.00	-2.89	2.65	0.06	0.05	1.00	1.00
5	1e-02	224.55	206.82	1.00	1.00	1.84	1.83	1.00	1.00
	1e-04	74.64	42.91	1.00	1.01	0.73	0.41	1.00	1.00
	1e-06	79.09	49.27	-2.17	2.06	0.81	0.50	1.00	1.00
	1e-08	75.64	49.91	-2.83	7.31	0.75	0.47	1.00	1.00
	1e-10	78.00	51.64	-1.21	3.21	0.75	0.50	1.00	1.00
	1e-12	78.00	53.27	-2.11	2.50	0.77	0.52	1.00	1.00

Table 6: Results for Truncated Newton method applied to Extended Rosenbrock with specific finite differences, metrics are average metrics for successful attempts.

dimension	preconditioning h	iterations		convergence rate		time		success rate	
		False	True	False	True	False	True	False	True
3	1e-02	169.91	169.91	1.00	1.00	0.02	0.03	1.00	1.00
	1e-04	54.45	37.27	1.00	1.02	0.01	0.01	1.00	1.00
	1e-06	51.45	36.27	-2.69	2.35	0.01	0.01	1.00	1.00
	1e-08	53.64	36.27	-3.43	1.91	0.01	0.01	1.00	1.00
	1e-10	54.64	35.27	-2.94	2.01	0.01	0.01	1.00	1.00
	1e-12	53.82	35.91	-4.18	2.56	0.01	0.01	1.00	1.00
4	1e-02	191.09	182.64	1.00	1.00	0.15	0.17	1.00	1.00
	1e-04	64.27	42.55	1.00	1.01	0.06	0.05	1.00	1.00
	1e-06	62.18	43.00	-0.96	2.62	0.06	0.05	1.00	1.00
	1e-08	60.73	40.91	-2.95	2.87	0.06	0.05	1.00	1.00
	1e-10	60.82	42.73	-1.22	3.98	0.06	0.05	1.00	1.00
	1e-12	63.09	42.91	-2.01	2.51	0.06	0.05	1.00	1.00
5	1e-02	213.55	198.64	1.00	1.00	1.68	1.74	1.00	1.00
	1e-04	75.91	40.82	1.00	1.00	0.74	0.42	1.00	1.00
	1e-06	77.91	49.00	-2.96	6.91	0.77	0.49	1.00	1.00
	1e-08	77.64	50.18	-1.96	2.28	0.77	0.48	1.00	1.00
	1e-10	75.00	53.27	-4.07	3.38	0.75	0.52	1.00	1.00
	1e-12	75.09	52.45	-2.09	3.48	0.75	0.52	1.00	1.00

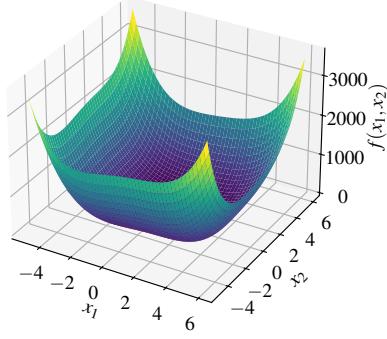


Figure 3: Surface plot of the 2-dimensional generalized Broyden tridiagonal function

#### 4.1 Exact gradient and Hessian

The gradient of the generalized Broyden tridiagonal function is given by the following expression,

$$\frac{\partial F}{\partial x_k} = \begin{cases} (3 - 4x_1)f_1(x) - f_2(x), & k = 1 \\ (3 - 4x_k)f_k(x) - f_{k+1}(x) - f_{k-1}(x), & 1 < k < n \\ (3 - 4x_n)f_n(x) - f_{n-1}(x), & k = n \end{cases} \quad (8)$$

computation can be eased considering that component  $k$  depends only on  $f_k$ ,  $f_{k+1}$  and  $f_{k-1}$ . The Hessian of the generalized Broyden tridiagonal function is given by the following expression.

$$\frac{\partial^2 F}{\partial x_k \partial x_j} = \begin{cases} (3 - 4x_1)^2 - 4f_1(x) + 1, & k = j = 1 \\ (3 - 4x_k)^2 - 4f_k(x) + 2, & 1 < k = j < n \\ (3 - 4x_n)^2 - 4f_n(x) + 1, & k = j = n \\ 4x_k + 4x_{k+1} - 6, & |k - j| = 1 \\ 1, & |k - j| = 2 \\ 0, & \text{otherwise} \end{cases} \quad (9)$$

Notice that the Hessian is a banded matrix, with only  $n$  non-zero elements on the diagonal,  $n - 1$  non-zero elements on the first co-diagonal and  $n - 2$  non-zero elements on the second co-diagonal.

Table 7 shows the results for the *Modified Newton method* applied to the generalized Broyden tridiagonal function with exact gradient and Hessian. All attempts are successful, and the method converges in a small number of iterations. Convergence rate is close to 2, which is the expected convergence rate for Newton's method. Preconditioning does not seem to have a significant impact on the performance of the method, compared to its impact on the performance on the extended Rosenbrock function due to the fact that the Hessian matrix for the Newton system is better scaled in the currently considered function.

Table 8 shows the results for the *Truncated Newton method* applied to the generalized Broyden tridiagonal function with exact gradient and Hessian. All attempts are successful, and the method converges in a small number of iterations. When preconditioning is not adopted convergence rate is lower than 2, which is expected since we are solving the Newton system with a relative tolerance that is guaranteed to yield superlinear convergence. Differently from the Modified Newton method, the Truncated Newton method is more sensitive to preconditioning, which can be observed by the fact that the number of iterations is lower when preconditioning is adopted and convergence rate is almost 2.

Comparing performance of the two methods, the Truncated Newton method is 2 to 3 times faster than the Modified Newton method due to the fact that the Newton system is solved approximately, which is less computationally expensive than solving it exactly. However, the Truncated Newton methods requires more iterations to converge than the Modified Newton method, which is expected since the Modified Newton exactly solves the Newton system at each iteration.

#### 4.2 Finite differences gradient and Hessian

When applying 2, one can notice that the terms  $F(x + he_k)$  and  $F(x - he_k)$  only differ by terms  $f_k$ ,  $f_{k+1}$  and  $f_{k-1}$ . Then to make function evaluations less expensive, we can define the following function  $F_{fd,k}$ , which can

Table 7: Results for Modified Newton method applied to Generalized Broyden with exact gradient and hessian, metrics are average metrics for successful attempts.

preconditioning dimension	iterations		convergence rate		time		success rate	
	False	True	False	True	False	True	False	True
3	8.636	8.636	1.974	1.978	0.006	0.005	1.00	1.00
4	8.000	8.273	1.829	1.849	0.037	0.034	1.00	1.00
5	7.909	10.545	1.794	1.989	0.291	0.300	1.00	1.00

Table 8: Results for Truncated Newton method applied to Generalized Broyden with exact gradient and hessian, metrics are average metrics for successful attempts.

preconditioning dimension	iterations		convergence rate		time		success rate	
	False	True	False	True	False	True	False	True
3	11.818	9.000	1.774	1.971	0.002	0.003	1.00	1.00
4	12.727	9.727	1.346	1.987	0.014	0.015	1.00	1.00
5	13.727	9.636	1.905	1.923	0.125	0.108	1.00	1.00

be plugged in 2 in place of  $F$  yielding the same result.

$$F_{fd,k}(x) = \frac{1}{2}f_k^2(x) + \frac{1}{2}f_{k+1}^2(x) + \frac{1}{2}f_{k-1}^2(x)$$

The same procedure can be applied for the Hessian, considering that:

- function evaluations to compute entry  $h_{k,k}$  differ only by  $f_k$ ,  $f_{k+1}$  and  $f_{k-1}$ ;
- function evaluations to compute entry  $h_{k,k+1}$  differ only by  $f_k$  and  $f_{k+1}$ ;
- function evaluations to compute entry  $h_{k,k+2}$  differ only by  $f_{k-1}$ .

Then to make function evaluations less expensive, we can define the functions  $F_{fd,k,k}$ ,  $F_{fd,k,k+1}$ ,  $F_{fd,k,k+2}$ , which can be plugged in 3 in place of  $F$  yielding the same result to compute entries  $h_{k,k}$ ,  $h_{k,k+1}$  and  $h_{k,k+2}$  respectively.

$$\begin{aligned} F_{fd,k,k}(x) &= \frac{1}{2}f_k^2(x) + \frac{1}{2}f_{k-1}^2(x) + \frac{1}{2}f_{k+1}^2(x) \\ F_{fd,k,k+1}(x) &= \frac{1}{2}f_k^2(x) + \frac{1}{2}f_{k+1}^2(x) \\ F_{fd,k,k+2}(x) &= \frac{1}{2}f_{k-1}^2(x) \end{aligned}$$

When plugging the functions  $F_{fd,k}$ ,  $F_{fd,k,k}$ ,  $F_{fd,k,k+1}$  and  $F_{fd,k,k+2}$  into 2 and 3 it's convenient to expand them so that the computation of the gradient and Hessian is not subject to numerical cancellation as previously done for the extended Rosenbrock function in subsection 3.2.

Tables 9 and 10 show the results for the *Modified Newton method* applied to the generalized Broyden tridiagonal function with absolute and specific finite differences respectively. All attempts are successful, but attempts with  $h = 10^{-2}$  which don't converge within the fixed maximum number of iterations  $k_{max} = 1000$ , both for absolute and specific differences, for all dimensions. This is probably due to the fact that  $h = 10^{-2}$  is not a suitable increment for the finite differences method to approximate the gradient and Hessian of the generalized Broyden tridiagonal function. When the method converges, it does so in a small number of iterations with a convergence rate close to 2, which is the expected convergence rate for Newton's method.

Tables 11 and 12 show the results for the *Truncated Newton method* applied to the generalized Broyden tridiagonal function with absolute and specific finite differences respectively. All attempts are successful, but attempts with  $h = 10^{-2}$  which converge within the fixed maximum number of iterations  $k_{max} = 1000$  only once for  $n = 10^3$ , regardless of the type (absolute or specific) of finite differences adopted. When the method converges, it does so in a small number of iterations with a convergence rate that as expected is superlinear, almost quadratic for  $n = 10^5$ , even if preconditioning is not adopted. When preconditioning is adopted, the number of iterations is lower and the convergence rate is closer to 2, which is the expected convergence rate for Newton's method.

Table 9: Results for Modified Newton method applied to Generalized Broyden with absolute finite differences, metrics are average metrics for successful attempts.

dimension	preconditioning h	iterations		convergence rate		time		success rate	
		False	True	False	True	False	True	False	True
3	1e-04	8.545	8.545	2.016	2.010	0.006	0.006	1.00	1.00
	1e-06	8.545	8.636	1.964	1.979	0.006	0.006	1.00	1.00
	1e-08	8.636	8.636	1.974	1.978	0.006	0.005	1.00	1.00
	1e-10	8.636	8.636	1.974	1.978	0.006	0.006	1.00	1.00
	1e-12	8.636	8.636	1.973	1.978	0.006	0.006	1.00	1.00
4	1e-04	7.909	8.273	1.803	1.829	0.036	0.031	1.00	1.00
	1e-06	8.000	8.273	1.832	1.851	0.036	0.031	1.00	1.00
	1e-08	8.000	8.273	1.829	1.849	0.038	0.031	1.00	1.00
	1e-10	8.000	8.273	1.829	1.849	0.038	0.031	1.00	1.00
	1e-12	8.000	8.273	1.829	1.849	0.038	0.032	1.00	1.00
5	1e-04	7.818	10.455	1.751	2.045	0.321	0.344	1.00	1.00
	1e-06	7.909	10.545	1.796	1.992	0.333	0.332	1.00	1.00
	1e-08	7.909	10.545	1.794	1.989	0.324	0.337	1.00	1.00
	1e-10	7.909	10.545	1.794	1.989	0.314	0.325	1.00	1.00
	1e-12	7.909	10.545	1.794	1.989	0.320	0.328	1.00	1.00

Table 10: Results for Modified Newton method applied to Generalized Broyden with specific finite differences, metrics are average metrics for successful attempts.

dimension	preconditioning h	iterations		convergence rate		time		success rate	
		False	True	False	True	False	True	False	True
3	1e-04	8.545	8.636	1.977	2.000	0.006	0.006	1.00	1.00
	1e-06	8.545	8.636	1.964	1.979	0.006	0.005	1.00	1.00
	1e-08	8.636	8.636	1.974	1.978	0.007	0.007	1.00	1.00
	1e-10	8.636	8.636	1.973	1.978	0.006	0.006	1.00	1.00
	1e-12	8.636	8.636	1.973	1.978	0.007	0.006	1.00	1.00
4	1e-04	7.909	8.273	1.804	1.835	0.037	0.031	1.00	1.00
	1e-06	8.000	8.273	1.830	1.850	0.050	0.031	1.00	1.00
	1e-08	8.000	8.273	1.829	1.849	0.038	0.032	1.00	1.00
	1e-10	8.000	8.273	1.829	1.849	0.038	0.032	1.00	1.00
	1e-12	8.000	8.273	1.829	1.849	0.038	0.031	1.00	1.00
5	1e-04	7.818	10.545	1.752	2.020	0.315	0.325	1.00	1.00
	1e-06	7.909	10.545	1.795	1.990	0.314	0.326	1.00	1.00
	1e-08	7.909	10.545	1.794	1.989	0.329	0.329	1.00	1.00
	1e-10	7.909	10.545	1.794	1.989	0.325	0.327	1.00	1.00
	1e-12	7.909	10.545	1.794	1.989	0.313	0.325	1.00	1.00

Table 11: Results for Truncated Newton method applied to Generalized Broyden with absolute finite differences, metrics are average metrics for successful attempts.

dimension	preconditioning h	iterations		convergence rate		time		success rate	
		False	True	False	True	False	True	False	True
3	1e-02	13.000	NaN	1.051	NaN	0.002	NaN	0.09	NaN
	1e-04	11.818	9.091	1.773	2.009	0.003	0.003	1.00	1.00
	1e-06	11.818	9.000	1.774	1.972	0.003	0.003	1.00	1.00
	1e-08	11.818	9.000	1.774	1.971	0.003	0.003	1.00	1.00
	1e-10	11.818	9.000	1.774	1.971	0.003	0.003	1.00	1.00
	1e-12	11.818	9.000	1.774	1.971	0.003	0.004	1.00	1.00
4	1e-04	12.727	9.545	1.347	1.995	0.019	0.018	1.00	1.00
	1e-06	12.727	9.727	1.346	1.990	0.018	0.017	1.00	1.00
	1e-08	12.727	9.727	1.346	1.987	0.018	0.017	1.00	1.00
	1e-10	12.727	9.727	1.346	1.987	0.018	0.018	1.00	1.00
	1e-12	12.727	9.727	1.346	1.987	0.019	0.018	1.00	1.00
5	1e-04	13.727	9.545	1.905	1.903	0.160	0.132	1.00	1.00
	1e-06	13.727	9.636	1.905	1.926	0.162	0.133	1.00	1.00
	1e-08	13.727	9.636	1.905	1.923	0.159	0.137	1.00	1.00
	1e-10	13.727	9.636	1.905	1.923	0.161	0.132	1.00	1.00
	1e-12	13.727	9.636	1.905	1.923	0.156	0.132	1.00	1.00

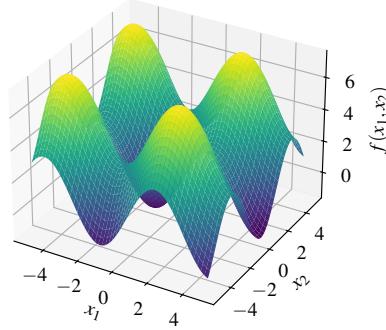


Figure 4: Surface plot of the 2-dimensional banded trigonometric function

Also for this function, it's evident that the Truncated Newton method is faster than the Modified Newton method, but requires more iterations to converge. Moreover, the Truncated Newton method is more sensitive to preconditioning than the Modified Newton method.

## 5 Banded trigonometric function

The banded trigonometric function is defined as follows.

$$f(x) = \sum_{i=1}^n i[(1 - \cos x_i) + \sin x_{i-1} - \sin x_{i+1}] \quad (10)$$

Figure 4 shows the surface plot of the 2-dimensional banded trigonometric function. It has a highly oscillatory landscape with multiple peaks and valleys due to its sinusoidal terms. This results in a mix of locally smooth and rapidly changing regions, making optimization sensitive to initialization and prone to multiple local minima.

Table 12: Results for Truncated Newton method applied to Generalized Broyden with specific finite differences, metrics are average metrics for successful attempts.

dimension	preconditioning h	iterations		convergence rate		time		success rate	
		False	True	False	True	False	True	False	True
3	1e-04	11.818	9.091	1.773	1.986	0.003	0.003	1.00	1.00
	1e-06	11.818	9.000	1.774	1.971	0.003	0.003	1.00	1.00
	1e-08	11.818	9.000	1.774	1.971	0.003	0.003	1.00	1.00
	1e-10	11.818	9.000	1.774	1.971	0.003	0.004	1.00	1.00
	1e-12	11.818	9.000	1.774	1.971	0.003	0.003	1.00	1.00
4	1e-04	12.727	9.545	1.347	1.982	0.018	0.017	1.00	1.00
	1e-06	12.727	9.727	1.346	1.989	0.017	0.018	1.00	1.00
	1e-08	12.727	9.727	1.346	1.987	0.018	0.018	1.00	1.00
	1e-10	12.727	9.727	1.346	1.987	0.017	0.017	1.00	1.00
	1e-12	12.727	9.727	1.346	1.987	0.018	0.018	1.00	1.00
5	1e-04	13.727	9.636	1.905	1.930	0.155	0.133	1.00	1.00
	1e-06	13.727	9.636	1.905	1.924	0.159	0.131	1.00	1.00
	1e-08	13.727	9.636	1.905	1.923	0.160	0.134	1.00	1.00
	1e-10	13.727	9.636	1.905	1.923	0.158	0.132	1.00	1.00
	1e-12	13.727	9.636	1.905	1.923	0.156	0.132	1.00	1.00

## 5.1 Exact gradient and Hessian

The gradient of the banded trigonometric function is given by the following expression.

$$\frac{\partial F}{\partial x_k} = \begin{cases} k \sin x_k + 2 \cos x_k, & 1 \leq k < n \\ n \sin x_n - (n-1) \cos x_n, & k = n \end{cases} \quad (11)$$

The Hessian of the banded trigonometric function is given by the following expression.

$$\frac{\partial^2 F}{\partial x_k \partial x_j} = \begin{cases} k \cos x_k - 2 \sin(x_k), & 1 \leq k = j < n \\ n \cos x_n + (n-1) \sin x_n, & k = j = n \\ 0, & \text{otherwise} \end{cases} \quad (12)$$

Notice that the Hessian is a diagonal matrix, which makes the optimization problem easier to solve. However, the Hessian of the matrix has very distant eigenvalues due to the fact that the  $i$ -th diagonal entry is multiplied by  $i$ : the problem may become increasingly ill-conditioned as the dimension  $n$  increases. Due to this fact, we only perform optimization with preconditioning.

Table 13 shows the results for the *Modified Newton method* applied to the banded trigonometric function with exact gradient and Hessian. Modified newton method converges for all points only when  $n = 10^3$ , while attempts with  $n = 10^4$  and  $n = 10^5$  yield only one success each. This is due to the fact that being the problem badly scaled, in case of a non positive-definite Hessian matrix, the modification of the Hessian matrix may happen with a  $\tau$  that is too large. Figure 5 compares the empirical cumulative distribution function of  $\tau_k$  for the banded trigonometric problem and the extended Rosenbrock function with exact gradient and Hessian. One can notice that while for the extended Rosenbrock function the maximum value of the correction does not depend on the dimension, for the banded trigonometric problem the maximum value of the correction increases with the dimension. Moreover, as suggested by the factor  $i$  that multiplies the  $i$ -th diagonal entry of the Hessian, the maximum correction needed for the dimension  $10^n$  is close to  $10^n$ . While a correction of this magnitude is acceptable for  $n = 10^3$ , it is not for larger values of  $n$ , leading to a Hessian matrix that is too different from the original one and resulting in a poor descent direction. All failures happen because convergence has not been reached within  $k_{max} = 10^3$  iterations.

Table 14 shows the results for the *Truncated Newton method* applied to the banded trigonometric function with exact gradient and Hessian. All attempts are successful, and the method converges in a small number of iterations with a large experimental rate of convergence. So, when preconditioning is adopted, the Truncated Newton method is able to solve the problem efficiently unlike Modified Newton method. Probably adopting another matrix correction strategy that impacts less on the whole matrix, such as the one based on *Modified Cholesky factorization* proposed in [2], could lead to better results.

Table 13: Results for Modified Newton method applied to Banded Trigonometric with exact gradient and hessian, metrics are average metrics for successful attempts. Results are given only with preconditioning adopted.

dimension	iterations	convergence rate	time	success rate
3	14.273	2.782	0.018	1.00
4	7.000	2.946	0.021	0.09
5	7.000	2.946	0.115	0.09

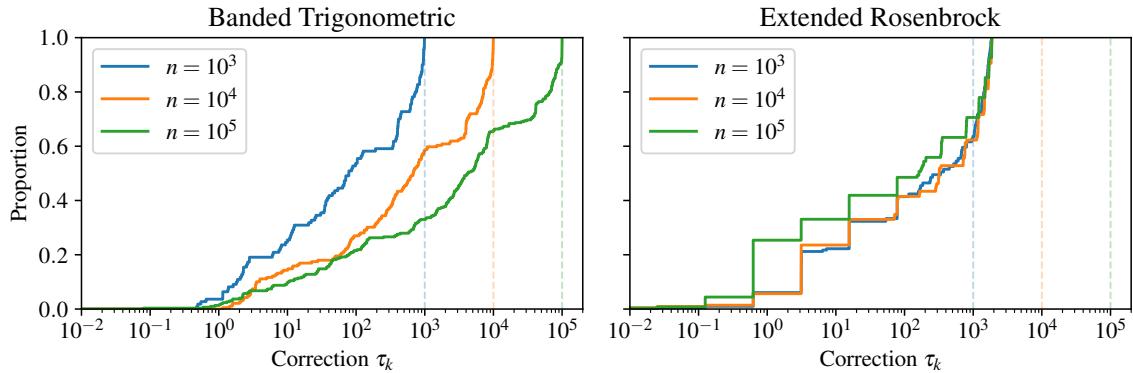


Figure 5: Empirical cumulative distribution function of  $\tau_k$  in banded trigonometric problem and extended Rosenbrock function with exact gradient and Hessian, considering all times a correction is needed across all runs. Dashed lines mark the values  $10^3$ ,  $10^4$  and  $10^5$ .

Table 14: Results for Truncated Newton method applied to Banded Trigonometric with exact gradient and hessian, metrics are average metrics for successful attempts. Results are given only with preconditioning adopted.

dimension	iterations	convergence rate	time	success rate
3	14.273	2.790	0.009	1.00
4	20.818	2.714	0.028	1.00
5	25.273	2.774	0.302	1.00

## 5.2 Finite differences gradient and Hessian

When applying 2, one can notice that the terms  $F(x + he_k)$  and  $F(x - he_k)$  only differ by summands with indices  $i = k$  and  $i = k \pm 1$ . Then to make function evaluations less expensive, we can define the following function  $F_{fd,k}$ , that can be plugged in 2 in place of  $F$  yielding the same result.

$$F_{fd,k} = \sum_{i=k-1}^{k+1} i[(1 - \cos x_i) + \sin x_{i-1} - \sin x_{i+1}]$$

Some of the terms in  $F_{fd,k}$  are constant and do not depend on  $x_k$ , so they will eventually subtract and we can further simplify the expression.

$$F_{fd,k} = \begin{cases} -k \cos x_k + 2 \sin x_k, & 1 \leq k < n \\ n \cos x_n - (n-1) \sin x_n, & k = n \end{cases}$$

The same procedure can be applied to the Hessian, considering that terms in 3 only differ by summands with indices  $i = k$  and  $i = k \pm 1$ . When plugging the function  $F_{fd,k}$  into 2 and 3 it's convenient to expand them so that the computation of the gradient and Hessian is not subject to numerical cancellation. After expanding the functions, the gradient and Hessian can be approximated as follows.

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

$$\frac{\partial^2 F}{\partial x_k^2} \approx \begin{cases} (k \cos x_k - 2 \sin x_k) - h(k \sin x_k + 2 \cos x_k), & 1 \leq k < n \\ (n \cos x_n - (n-1) \sin x_n) - h(n \sin x_n - (n-1) \cos x_n), & k = n \end{cases}$$

The approximation for the gradient is obtained applying the trigonometric identities for the sum of angles. The approximation for the Hessian is obtained by applying the aforementioned trigonometric identities in order to isolate the following terms.

$$\cos(2h) - 2 \cos h \approx -1 - h^2 + \mathcal{O}(h^4), \quad \sin(2h) - 2 \sin h \approx -h^3 + \mathcal{O}(h^5)$$

To avoid numerical cancellation, it is necessary to substitute them with their Taylor expansions.

Tables 15 and 16 show the results for the *Modified Newton method* applied to the banded trigonometric function with absolute and specific finite differences, respectively. In terms of success rate, results are comparable to the ones obtained with exact gradient and Hessian. However, when specific finite differences are adopted, approximation with  $h = 10^{-2}$  and  $h = 10^{-4}$  give poor results, yielding respectively 0% and 9% success rate. On the other hand, when increments are constant, we get at least 9% success rate for all dimensions, all increments. When absolute increment is adopted, all failures happen due to the fact that convergence has not been reached within  $k_{max} = 10^3$  iterations. When specific increment is adopted, most failures happen because the method does not converge within  $k_{max} = 10^3$  iterations, but in 3 cases over 61 failures for  $n = 10^4$  and in 4 cases over 61 failures for  $n = 10^5$ , failure happens because Armijo condition can't be satisfied. We did not perform tuning on parameters  $\rho$  and  $c_1$  of the Armijo condition, since failures are probably due to the fact that the Hessian correction strategy is not suitable for a problem that is very ill-conditioned.

Tables 17 and 18 show the results for the *Truncated Newton method* applied to the banded trigonometric function with absolute and specific finite differences, respectively. All attempts are successful, and the method converges in a small number of iterations with a large experimental rate of convergence. The results are similar to the ones obtained with exact gradient and Hessian, confirming that the Truncated Newton method is a suitable algorithm to tackle the banded trigonometric problem optimization.

## 6 Conclusions

Our experiments provide insights into the practical performance of Newton-type methods for large-scale unconstrained optimization problems. The Modified Newton Method, while robust, can suffer from inefficiencies when dealing with ill-conditioned Hessians, particularly in the banded trigonometric function. On the other hand, the Truncated Newton Method, which relies on iterative solutions to the Newton system, generally achieves better computational efficiency, particularly for high-dimensional problems. However, it is more sensitive to preconditioning, which significantly influences its convergence rate and stability.

Finite difference approximations, though useful in the absence of exact derivatives, introduce additional computational costs and potential inaccuracies, particularly when the step size is not chosen carefully.

Table 15: Results for Modified Newton method applied to Banded Trigonometric with absolute finite differences, metrics are average metrics for successful attempts. Results are given only with preconditioning adopted.

dimension	h	iterations	convergence rate	time	success rate
3	1e-02	14.818	2.548	0.010	1.00
	1e-04	14.545	2.807	0.007	1.00
	1e-06	14.091	2.842	0.006	1.00
	1e-08	14.182	2.575	0.006	1.00
	1e-10	13.364	2.630	0.006	1.00
	1e-12	13.909	2.695	0.006	1.00
4	1e-02	8.000	2.256	0.018	0.09
	1e-04	13.000	2.605	0.032	0.18
	1e-06	6.000	2.857	0.014	0.09
	1e-08	14.500	2.844	0.033	0.18
	1e-10	7.000	2.948	0.015	0.09
	1e-12	7.000	2.946	0.016	0.09
5	1e-02	8.000	2.256	0.123	0.09
	1e-04	8.000	2.951	0.129	0.09
	1e-06	6.000	2.856	0.100	0.09
	1e-08	6.000	2.749	0.098	0.09
	1e-10	7.000	2.948	0.108	0.09
	1e-12	7.000	2.946	0.117	0.09

Table 16: Results for Modified Newton method applied to Banded Trigonometric with specific finite differences, metrics are average metrics for successful attempts. Results are given only with preconditioning adopted.

dimension	h	iterations	convergence rate	time	success rate
3	1e-04	9.000	1.074	0.018	0.09
	1e-06	16.600	1.104	0.007	0.91
	1e-08	13.636	2.582	0.006	1.00
	1e-10	13.455	2.636	0.006	1.00
	1e-12	13.818	2.377	0.006	1.00
4	1e-04	9.000	1.074	0.021	0.09
	1e-06	6.000	2.895	0.014	0.09
	1e-08	6.000	2.750	0.014	0.09
	1e-10	7.000	2.948	0.015	0.09
	1e-12	7.000	2.946	0.017	0.09
5	1e-04	13.000	1.001	0.201	0.09
	1e-06	6.000	2.894	0.121	0.09
	1e-08	6.000	2.749	0.107	0.09
	1e-10	7.000	2.948	0.111	0.09
	1e-12	7.000	2.946	0.121	0.09

Table 17: Results for Truncated Newton method applied to Banded Trigonometric with absolute finite differences, metrics are average metrics for successful attempts. Results are given only with preconditioning adopted.

dimension	h	iterations	convergence rate	time	success rate
3	1e-02	15.091	2.735	0.007	1.00
	1e-04	14.182	2.476	0.006	1.00
	1e-06	14.182	2.488	0.006	1.00
	1e-08	14.273	2.776	0.007	1.00
	1e-10	14.273	2.756	0.007	1.00
	1e-12	14.182	2.758	0.006	1.00
4	1e-02	19.545	2.709	0.034	1.00
	1e-04	19.818	2.609	0.037	1.00
	1e-06	20.727	2.665	0.034	1.00
	1e-08	21.364	2.740	0.036	1.00
	1e-10	20.636	2.731	0.033	1.00
	1e-12	20.455	2.729	0.034	1.00
5	1e-02	24.455	2.717	0.375	1.00
	1e-04	26.727	2.516	0.379	1.00
	1e-06	25.636	2.754	0.384	1.00
	1e-08	25.636	2.770	0.394	1.00
	1e-10	25.727	2.770	0.396	1.00
	1e-12	25.364	2.727	0.424	1.00

Table 18: Results for Truncated Newton method applied to Banded Trigonometric with specific finite differences, metrics are average metrics for successful attempts. Results are given only with preconditioning adopted.

dimension	h	iterations	convergence rate	time	success rate
3	1e-02	15.182	2.257	0.006	1.00
	1e-04	15.091	2.836	0.007	1.00
	1e-06	14.000	2.710	0.007	1.00
	1e-08	14.273	2.800	0.006	1.00
	1e-10	14.545	2.785	0.007	1.00
	1e-12	14.182	2.781	0.006	1.00
4	1e-02	19.273	2.159	0.033	1.00
	1e-04	20.182	2.668	0.034	1.00
	1e-06	20.636	2.790	0.035	1.00
	1e-08	21.000	2.666	0.035	1.00
	1e-10	20.727	2.647	0.033	1.00
	1e-12	20.182	2.746	0.033	1.00
5	1e-02	24.364	2.263	0.372	1.00
	1e-04	25.455	2.733	0.365	1.00
	1e-06	26.273	2.627	0.418	1.00
	1e-08	25.636	2.785	0.393	1.00
	1e-10	25.455	2.770	0.377	1.00
	1e-12	24.909	2.375	0.395	1.00

## A Code snippets

AP: [TODO]

### A.1 Code for method implementations

#### A.1.1 Modified Newton method

Listing 1: Implementation of the Modified Newton method

```
1 function [xk, fk, gradfk_norm, k, T, success, xseq] = ...
2     modified_newton(x0, f, gradf, Hessf, beta, ...
3         kmax, tolgrad, c1, rho, btmax, max_chol_iter, preconditioning, logging,
4         modification_coeff)
5 %
6 %
7 % INPUTS:
8 % x0 = n-dimensional column vector;
9 % f = function handle that describes a function R^n->R;
10 % gradf = function handle that describes the gradient of f;
11 % Hessf = function handle that describes the Hessian of f;
12 % kmax = maximum number of iterations permitted;
13 % tolgrad = value used as stopping criterion w.r.t. the norm of the gradient;
14 % c1 = the factor of the Armijo condition that must be a scalar in (0,1);
15 % rho = fixed factor, lesser than 1, used for reducing alpha0;
16 % btmax = maximum number of steps for updating alpha during the backtracking
17 % strategy.
18 %
19 % OUTPUTS:
20 % xk = the last x computed by the function;
21 % fk = the value f(xk);
22 % gradfk_norm = value of the norm of gradf(xk)
23 % k = index of the last iteration performed
24 % xseq = n-by-k matrix where the columns are the elements xk of the
25 % sequence
26 % btseq = 1-by-k vector where elements are the number of backtracking
27 % iterations at each optimization step.
28 %
29 if nargin < 14
30     modification_coeff = 2;
31 end
32
33 n = length(x0);
34 % Function handle for the armijo condition
35 farmijo = @(fk, alpha, c1_gradfk_pk) ...
36     fk + alpha * c1_gradfk_pk;
37
38 % Initializations
39 if logging
40     xseq = zeros(n, kmax);
41 else
42     xseq = [];
43 end
44 success = 1;
45
46 gradfkseq = zeros(1, kmax);
47 fkseq = zeros(1, kmax);
48 btseq = zeros(1, kmax);
49 pcgiterseq = zeros(1, kmax);
50 correctionseq = zeros(1, kmax);
51 errornormseq = zeros(1, kmax);
```

```

50
51 xk = x0;
52 fk = f(xk);
53 gradfk = gradf(xk);
54 k = 0;
55 gradfk_norm = norm(gradfk);
56
57 Hessfk = Hessf(xk);
58
59 while k < kmax && gradfk_norm >= tolgrad
60     % Compute the descent direction as solution of
61     % Hessf(xk) p = - gradf(xk)
62     %%%%%% L.S. SOLVED WITH BACKSLASH (NOT USED) %%%%%%
63     % pk = -Hessf(xk)\gradfk;
64     %%%%%% L.S. SOLVED WITH pcg %%%%%%
65     % For simplicity: default values for tol and maxit; no preconditioning
66     % pk = pcg(Hessf(xk), -gradfk);
67     % If you want to silence the messages about "solution quality", use
68     % instead:
69     [B, tau] = chol_with_addition(Hessfk, beta, modification_coeff,
70         max_chol_iter);
71     % B = eigenvalue_modification(Hessfk);
72     % B = modchol_ldlt(Hessfk);
73     % TODO Warning: Input tol may not be achievable by PCG - Try to use a
74     % bigger tolerance
75     Hkm = Hessfk + B;
76     if preconditioning
77         try
78             L = ichol(Hkm);
79             [pk, ~, ~, iterk, ~] = pcg(Hkm, -gradfk, 1e-6, 1000, L, L');
80         catch
81             [pk, ~, ~, iterk, ~] = pcg(Hkm, -gradfk);
82         end
83     else
84         [pk, ~, ~, iterk, ~] = pcg(Hkm, -gradfk);
85     end
86
87     % Reset the value of alpha
88     alpha = 1;
89
90     % Compute the candidate new xk
91     xnew = xk + alpha * pk;
92     % Compute the value of f in the candidate new xk
93     fnew = f(xnew);
94
95     c1_gradfk_pk = c1 * gradfk' * pk;
96     bt = 0;
97     % Backtracking strategy:
98     % 2nd condition is the Armijo condition not satisfied
99     while bt < btmax && fnew > farmijo(fk, alpha, c1_gradfk_pk)
100         % Reduce the value of alpha
101         alpha = rho * alpha;
102         % Update xnew and fnew w.r.t. the reduced alpha
103         xnew = xk + alpha * pk;
104         fnew = f(xnew);
105
106         % Increase the counter by one
107         bt = bt + 1;

```

```

108 end
109 if bt == btmax && fnew > farmijo(fk, alpha, c1_gradfk_pk)
110     disp("Armijo condition could not be satisfied!")
111     success = 0;
112     break
113 end
114 errornormseq(k+1) = norm(xnew - xk);
115 % Update xk, fk, gradfk_norm
116 xk = xnew;
117 fk = fnew;
118 gradfk = gradf(xk);
119 gradfk_norm = norm(gradfk);
120 Hessfk = Hessf(xk);

121 % Increase the step by one
122 k = k + 1;

123 if logging
124     xseq(:, k) = xk;
125 end
126 gradfkseq(k) = gradfk_norm;
127 fkseq(k) = fk;
128 btseq(k) = bt;
129 pcgiterseq(k) = iterk;
130 correctionseq(k) = tau;
131
132 end

133 gradfkseq = gradfkseq(1:k);
134 fkseq = fkseq(1:k);
135 btseq = btseq(1:k);
136 pcgiterseq = pcgiterseq(1:k);
137 correctionseq = correctionseq(1:k);
138 errornormseq = errornormseq(1:k);
139 T = table(gradfkseq', fkseq', btseq', pcgiterseq', correctionseq',
140             errornormseq', ...
141             'VariableNames', {'gradient_norm', 'function_value', 'backtrack',
142                               'inner_iterations', 'correction', 'error_norm'});
142 if logging
143     xseq = xseq(:, 1:k);
144 end
145
146 if k >= kmax && gradfk_norm >= tolgrad
147     success = 0;
148 end
149
150
151 end

```

Listing 2: Implementation of *Cholesky with added multiple of the identity*

```

1 function [B, tau] = chol_with_addition(A, beta, coefficient, max_iter)
2
3 mindiag = min(diag(A));
4 if mindiag > 0
5     tau = 0;
6 else
7     tau = -mindiag + beta;
8 end
9
10 sizes = size(A);
11 n = sizes(1);

```

```

12
13 for i = 1:max_iter
14     try chol(A + tau*speye(n));
15         if i > 1
16             end
17             break
18         catch
19             tau = max(coefficient*tau, beta);
20         end
21     end
22 if i >= max_iter
23     tau = 0;
24     disp('Tau could not be found!')
25 end
26
27 B = speye(n) * tau;
28
29 end

```

### A.1.2 Truncated Newton method

Listing 3: Implementation of the Truncated Newton method

```

1 function [xk, fk, gradfk_norm, k, T, success, xseq] = ...
2     truncated_newton(x0, f, gradf, Hessf, ...
3         kmax, tolgrad, c1, rho, btmax, fterms, pcg_maxit, preconditioning,
4         logging)
5 %
6 % Function that performs the Inexact Newton optimization method, using
7 % backtracking strategy for the step-length selection.
8 %
9 % INPUTS:
10 % x0 = n-dimensional column vector;
11 % f = function handle that describes a function R^n->R;
12 % gradf = function handle that describes the gradient of f;
13 % Hessf = function handle that describes the Hessian of f;
14 % kmax = maximum number of iterations permitted;
15 % tolgrad = value used as stopping criterion w.r.t. the norm of the
16 % gradient;
17 % c1 = the factor of the Armijo condition that must be a scalar in (0,1);
18 % rho = fixed factor, lesser than 1, used for reducing alpha0;
19 % btmax = maximum number of steps for updating alpha during the
20 % backtracking strategy.
21 % fterms = function handle taking as input arguments k and gradfk, and
22 % returning the forcing term etak
23 % pcg_maxit = maximum number of iterations for the pcg solver
24 %
25 % OUTPUTS:
26 % xk = the last x computed by the function;
27 % fk = the value f(xk);
28 % gradfk_norm = value of the norm of gradf(xk)
29 % k = index of the last iteration performed
30 % xseq = n-by-k matrix where the columns are the elements xk of the
31 % sequence
32 % btseq = 1-by-k vector where elements are the number of backtracking
33 % iterations at each optimization step.
34 % pcgiterseq = 1-by-k vector where elements are the number of pcg
35 % iterations at each optimization step.
36 %

```

```

35 n = length(x0);
36 % Function handle for the armijo condition
37 farmijo = @(fk, alpha, c1_gradfk_pk) ...
38     fk + alpha * c1_gradfk_pk;
39
40 % Initializations
41 if logging
42     xseq = zeros(n, kmax);
43 else
44     xseq = [];
45 end
46 success = 1;
47
48 gradfkseq = zeros(1, kmax);
49 fkseq = zeros(1, kmax);
50 btseq = zeros(1, kmax);
51 pcgiterseq = zeros(1, kmax);
52 truncatedseq = zeros(1, kmax);
53 errornormseq = zeros(1, kmax);
54
55 xk = x0;
56 fk = f(xk);
57 gradfk = gradf(xk);
58 k = 0;
59 gradfk_norm = norm(gradfk);
60
61 while k < kmax && gradfk_norm >= tolgrad
62     % "INEXACTLY" compute the descent direction as approximated solution of
63     % Hessf(xk) p = - gradf(xk)
64
65     % TOLERANCE VARYING W.R.T. FORCING TERMS:
66     etak = fterms(k, gradfk);
67     % ATTENTION! We will use directly eta_k as tolerance in the pcg because
68     % this function looks at the RELATIVE RESIDUAL and not the RESIDUAL!
69
70     %%%%%% L.S. SOLVED WITH pcg %%%%%%
71     % For simplicity: default values for tol and maxit; no preconditioning
72     % pk = pcg(Hessf(xk), -gradfk, etak, pcg_maxit);
73     % If you want to silence the messages about solution "quality" use
74     % instead:
75     % [pk, flagk, relresk, iterk, resveck] = pcg(Hessf(xk), ...
76     % -gradfk, etak, pcg_maxit);
77     % [pk, ~, iterk] = cg(Hessf(xk), -gradfk, pcg_maxit, etak);
78     Hk = Hessf(xk);
79     if preconditioning
80         try
81             L = ichol(Hk);
82             [pk, ~, iterk, truncated] = cg_preconditioned(Hk, -gradfk,
83                 pcg_maxit, etak, L);
84         catch
85             % If the preconditioner fails, we will use the default one
86             [pk, ~, iterk, truncated] = cg(Hk, -gradfk, pcg_maxit, etak);
87         end
88     else
89         [pk, ~, iterk, truncated] = cg(Hessf(xk), -gradfk, pcg_maxit, etak);
90     end
91
92     % Reset the value of alpha
93     alpha = 1;

```

```

94
95 % Compute the candidate new xk
96 xnew = xk + alpha * pk;
97 % Compute the value of f in the candidate new xk
98 fnew = f(xnew);

99
100 c1_gradfk_pk = c1 * gradfk' * pk;
101 bt = 0;
102 % Backtracking strategy:
103 % 2nd condition is the Armijo condition not satisfied
104 while bt < btmax && fnew > farmijo(fk, alpha, c1_gradfk_pk)
105     % Reduce the value of alpha
106     alpha = rho * alpha;
107     % Update xnew and fnew w.r.t. the reduced alpha
108     xnew = xk + alpha * pk;
109     fnew = f(xnew);

110
111     % Increase the counter by one
112     bt = bt + 1;
113 end
114 if bt == btmax && fnew > farmijo(fk, alpha, c1_gradfk_pk)
115     disp('Armijo condition could not be satisfied!')
116     success = 0;
117     break
118 end
119 errornormseq(k+1) = norm(xnew - xk);
120 % Update xk, fk, gradfk_norm
121 xk = xnew;
122 fk = fnew;
123 gradfk = gradf(xk);
124 gradfk_norm = norm(gradfk);

125
126 % Increase the step by one
127 k = k + 1;

128
129 if logging
130     xseq(:, k) = xk;
131 end
132 gradfkseq(k) = gradfk_norm;
133 fkseq(k) = fk;
134 btseq(k) = bt;
135 pcgiterseq(k) = iterk;
136 truncatedseq(k) = truncated;
137 end
138
139 gradfkseq = gradfkseq(1:k);
140 fkseq = fkseq(1:k);
141 btseq = btseq(1:k);
142 pcgiterseq = pcgiterseq(1:k);
143 truncatedseq = truncatedseq(1:k);
144 errornormseq = errornormseq(1:k);
145 T = table(gradfkseq', fkseq', btseq', pcgiterseq', truncatedseq',
146             errornormseq', ...
147             'VariableNames', {'gradient_norm', 'function_value', 'backtrack',
148             'inner_iterations', 'truncated', 'error_norm'});
149
150 if logging
151     xseq = xseq(:, 1:k);
152 end

```

```

152 if k >= kmax && gradfk_norm >= tolgrad
153     success = 0;
154 end
155
156 end

```

Listing 4: Implementation of conjugate gradient method with truncation

```

1 function [xk, k, relres, truncated] = ...
2     cg(A, b, kmax, tol)
3 % function [xk, k, relres] = ...
4 %     lab01_cg_linsys(A, b, x0, kmax, tol)
5 %
6 % Conjugate Gradient Method for linear systems.
7 %
8 % INPUTS:
9 % A : n-by-n symmetric, positive (semi-)definite matrix of the system
10 % b : column vector of n elements, known terms of the system
11 % x0 : columns vector of n elements, starting guess
12 % kmax : positive integer, maximum number of steps
13 % tol : positive real value, tolerance for relative residual
14
15 xk = zeros(size(b));
16 rk = b - A * xk;
17 pk = rk;
18
19 truncated = 0;
20 k = 0;
21
22 norm_b = norm(b);
23 relres = norm(rk) / norm_b;
24
25 while k < kmax && relres > tol
26     zk = A * pk;
27     if pk' * zk <= 0
28         if k == 0
29             xk = b;
30         end
31         truncated = 1;
32         % disp(['Stopped at iteration ', num2str(k)]);
33         return
34     end
35     alphak = (rk' * pk) / (pk' * zk);
36     xk = xk + alphak * pk;
37     rk = rk - alphak * zk;
38
39     betak = -(rk' * zk) / (pk' * zk);
40     pk = rk + betak * pk;
41
42     relres = norm(rk) / norm_b;
43     k = k + 1;
44 end
45
46 end

```

Listing 5: Implementation of preconditioned conjugate gradient method with truncation

```

1 function [xk, k, relres, truncated] = ...
2     cg_preconditioned(A, b, kmax, tol, L)
3 % function [xk, k, relres] = ...

```

```

4      lab01_cg_linsys(A, b, x0, kmax, tol)
5
6 % Conjugate Gradient Method for linear systems.
7
8 % INPUTS:
9 % A : n-by-n symmetric, positive (semi-)definite matrix of the system
10 % b : column vector of n elements, known terms of the system
11 % x0 : columns vector of n elements, starting guess
12 % kmax : positive integer, maximum number of steps
13 % tol : positive real value, tolerance for relative residual
14
15 xk = zeros(size(b));
16 rk = A * xk - b;
17 wk = L \ rk;
18 yk = L' \ wk;
19 pk = -yk;
20
21 k = 0;
22 truncated = 0;
23
24 norm_b = norm(b);
25 relres = norm(rk) / norm_b;
26
27 while k < kmax && relres > tol
28     zk = A * pk;
29     if pk' * zk <= 0
30         if k == 0
31             xk = b;
32         end
33         truncated = 1;
34         % disp(['Stopped at iteration ', num2str(k)]);
35         return
36     end
37     alphak = (rk' * yk) / (pk' * zk);
38     xk = xk + alphak * pk;
39     rk_new = rk + alphak * zk;
40
41     wk = L \ rk_new;
42     yk_new = L' \ wk;
43
44     betak = (rk_new' * yk_new) / (rk' * yk);
45     pk = -yk_new + betak * pk;
46
47     rk = rk_new;
48     yk = yk_new;
49
50     relres = norm(rk) / norm_b;
51     k = k + 1;
52 end
53
54 end

```

## A.2 Code for objective functions

### A.2.1 Extended Rosenbrock function

Listing 6: Extended Rosenbrock function

```

1 function [f] = extended_rosenbrock(x)
2 n = length(x);

```

```

3 fk_odd = @(x) 10*(x(1:2:n-1).^2 - x(2:2:n));
4 fk_even = @(x) x(1:2:n-1) - 1;
5 f = 0.5*sum(fk_odd(x).^2 + fk_even(x).^2);
6 end

```

Listing 7: Gradient of the Extended Rosenbrock function

```

1 function [gradf] = extended_rosenbrock_gradf(x)
2 n = length(x);
3 gradf = zeros(n,1);
4 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) -
5 1;
6 gradf(2:2:n) = -100*(x(1:2:n-1).^2 - x(2:2:n));
7 end

```

Listing 8: Hessian of the Extended Rosenbrock function

```

1 function [Hessf] = extended_rosenbrock_Hessf(x)
2 n = length(x);
3 Bin = zeros(n,3);
4 Bin(1:2:n-1, 2) = 200*(3*x(1:2:n-1).^2 - x(2:2:n)) + 1;
5 Bin(2:2:n, 2) = 100;
6 Bin(1:2:n-1, 1) = -200*x(1:2:n-1);
7 Bin(2:2:n-2, 1) = 0;
8 Bin(2:n, 3) = Bin(1:n-1, 1);
9 Hessf = spdiags(Bin, -1:1, n, n);
10 end

```

Listing 9: Approximated gradient of the Extended Rosenbrock function

```

1 function [gradf] = extended_rosenbrock_gradf_fd(x, h, relative)
2 n = length(x);
3 if relative
4     hs = h*abs(x);
5 else
6     hs = h*ones(n, 1);
7 end
8
9 gradf = zeros(n, 1);
10 gradf(1:2:n) = 2*x(1:2:n).*hs(1:2:n) - 2*hs(1:2:n)...
11     + 400*x(1:2:n).^3.*hs(1:2:n) + 400*x(1:2:n).*hs(1:2:n).^3 -
12     400*x(1:2:n).*x(2:2:n).*hs(1:2:n);
13 gradf(2:2:n) = -200.*hs(2:2:n).*x(1:2:n).^2 + 200*hs(2:2:n).*x(2:2:n);
14 gradf = gradf ./ (2*hs);
15 end

```

Listing 10: Approximated Hessian of the Extended Rosenbrock function

```

1 function [Hessf] = extended_rosenbrock_Hessf_fd(x, h, relative)
2 n = length(x);
3 Bin = zeros(n, 3);
4 if relative
5     hs = h*abs(x);
6 else
7     hs = h*ones(n, 1);
8 end
9
10 % Diagonal
11 Bin(1:2:n, 2) = 1200*hs(1:2:n).*x(1:2:n) - 200.*x(2:2:n)...
12     + 1 + 700*hs(1:2:n).^2 + 600*x(1:2:n).^2;

```

```

13 Bin(2:2:n, 2) = 100;
14 % Off-diagonal
15 Bin(1:2:n, 1) = -100*hs(1:2:n) - 200*x(1:2:n);
16 Bin(2:2:n, 3) = Bin(1:2:n, 1);
17 Hessf = spdiags(Bin, -1:1, n, n);
18 end

```

### A.2.2 Generalized Broyden tridiagonal function

Listing 11: Generalized Broyden tridiagonal function

```

1 function [f] = extended_rosenbrock(x)
2 n = length(x);
3 fk_odd = @(x) 10*(x(1:2:n-1).^2 - x(2:2:n));
4 fk_even = @(x) x(1:2:n-1) - 1;
5 f = 0.5*sum(fk_odd(x).^2 + fk_even(x).^2);
6 end

```

Listing 12: Gradient of the generalized Broyden tridiagonal function

```

1 % Problem 32
2 function [f] = generalized_broyden(x)
3     fk_function = @(x) (3-2*x).*x + 1 - [0; x(1:end-1)] - [x(2:end); 0];
4     fk = fk_function(x);
5     f = sum(fk.^2)/2;
6 end

```

Listing 13: Hessian of the generalized Broyden tridiagonal function

```

1 function [Hessf] = generalized_broyden_Hessf(x)
2     n = length(x);
3     fk_function = @(x) (3-2*x).*x + 1 - [0; x(1:end-1)] - [x(2:end); 0];
4     fk = fk_function(x);
5     Bin = zeros(n, 5);
6     % Main diagonal
7     Bin(2:n-1, 3) = -4*fk(2:n-1) + (3-4*x(2:n-1)).^2 + 2;
8     Bin(1, 3) = (3-4*x(1)).^2 - 4*fk(1) + 1;
9     Bin(n, 3) = (3-4*x(n)).^2 - 4*fk(n) + 1;
10    % First codiagonal
11    Bin(1:n-1, 2) = 4*x(1:n-1) + 4*x(2:n) - 6;
12    Bin(2:n, 4) = Bin(1:n-1, 2);
13    % Second codiagonal
14    Bin(1:n-2, 1) = 1;
15    Bin(3:n, 5) = 1;
16    % Create banded matrix
17    Hessf = spdiags(Bin, -2:2, n, n);
18 end

```

Listing 14: Approximated gradient of the generalized Broyden tridiagonal function

```

1 function [gradf] = generalized_broyden_gradf_fd(x, h, relative)
2     n = length(x);
3     if relative
4         hs = h*abs(x);
5     else
6         hs = h*ones(n, 1);
7     end
8     xm2 = [0; 0; x(1:end-2)];
9     xm1 = [0; x(1:end-1)];
10    xp1 = [x(2:end); 0];

```

```

11 xp2 = [x(3:end); 0; 0];
12 gradf = 2*xm1.^2 + 4*xm1.*x - 6*xm1 + 8*x.^3 - 18*x.^2 + 4*x.*xp1 +
13     8*x.*hs.^2 + 7.*x ...
14     + 2*xp1.^2 - 6*xp1 - 6*hs.^2 + xm2 + xp2 + 1;
15 gradf(1) = 6*x(1) - 6*x(2) + x(3) + 4*x(1)*x(2) + 8*x(1)*hs(1)^2 ...
16     - 18*x(1)^2 + 8*x(1)^3 + 2*x(2)^2 - 6*hs(1)^2 + 2;
17 gradf(n) = x(n-2) - 6*x(n-1) + 6*x(n) + 4*x(n-1)*x(n) + 8*x(n)*hs(n)^2
18     ...
19     + 2*x(n-1)^2 - 18*x(n)^2 + 8*x(n)^3 - 6*hs(n)^2 + 2;
20 end

```

Listing 15: Approximated Hessian of the generalized Broyden tridiagonal function

```

1 function [Hessf] = generalized_broyden_Hessf_fd(x, h, relative)
2 n = length(x);
3 if relative
4     hs = h*abs(x);
5 else
6     hs = h*ones(n, 1);
7 end
8 Bin = zeros(n, 5);
9
10 xm1 = [0; x(1:n-1)];
11 xp1 = [x(2:n); 0];
12
13 % Diagonal
14 Bin(1:n, 3) = 24*x.^2 + 48*x.*hs - 36*x + 28*hs.^2 - 36*hs + 4*xm1 + 4*xp1 +
15     7;
16 Bin(1, 3) = 24*x(1)^2 + 48*x(1)*hs(1) - 36*x(1) + 28*hs(1)^2 - 36*hs(1) +
17     4*x(2) + 6;
18 Bin(n, 3) = 24*x(n)^2 + 48*x(n)*hs(n) - 36*x(n) + 28*hs(n)^2 - 36*hs(n) +
19     4*x(n-1) + 6;
20 % First codiagonal
21 Bin(1:n-1, 2) = 4*x(1:n-1) + 4*x(2:n) + 2*hs(1:n-1) + 2*hs(2:n) - 6;
22 Bin(2:n, 4) = Bin(1:n-1, 2);
23 % Second codiagonal
24 Bin(1:n-2, 1) = 1;
25 Bin(3:n, 5) = 1;
26
27 Hessf = spdiags(Bin, -2:2, n, n);
28 end

```

### A.2.3 Banded trigonometric problem

Listing 16: Banded trigonometric problem

```

1 function [f] = banded_trigonometric(x)
2     n = length(x);
3     x_m1 = [0; x(1:n-1)];
4     x_p1 = [x(2:n); 0];
5     i = (1:n)';
6     f = sum(i.*(1-cos(x) + sin(x_m1) - sin(x_p1)));
7 end

```

Listing 17: Gradient of the banded trigonometric problem

```

1 function [gradf] = banded_trigonometric_gradf(x)
2     n = length(x);
3     i = (2:n-1)';
4     gradf = zeros(n, 1);

```

```

5 gradf(2:n-1) = i.*sin(x(2:n-1)) + 2*cos(x(2:n-1));
6 gradf(1) = sin(x(1)) + 2*cos(x(1));
7 gradf(n) = n*sin(x(n)) - (n-1)*cos(x(n));
8 end

```

Listing 18: Hessian of the banded trigonometric problem

```

1 function [Hessf] = banded_trigonometric_Hessf(x)
2     n = length(x);
3     diag_entries = zeros(n, 1);
4     i = (2:n-1)';
5     diag_entries(2:n-1) = i.*cos(x(2:n-1)) - 2*sin(x(2:n-1));
6     diag_entries(1) = cos(x(1)) - 2*sin(x(1));
7     diag_entries(n) = n*cos(x(n)) + (n-1)*sin(x(n));
8     Hessf = spdiags(diag_entries, 0, n, n);
9 end

```

Listing 19: Approximated gradient of the banded trigonometric problem

```

1 function [gradf] = banded_trigonometric_gradf_fd(x, h, relative)
2     n = length(x);
3     if relative
4         hs = h*abs(x);
5     else
6         hs = h*ones(n, 1);
7     end
8
9     i = (1:n-1)';
10
11    gradf = [
12        2*i.*sin(x(1:n-1)).*sin(hs(1:n-1)) + 4*cos(x(1:n-1)).*sin(hs(1:n-1));
13        2*n.*sin(x(n)).*sin(hs(n)) - 2*(n-1)*cos(x(n)).*sin(hs(n));
14    ];
15    gradf = gradf ./ (2*hs);
16 end

```

Listing 20: Approximated Hessian of the banded trigonometric problem

```

1 function [Hessf] = banded_trigonometric_Hessf_fd(x, h, relative)
2     n = length(x);
3     if relative
4         hs = h*abs(x);
5     else
6         hs = h*ones(n, 1);
7     end
8
9     i = (1:n)';
10    diagonal = (-i.*cos(x) + 2*sin(x)).*(-1) ...
11        + (i.*sin(x) + 2*cos(x)).*(-hs);
12    diagonal(n) = (-n.*cos(x(n)) - (n-1)*sin(x(n))).*(-1) ...
13        + (n.*sin(x(n)) - (n-1)*cos(x(n))).*(-hs(n));
14    Hessf = spdiags(diagonal, 0, n, n);
15 end

```

### A.3 Utility code for running experiments

Listing 21: Script to run all the experiments for all functions

```

1 clear all; close all; clc; %#ok<CLALL>
2

```

```

3 load('data/test_functions2.mat');
4 load('data/forcing_terms.mat');

5
6 % Seed for reproducibility
7 seed = min([331794 337131 338682]);
8
9 % //TODO rosenbrock
10
11 fterm = @(k, gradf) fterms_suplin(k, gradf);
12 kmax = dictionary([3, 4, 5], [1e3, 1e3, 1e3]);
13 modified_coeffs = [5, 2, 2];
14
15 rng(seed);
16 disp('**** CHAINED ROSENROCK ****')
17 codiags = 1;
18 test(@(x) extended_rosenbrock(x), @(x) extended_rosenbrock_gradf(x), ...
19     @(x)extended_rosenbrock_Hessf(x), @(n)
20         extended_rosenbrock_initializer(n), ...
21     @extended_rosenbrock_gradf_fd, @extended_rosenbrock_Hessf_fd, codiags, ...
22     kmax, 1e-6, 1e-4, 0.5, 50, 100, 1e-3, fterm, 100,
23         '../results/extended_rosenbrock/', true, 5);
24
25 rng(seed);
26 disp('**** GENERALIZED BROYDEN ****')
27 codiags = 2;
28 test(@(x) generalized_broyden(x), @(x) generalized_broyden_gradf(x), ...
29     @(x)generalized_broyden_Hessf(x), @(n)
30         generalized_broyden_initializer(n), ...
31     @generalized_broyden_gradf_fd, @generalized_broyden_Hessf_fd, codiags, ...
32     kmax, 1e-6, 1e-4, 0.5, 50, 100, 1e-3, fterm, 100,
33         '../results/generalized_broyden/', true, 2);
34
35 rng(seed);
36 disp('**** BANDED TRIGONOMETRIC ****')
37 codiags = 0;
38 test(@(x) banded_trigonometric(x), @(x) banded_trigonometric_gradf(x), ...
39     @(x)banded_trigonometric_Hessf(x), @(n)
40         banded_trigonometric_initializer(n), ...
41     @banded_trigonometric_gradf_fd, @banded_trigonometric_Hessf_fd,
42         codiags, ...
43     kmax, 1e-6, 1e-4, 0.5, 50, 100, 1e-3, fterm, 100,
44         '../results/banded_trigonometric/', true, 5);

```

Listing 22: Script to run all the experiments for a given function

```

1 function [] = test(f, gradf, Hessf, initializer, gradf_fd, Hessf_fd,
2     codiags, ...
3     kmax, tolgrad, c1, rho, btmax, chol_maxit, beta, fterms, pcg_maxit,
4     root_dir, findiff, tau_coeff)
5 %
6 % INPUTS
7 % n = dimension of the problem;
8 % f = function handle that describes a function R^n -> R;
9 % gradf = function handle that describes the gradient of f;
10 % Hessf = function handle that describes the Hessian of f;
11 % initializer = function handle that generate starting point x0;
12 % kmax = maximum number of iterations permitted;
13 % tolgrad = value used as stopping criterion w.r.t. the norm of the gradient;
14 % c1 = the factor of the Armijo condition that must be a scalar in (0,1);
15 % rho = fixed factor, lesser than 1, used for reducing alpha0;

```

```

14 % btmax = maximum number of steps for updating alpha during the backtracking
15 % strategy.
16 % chol_maxit = maximum number of iterations for the Cholesky factorization
17 % beta = fixed factor, greater than 1, used for the Cholesky factorization;
18 % fterms = function handle taking as input arguments k and gradfk, and
19 % returning the forcing term etak
20 % pcg_maxit = maximum number of iterations for the pcg solver
21 % h = step size for the finite difference approximation of the gradient and
22 % Hessian.
23 %
24 % OUTPUTS
25 %
26 if ~exist(root_dir, 'dir')
27     mkdir(root_dir);
28 end
29 experiment = 1;
30
31 if nargin < 16
32     findiff = false;
33 end
34
35 for i=[3, 4, 5]
36     % Dimension of the problem
37     n=10^i;
38     % Initialization of the starting point
39     x0 = initializer(n);
40
41     % Run the optimization algorithm for 11 different starting points
42     for j=0:10
43         if j == 0
44             x0_j = x0;
45         else
46             x0_j = 2 * rand(n, 1) + x0 - 1;
47         end
48         disp([9, '* STARTING POINT: (dim:1e', num2str(i), ', test point #',
49               num2str(j), ')'])
50         disp([9, 9, '- EXACT GRADIENT AND HESSIAN ****'])
51         % Exact gradient and Hessian
52         for pre = [0, 1]
53             [fk_m, gradfk_norm_m, k_m, T_m, time_m, success_m, ...
54              fk_t, gradfk_norm_t, k_t, T_t, time_t, success_t] =
55              run_optimization(x0_j, f, gradf, Hessf, beta, kmax(i),
56                               tolgrad, c1, rho, btmax, chol_maxit, fterms, pcg_maxit,
57                               pre, tau_coeff);
58             logger(root_dir, experiment, success_m, 0, pre, i, j, 1, 0, 0,
59                   fk_m, gradfk_norm_m, k_m, T_m, time_m);
60             experiment = experiment+1;
61             logger(root_dir, experiment, success_t, 1, pre, i, j, 1, 0, 0,
62                   fk_t, gradfk_norm_t, k_t, T_t, time_t);
63             experiment = experiment+1;
64         end
65         if findiff
66             % Finite difference gradient and Hessian for different values of
67             % h
68             for k=2:2:12
69                 h = 10^(-k);
70
71                 % Absolute
72                 disp([9, 9, '- ABSOLUTE FINITE DIFFERENCE GRADIENT AND
73                       HESSIAN - h=', num2str(h), ' ****'])
74             end
75         end
76     end
77 end

```

```

63      % findiff_gradf = @(x) findiff_grad(f, x, h, 'c', false);
64      findiff_gradf = @(x) gradf_fd(x, h, false);
65      for pre = [0, 1]
66          [fk_m, gradfk_norm_m, k_m, T_m, time_m, success_m, ...
67           fk_t, gradfk_norm_t, k_t, T_t, time_t, success_t] =
68              run_optimization(x0_j, f, findiff_gradf, @(x)
69              Hessf_fd(x, h, false), beta, kmax(i), tolgrad,
70              c1, rho, btmax, chol_maxit, fterms, pcg_maxit,
71              pre, tau_coeff);
72          logger(root_dir, experiment, success_m, 0, pre, i, j, 0,
73                 h, 1, fk_m, gradfk_norm_m, k_m, T_m, time_m);
74          experiment = experiment+1;
75          logger(root_dir, experiment, success_t, 1, pre, i, j, 0,
76                 h, 1, fk_t, gradfk_norm_t, k_t, T_t, time_t);
77          experiment = experiment+1;
78      end
79      % Relative
80      % findiff_gradf = @(x) findiff_grad(f, x, h, 'c', true);
81      findiff_gradf = @(x) gradf_fd(x, h, true);
82      disp([9, 9, '- RELATIVE FINITE DIFFERENCE GRADIENT AND
83            HESSIAN - h= ', num2str(h), ' ****'])
84      for pre = [0, 1]
85          [fk_m, gradfk_norm_m, k_m, T_m, time_m, success_m, ...
86           fk_t, gradfk_norm_t, k_t, T_t, time_t, success_t] =
87              run_optimization(x0_j, f, findiff_gradf, @(x)
88              Hessf_fd(x, h, true), beta, kmax(i), tolgrad, c1,
89              rho, btmax, chol_maxit, fterms, pcg_maxit, pre,
90              tau_coeff);
91          logger(root_dir, experiment, success_m, 0, pre, i, j, 0,
92                 h, 0, fk_m, gradfk_norm_m, k_m, T_m, time_m);
93          experiment = experiment+1;
94          logger(root_dir, experiment, success_t, 1, pre, i, j, 0,
95                 h, 0, fk_t, gradfk_norm_t, k_t, T_t, time_t);
96          experiment = experiment+1;
97      end
98  end
99 end

```

Listing 23: Script to run experiments for a given configuration

```

1 function [fk_m, gradfk_norm_m, k_m, T_m, time_m, success_m, ...
2     fk_t, gradfk_norm_t, k_t, T_t, time_t, success_t] =
3     run_optimization(x0, f, gradf, Hessf, beta, ...
4     kmax, tolgrad, c1, rho, btmax, chol_maxit, fterms, pcg_maxit, pre,
5     tau_coeff)

6 disp([9, 9, 9, 'MODIFIED NEWTON *****'])
7 disp([9, 9, 9, 'PRECONDITIONING: ', num2str(pre)])
8 tic;
9 [~, fk_m, gradfk_norm_m, k_m, T_m, success_m, ~] = ...
10    modified_newton(x0, f, gradf, Hessf, beta, ...
11    kmax, tolgrad, c1, rho, btmax, chol_maxit, pre, false, tau_coeff);
12 time_m = toc;
13 disp([9, 9, 9, 'f(xk): ', num2str(fk_m)])
14 disp([9, 9, 9, 'gradfk_norm: ', num2str(gradfk_norm_m)])
15 disp([9, 9, 9, 'k: ', num2str(k_m)])
16 disp([9, 9, 9, '*****'])

```

```

16 disp([9, 9, 9, 'TRUNCATED NEWTON'])
17 disp([9, 9, 9, 'PRECONDITIONING: ', num2str(pre)])
18 tic;
19 [~, fk_t, gradfk_norm_t, k_t, T_t, success_t, ~] = ...
20     truncated_newton(x0, f, gradf, Hessf, ...
21         kmax, tolgrad, c1, rho, btmax, fterms, pcg_maxit, pre, false);
22 time_t = toc;
23 disp([9, 9, 9, 'f(xk): ', num2str(fk_t)])
24 disp([9, 9, 9, 'gradfk_norm: ', num2str(gradfk_norm_t)])
25 disp([9, 9, 9, 'k: ', num2str(k_t)])
26 disp([9, 9, 9, '*****' ])
27
28
29 end

```

Listing 24: Logger to CSV files

```

1 function [] = logger(root_dir, i, success, method, pre, dimension,
2     point_number, exact, h, absolute, fk, gradfk, k, T, time)
3     % Create the directory if it does not exist
4     if ~exist(root_dir, 'dir')
5         mkdir(root_dir);
6     end
7     % Save the results
8     writetable(T, [root_dir, 'experiment_', num2str(i), '.csv']);
9     % Append the experiment results to the log csv file
10    writematrix([i, success, method, pre, dimension, point_number, exact, h,
11        absolute, fk, gradfk, k, time], [root_dir, 'results.csv'],
12        'WriteMode', 'append');
13 end

```

## References

- [1] Ladislav Luksan and Jan Vlček. *Test Problems for Unconstrained Optimization*. Nov. 2003.
- [2] Jorge Nocedal and Stephen J. Wright. “Numerical optimization”. English (US). In: *Springer Series in Operations Research and Financial Engineering*. Springer Series in Operations Research and Financial Engineering. Springer Nature, 2006, pp. 1–664.