OPT201. Differentiable optimization I (2024)

Practical Report

Implementation of local SQP : Sequential Quadratic Programming

Tingjia ZHANG & Abdoulaye TRAORE

26 November - 06 January 2025

Lead Instructor :
M. Sorin Mihai GRAD
Professor at UMA

Teaching Assistant :
M. PINTA Titus
Ph.D Student

Table des matières

Introduction

During this project, we have used Python.

As part of this practical session, we focus on the **Sequential Quadratic Programming** (SQP) method for the iterative resolution of constrained differentiable optimization problems using a local SQP algorithm.

The main objective is to numerically solve the following quadratic problem :

$$\inf_{x=(x_1,\dots,x_5)^\top \in \mathbb{R}^5} \left\{ e^{\prod_{i=1}^{5} x_i} - \frac{1}{2}\left(x_1^3 + x_2^3 + 1\right)^2 \right\},$$

$$\begin{aligned} &\sum_{i=1}^{5} x_i^2 = 10, \\ &x_2 x_3 = 5 x_4 x_5, \\ &x_1^3 + x_2^3 = -1. \end{aligned} \tag{P24}$$

The problem introduces specific challenges, which we address progressively throughout the report :

1. **Formulation of the problem and discussion of the conditions for applying the KKT theorem.**
2. **Description of the local SQP algorithm** : We will introduce the local SQP algorithm based on the Lagrange-Newton method.
3. **Computation of the gradient, Jacobian, and Hessian matrix.**
4. **Justification of the conditions for applying the Lagrange-Newton algorithm to our problem.**
5. **Implementation in Python** : We will present our Python implementation of the local SQP algorithm.
6. **Solutions obtained and relevant quantities.**
7. **Analysis of the obtained solutions and comments.**
8. **Comparision with solutions from open-source solver.**

Additionally, the goal of this project is not only to apply the theoretical knowledge learned during this course but also to gain more hands-on experience in programming.

1 Problem formulation and discussion of conditions for the KKT theorem

1.1 Formulation of the problem

The optimization problem to be solved is :

$$\begin{aligned}
&\inf_{x=(x_1,\dots,x_5)^\top\in\mathbb{R}^5} \left\{ e^{\prod_{i=1}^5 x_i} - \frac{1}{2}\left(x_1^3+x_2^3+1\right)^2 \right\}, \\
&\sum_{i=1}^5 x_i^2 = 10, \\
&x_2x_3 = 5x_4x_5, \\
&x_1^3+x_2^3=-1.
\end{aligned} \tag{P24}$$

The constraint $x_1^3+x_2^3=-1$ simplifies the second term in the objective function.
Notice that plugging a constraint in the objective function can sometimes generate some errors due to hidden variables. In this case, keeping the condition ensure that the problem are equivalente.

Since the exponential function e^x is strictly increasing, minimizing $e^{\prod_{i=1}^5 x_i}$ is equivalent to minimizing the product :

$$\inf_{x=(x_1,\dots,x_5)^\top\in\mathbb{R}^5} \prod_{i=1}^5 x_i.$$

Thus, the equivalence holds :

$$\inf_{x=(x_1,\dots,x_5)^\top\in\mathbb{R}^5} e^{\prod_{i=1}^5 x_i} \iff \inf_{x=(x_1,\dots,x_5)^\top\in\mathbb{R}^5} \prod_{i=1}^5 x_i.$$

Let introduce the $\mathcal{C}^2$ functions $f, g_j : \mathbb{R}^5 \to \mathbb{R}$, $j \in I_3$, denote $g=(g_1,\dots,g_3)^\top : \mathbb{R}^5 \to \mathbb{R}^3$.

$$f(x) = \prod_{i=1}^5 x_i \quad \text{and} \quad g(x) = \begin{bmatrix} \sum_{i=1}^5 x_i^2 - 10 \\ x_2x_3 - 5x_4x_5 \\ x_1^3+x_2^3+1 \end{bmatrix}$$

The primal optimization problem becomes :

$$\boxed{\inf_{x\in A} f(x)} \tag{PCE}$$

where $A = \{x \in \mathbb{R}^5 : g(x) = 0_3\}$.

1.2 Karush–Kuhn–Tucker (KKT) Conditions

$$\boxed{\inf_{x\in A} f(x)} \tag{PCE}$$

where $A = \{x \in \mathbb{R}^5 : g(x) = 0_3\}$ and functions defined as :

$$f(x) = \prod_{i=1}^{5} x_i, \quad g(x) = \begin{bmatrix} \sum_{i=1}^{5} x_i^2 - 10 \\ x_2x_3 - 5x_4x_5 \\ x_1^3 + x_2^3 + 1 \end{bmatrix}.$$

Characterization of the Set A and function f

- — The set A is not empty (**feasibility**).
- — The set A is bounded and closed (**compactness**).
- — The function $f(x)$ is continuous over A (**continuity**).

Feasibility

The feasible set A is governed by the following constraints :

$$1.\ \sum_{i=1}^{5} x_i^2 = 10,$$
$$2.\ x_2x_3 = 5x_4x_5,$$
$$3.\ x_1^3 + x_2^3 = -1.$$

To confirm that A is not empty, consider the point :

$$x = (x_1, x_2, x_3, x_4, x_5) = (-1, 0, 3, 0, 0).$$

We get :

- — $(-1)^2 + 0^2 + 3^2 + 0^2 + 0^2 = 10,$
- — $0 \times 3 = 5 \times 0 \times 0 = 0,$
- — $(-1)^3 + 0^3 = -1.$

Hence, the set A contains at least one point, proving its non-emptiness.

Compactness of the Feasible Set

The first condition (hence $|x_i| \leq \sqrt{10}$ for $i \in I_5$) restricts A to lie on the surface of a 5-dimensional sphere with radius $\sqrt{10}$.
The second and third conditions impose additional nonlinear equality constraints. These constraints ensure that no point in A can have an arbitrarily large norm. Moreover, since g is componentwise continuous, A is closed as the preimage of the closed set $\{0\}$ under a continuous mapping.
By combining these properties, A is compact because it is both closed and bounded in finite-dimensional space ($\mathbb{R}^5$).

Continuity of the Objective Function :

The function to be minimized is :

$$f(x) = \prod_{i=1}^{5} x_i.$$

is clearly continue over R^5 because it's polynomial in components of x..

Existence and uniqueness of the solution

We conclude that the problem (PCE) has at least one solution, denote by $\bar{\mathbf{x}}$, since f is continuous on the compact set A according to Weierstrass theorem.

Notice that the uniqueness of the solution can't be establish in general because A and f aren't convexe. So, we can obtain numerous local optimal solutions.

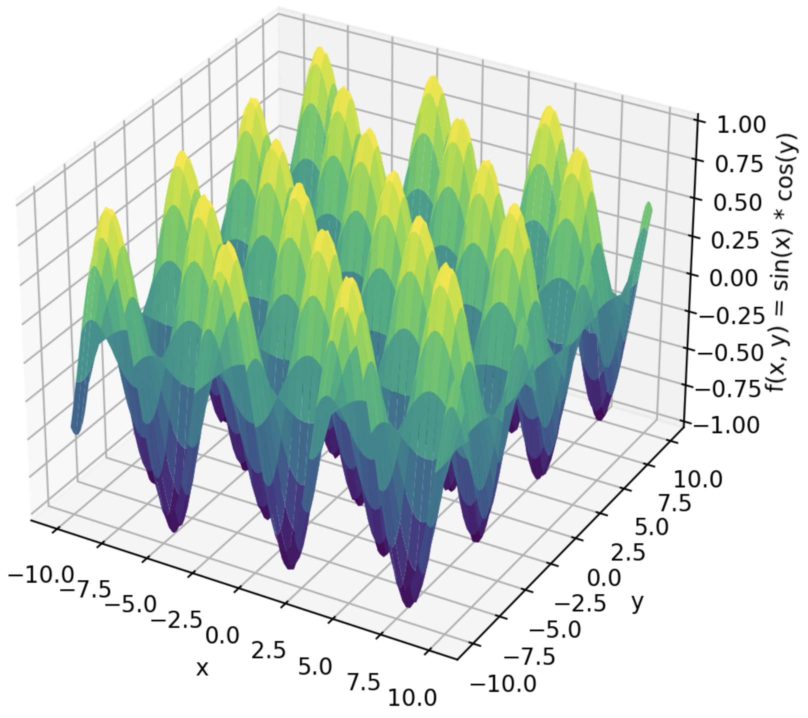


FIGURE 1.1 – Illustration of multiple local optima

Karush–Kuhn–Tucker system

As $\bar{x}$ is a local optimal solution to (PCE) and ACQ(Abadie's Constraint Qualification) are fullfilled, we know that there exist $\lambda \in R^3$ such that $\bar{x}$ satisfies the KKT system associated to (PCE) according to the theorem 60 [refers to Lecture Notes].

The lagrangian function associated to the problem (PCE) is :

$\mathcal{L} : \mathbb{R}^5 \times \mathbb{R}^3 \to \mathbb{R}, \quad \mathcal{L}(x, \lambda) = f(x) + \lambda^\top g(x)$

Thus, the Karush–Kuhn–Tucker (KKT) system associated with this optimization problem is :

$$\begin{aligned} \nabla f(\bar{x}) + J_g(\bar{x})^\top \lambda = 0, &\quad \text{(stationarity)} \\ g(\bar{x}) = 0, &\quad \text{(feasibility)} \end{aligned}$$

where $J_g(\bar{x})$ represents the Jacobian matrix of $g(x)$, and $\lambda \in R^3$ is the vector of Lagrange multipliers.

2 Description of the local SQP algorithm

The local Sequential Quadratic Programming (SQP) algorithm is an iterative method designed to solve constrained optimization problems. It generates a sequence that converges to a KKT-point of the original problem (PCE) by solving a quadratic subproblem (PQ_k) and its associated linear system (LE_k) at each iteration.

2.1 Original Problem PCE

The original constrained optimization problem is defined as :

$$\boxed{\inf_{x\in\mathbb{R}^n} f(x) \quad \text{subject to} \quad g(x) = 0,} \tag{PCE}$$

where :

$$f(x) = \prod_{i=1}^{5} x_i, \quad g(x) = \begin{bmatrix} \sum_{i=1}^{5} x_i^2 - 10 \\ x_2x_3 - 5x_4x_5 \\ x_1^3 + x_2^3 + 1 \end{bmatrix}.$$

Equality System (LE_k)

Let $(x^0, \lambda^0) \in \mathbb{R}^5 \times \mathbb{R}^3$ and the sequences generated by :

$$\begin{pmatrix} x^{k+1} \\ \lambda^{k+1} \end{pmatrix} = \begin{pmatrix} x^k \\ \lambda^k \end{pmatrix} + \begin{pmatrix} a^k \\ b^k \end{pmatrix}, \quad k \in \mathbb{N},$$

where, for each $k \in \mathbb{N}$, $(a^k, b^k) \in \mathbb{R}^n \times \mathbb{R}^m$ solves the equality system :

$$(\mathrm{LE}_k) \quad \underbrace{\begin{pmatrix} H_x\mathcal{L}(x^k, \lambda^k) & J_g(x^k)^\top \\ J_g(x^k) & 0_{m\times m} \end{pmatrix}}_{=:Q(x^k,\lambda^k)} \begin{pmatrix} a^k \\ b^k \end{pmatrix} = -\begin{pmatrix} \nabla f(x^k) + \left((\lambda^k)^\top J_g(x^k)\right)^\top \\ g(x^k) \end{pmatrix}.$$

Here :
- $H_x\mathcal{L}(x^k, \lambda^k)$ is the Hessian matrix of the Lagrangian,
- $J_g(x^k)$ is the Jacobian matrix of the constraints.

Quadratic Subproblem (PQ_k)

Next, we introduce a family of quadratic problems whose KKT-points generate a sequence that converges toward a KKT-point of (PCE). For a fixed $k \in \mathbb{N}$, taking $\zeta^k := \lambda^k + b^k$, (LE_k) becomes :

$$\boxed{(QS_k) \quad \begin{pmatrix} H_x\mathcal{L}(x^k, \lambda^k)a^k + \nabla f(x^k) + J_g(x^k)^\top \zeta^k \\ g(x^k) + J_g(x^k)a^k \end{pmatrix} = 0_{n+m},}$$

which actually represents the KKT-system of the quadratic problem :

$$\boxed{(PQ_k) \quad \inf_{a\in\mathbb{R}^n} \left\{ \frac{1}{2} a^\top H_x\mathcal{L}(x^k, \lambda^k)a + a^\top \nabla f(x^k) \right\}, \quad \text{subject to :} \quad g(x^k) + J_g(x^k)a = 0.}$$

For any $k \in \mathbb{N}$, as (PQ_k) has only affine constraints, no constraint qualification is required to derive its necessary optimality conditions.

2.2 Algorithm Steps

The local SQP algorithm can now be described as follows :

S.0 Initialize $x^0 \in \mathbb{R}^n$, $\lambda^0 \in \mathbb{R}^m$, and set $k = 0$.

S.1 Check the stopping criterion : If (x^k, λ^k) satisfies the KKT conditions of (PCE), then stop.

S.2 Solve the quadratic subproblem (PQ_k), or equivalently, the linear system (LE_k), to compute (a^k, b^k) :

$$Q(x^k, \lambda^k) \begin{bmatrix} a^k \\ b^k \end{bmatrix} = - \begin{bmatrix} \nabla f(x^k) + J_g(x^k)^\top \lambda^k \\ g(x^k) \end{bmatrix}.$$

S.3 Update the iterates :

$$x^{k+1} = x^k + a^k, \quad \lambda^{k+1} = \lambda^k + b^k.$$

S.4 Increment $k = k + 1$ and return to step [S1].

Convergence Analysis

The convergence of the algorithm depends on the following conditions :

1. The iterates $(\bar{x}^0, \bar{\lambda}^0)$ lie in a neighborhood of a KKT-pair $(\bar{x}, \bar{\lambda})$ of (PCE).
2. The matrix $Q(\bar{x}, \bar{\lambda})$ is invertible.
3. A constraint qualification condition, such as Abadie's Constraint Qualification (ACQ), holds.

Indeed, due to the **Convergence of the Lagrange-Newton Algorithm theorem**, if $(\bar{x}, \bar{\lambda}) \in \mathbb{R}^5 \times \mathbb{R}^3$ is a KKT-pair of (PCE) and the matrix $Q(\bar{x}, \bar{\lambda})$ is invertible, then there exists an $\epsilon > 0$ such that for any starting point $(x^0, \lambda^0) \in B_\epsilon(\bar{x}, \bar{\lambda})$, either :

$$(x^k, \lambda^k) = (\bar{x}, \bar{\lambda}) \quad \text{for some } k \in \mathbb{N}, \quad \text{or} \quad (x^k, \lambda^k) \xrightarrow{k \to \infty} (\bar{x}, \bar{\lambda}).$$

Stopping Criterion

The algorithm terminates when :

$$\|\nabla_x \mathcal{L}(x^k, \lambda^k)\| \leq \epsilon \quad \text{or} \quad \|x^{k+1} - x^k\| \leq \epsilon,$$

where $\epsilon > 0$ is a predefined tolerance.

2.3 Conclusion

The local SQP algorithm iteratively solves the quadratic subproblems (PQ_k) by leveraging the linear system (LE_k). The associated KKT system (QS_k) ensures that the updates are consistent with the original problem. Under appropriate conditions, this approach guarantees convergence to the solution of (PCE).

3 Computation of the Gradient, Jacobian, and Hessian Matrix

3.1 Gradient, Jacobian, and Hessian of the Functions

In the context of the local SQP algorithm, the functions of interest include :

— The objective function $f(x) = \prod_{i=1}^{5} x_i$,

— The constraint function $g(x) : \mathbb{R}^5 \to \mathbb{R}^3$, defined as :

$$g(x) = \begin{bmatrix} \sum_{i=1}^{5} x_i^2 - 10 \\ x_2x_3 - 5x_4x_5 \\ x_1^3 + x_2^3 + 1 \end{bmatrix}.$$

We don't detail the computation of these quantities, as they are considered elementary.

3.2 Gradient of the Objective function and Jacobian of the Constraint function

The gradient of $f(x)$ and the Jacobian of $g(x)$ are given by :

$$\nabla f(x) = \begin{bmatrix} x_2x_3x_4x_5 \\ x_1x_3x_4x_5 \\ x_1x_2x_4x_5 \\ x_1x_2x_3x_5 \\ x_1x_2x_3x_4 \end{bmatrix}, \quad J_g(x) = \begin{bmatrix} 2x_1 & 2x_2 & 2x_3 & 2x_4 & 2x_5 \\ 0 & x_3 & x_2 & -5x_5 & -5x_4 \\ 3x_1^2 & 3x_2^2 & 0 & 0 & 0 \end{bmatrix}.$$

3.3 Hessian of the Lagrangian

The Hessian matrix $H_x\mathcal{L}(x, \lambda) \in \mathbb{R}^{5\times5}$ is computed as :

$$H_x\mathcal{L}(x, \lambda) = \nabla^2 f(x) + \sum_{j=1}^{3} \lambda_j \nabla^2 g_j(x).$$

The total Hessian of the Lagrangian is :

$$H_x\mathcal{L}(x, \lambda) = \begin{bmatrix} 2\lambda_1 + 6\lambda_3x_1 & x_3x_4x_5 & x_2x_4x_5 & x_2x_3x_5 & x_2x_3x_4 \\ x_3x_4x_5 & 2\lambda_1 + 6\lambda_3x_2 & x_1x_4x_5 + \lambda_2 & x_1x_3x_5 & x_1x_3x_4 \\ x_2x_4x_5 & x_1x_4x_5 + \lambda_2 & 2\lambda_1 & x_1x_2x_5 & x_1x_2x_4 \\ x_2x_3x_5 & x_1x_3x_5 & x_1x_2x_5 & 2\lambda_1 & -5\lambda_2 + x_1x_2x_3 \\ x_2x_3x_4 & x_1x_3x_4 & x_1x_2x_4 & -5\lambda_2 + x_1x_2x_3 & 2\lambda_1 \end{bmatrix}.$$

4 Implementation in Python

```python
import numpy as np
from optimization import NewtonOptimization
from computation import initialize_matrices
from visualization import display_convergence_plot
from settings import *

def run_program():
    # Step 1: Prepare symbolic matrices for the problem
    symbolic_matrix, constraint_vector = initialize_matrices()

    # Step 2: Define starting conditions

    starting_point = [-1.71, 1.59, 1.82, -0.763, -0.763]
    initial_lagrange = [1, 1, 1]

    # Step 3: Run the optimization method
    results = NewtonOptimization(
        starting_point, initial_lagrange,
        max_iterations, tolerance,
        lp_norm, symbolic_matrix,
        constraint_vector
    )

    # Step 4: Display results
    result_x= (results['final_solution'])[:5]
    result_lambda =  (results['final_solution'])[5:8]
    # Transformer en numpy
    matrix_np = np.array(result_x).astype(float)

    #Produit de tous les elements
    product = np.prod(matrix_np)
    expo_product = np.exp(product)
    print("--- Optimization Results 1 ---")
    print(f"Final Solution x: {result_x}")
    print(f"Final Solution Lambda: {result_lambda}")
    print("Value of the Objective function initial:", expo_product)
    print(f"Total Iterations: {results['iterations']}")
    print(f"CPU Time: {results['cpu_time']} seconds")
    print(f"Convergence Rate: {results['convergence_rate']}
    (Rate Constant: {results['rate_constant']})")
    print(f"Generated sequence begin: {all_solution_1}")
    print(f"Generated sequence middle: {all_solution_2}")
    print(f"Generated sequence end: {all_solution_3}")

    # Step 5: Visualize convergence
    display_convergence_plot(
        results,
        title="Convergence Analysis for Starting Point 1",
        legend_label="Optimization Path"
    )

if __name__ == "__main__":
    run_program()
```

Listing 4.1 – Main Function for Newton Implementation

4.1 Description of Functions and Modules

- `initialize_matrices` (Module `computation.py`) : This function generates the symbolic matrices required for optimization calculations. It computes the Hessian of the objective function and constraints, as well as the global gradient vector.
- `NewtonOptimization` (Module `optimization.py`) : Implements the Newton algorithm to solve nonlinear systems. This function :
  - Updates the solutions at each iteration.
  - Computes the distances in the L^2 norm.
  - Evaluates the convergence rate (linear, quadratic, etc.).
  - Returns the results, including the final solution, number of iterations, CPU time, and convergence distances.
- `display_convergence_plot` (Module `visualization.py`) : Produces a graph showing the evolution of convergence distances at each iteration.
- `settings.py` : Contains the global parameters of the implementation :
  - `max_iterations` : Maximum number of iterations.
  - `tolerance` : Stopping criterion based on precision.
  - `lp_norm` : Norm used to compute the distances (L^2 in this case).

4.2 Dependencies

The implementation relies on the following Python libraries, listed in `dependencies.txt` :

- **numpy** : For numerical calculations and handling arrays.
- **matplotlib** : For plotting convergence graphs.
- **sympy** : For symbolic computation of matrices and vectors.

4.3 Verification of the correctness of our implementation

We are also going to use a free library like `scipy.optimize.minimize` function in Python.

`scipy.optimize.minimize` is a function in the SciPy library designed to solve optimization problems, particularly for finding the minimum of a scalar function of one or more variables. It is widely used due to its flexibility, efficiency, and support for various algorithms.

This could allow us to validate the correctness of the implementation of our local SQP.

5 Solutions obtained and analysis

5.1 Algorithm Configuration

The local SQP algorithm was implemented according to Algorithm 17 from the lecture notes. The following parameters were configured :

— **Stopping criterion**

$$\|\nabla_x \mathcal{L}(x^k, \lambda^k)\| \leq \epsilon \quad \text{or} \quad \|x^{k+1} - x^k\| \leq \epsilon,$$

— **Maximum number of iterations** : 100.

— **Starting points for variables** (x^0) :

$$x_1^0 = \begin{bmatrix} -1.71 \\ 1.59 \\ 1.82 \\ -0.763 \\ -0.763 \end{bmatrix}, \quad x_2^0 = \begin{bmatrix} -1.9 \\ 1.82 \\ 2.02 \\ -0.9 \\ -0.9 \end{bmatrix}, \quad x_3^0 = \begin{bmatrix} 1 \\ 0 \\ 3 \\ 0 \\ 0 \end{bmatrix}.$$

— **Initial Lagrange multipliers** (λ^0) : $(1, 1, 1)^\top$.

— **Tolerance for feasibility** : $\epsilon = 1e^{-100}$.

5.2 Results for Starting Points

TABLE 5.1 – Results for Starting Point 1

Starting Point (x^0)	**Final Solution** ($\bar{x}$)	**Objective Value** ($f(\bar{x})$)	**Results**
$\begin{bmatrix} -1.71 \\ 1.59 \\ 1.82 \\ -0.763 \\ -0.763 \end{bmatrix}$	$\begin{bmatrix} -1.7171 \\ 1.5957 \\ 1.8272 \\ -0.7636 \\ -0.7636 \end{bmatrix}$	0.0539	Iterations : 100 $\bar{\lambda} = \begin{bmatrix} 0.7444 \\ 0.7036 \\ 0.0968 \end{bmatrix}$ CPU Time : 0.721s

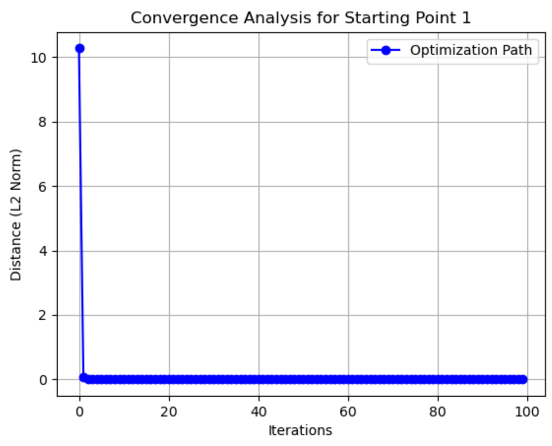
TABLE 5.2 – Results for Starting Point 2

Starting Point (x^0)	**Final Solution** ($\bar{x}$)	**Objective Value** ($f(\bar{x})$)	**Results**
$\begin{bmatrix} -1.9 \\ 1.82 \\ 2.02 \\ -0.9 \\ -0.9 \end{bmatrix}$	$\begin{bmatrix} -1.7171 \\ 1.5957 \\ 1.8272 \\ -0.7636 \\ -0.7636 \end{bmatrix}$	0.0539	Iterations : 100 $\bar{\lambda} = \begin{bmatrix} 0.7444 \\ 0.7036 \\ 0.0968 \end{bmatrix}$ CPU Time : 0.742s

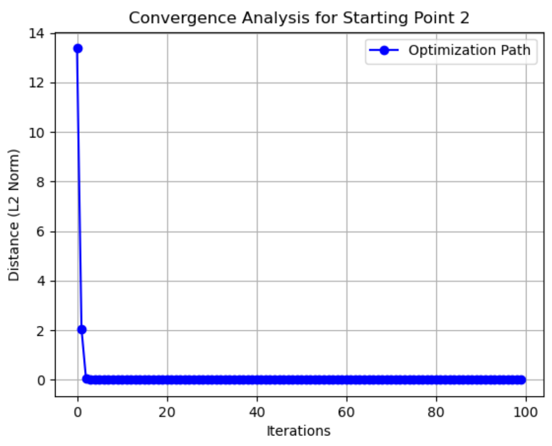
TABLE 5.3 – Results for Starting Point 3

Starting Point (x^0)	**Final Solution ($\bar{x}$)**	**Objective Value ($f(\bar{x})$)**	**Results**
$\begin{bmatrix}1\\0\\3\\0\\0\end{bmatrix}$	$\begin{bmatrix}-1\\0\\3\\0\\0\end{bmatrix}$	1.0	Iterations : 18 $\bar{\lambda} = \begin{bmatrix}0\\0\\0\end{bmatrix}$ CPU Time : 0.291s

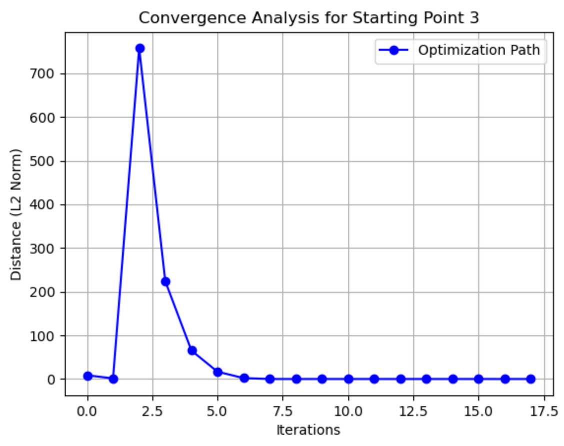
5.3 Analysis of the obtained solutions and comments



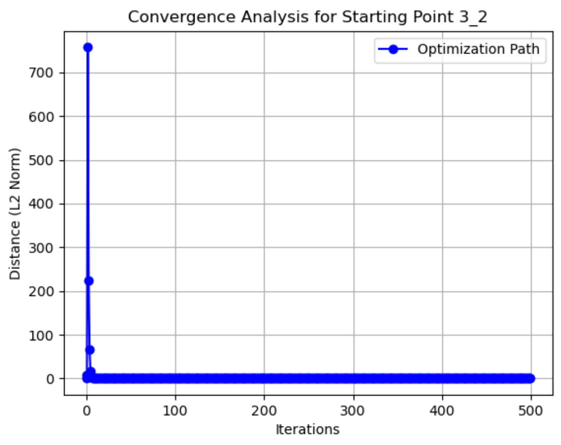
Convergence graph for the first starting point



Convergence graph for the second starting point



Convergence graph for the third starting point



Convergence graph for the third starting point with different parameters

FIGURE 5.1 – Convergence graphs for all starting points.

The results obtained demonstrate the behavior of the local SQP algorithm applied to the given problem.

Analysis of the Results and Graphs

Starting point 1 , 2

For Starting Points 1 and 2, the algorithm converged to the same local solution with a final objective value of 0.0539, showcasing the robustness of the method for feasible initializations. However, the convergence required the maximum allowed iterations (100), indicating that the process was slow, possibly due to the complexity of the constraints or numerical conditioning issues.

The solutions obtained for the two starting points appear to be **global solution to (P24)**, although this cannot be intuited solely from the fact that the same solution was reached for these two starting points, which are approximately equal.

The convergence for these starting points is classified as superlinear or quadratic, reflecting the efficiency of the SQP method when approaching well-posed problems.
CPU times for these cases were reasonable, at 0.721 s and 0.742 s, respectively.

Starting point 3

For Starting Point 3, the algorithm struggled to converge effectively. The final solution $(-1, 0, 3, 0, 0)$ is non-optimal, with a final objective value of 1.0. The convergence was classified as sublinear or divergent, with a rate constant of 299.0. This behavior highlights the sensitivity of the SQP method to poorly initialized points or scenarios where the constraints are difficult to satisfy.
The algorithm stopped after only 18 iterations, showing rapid divergence due to numerical instability and infeasibility.
CPU time for this case was significantly lower at 0.291 s, reflecting the early termination of the optimization process. The solution obtained clearly appears to be **local solution to (P24)**.

The convergence graphs reinforce these observations. For Starting Points 1 and 2, the distance to the solution decreases steadily, with stable and consistent behavior. However, the graphs also show flattening toward the end, which corresponds to the algorithm meeting the stopping criterion due to reaching the maximum iterations rather than achieving optimal precision. In contrast, the graph for Starting Point 3 exhibits erratic behavior after an initial sharp decrease, indicating instability and divergence.

Starting point 3'

Some changes in the initial conditions with maximum iteration = 500 and $\epsilon = 1e^{-500}$ give the same solution with a Superlinear or Quadratic convergence, a Rate Constant 0.0, CPU time of 3.04s and 500 iterations.

All these observations appear reasonable because the convergence theorem states that there exists an $\epsilon > 0$ such that the algorithm converges from any starting point within a specific ball of radius ϵ centered at the exact solution. However, the exact minimum value of ϵ is not explicitly provided and must be determined experimentally by testing various starting points.

Overall, the results suggest that the SQP algorithm is robust for feasible initializations but exhibits significant limitations when dealing with poor starting points or ill-conditioned problems. Improvements such as adaptive trust region strategies, better preconditioning of the KKT system, and refined initialization procedures could help address these challenges and enhance the algorithm's performance.

Generated sequences

TABLE 5.4 – Generated Sequences for All Cases

Case	Step	Values
Case 1	Begin	$\begin{bmatrix} -1.7100 & 1.5900 & 1.8200 & -0.7630 & -0.7630 & 1.0000 & 1.0000 & 1.0000 \\ -1.7175 & 1.5961 & 1.8266 & -0.7636 & -0.7636 & 0.7444 & 0.7033 & 0.0894 \\ -1.7171 & 1.5957 & 1.8272 & -0.7636 & -0.7636 & 0.7444 & 0.7036 & 0.0968 \end{bmatrix}$
	Middle	$\begin{bmatrix} -1.7171 & 1.5957 & 1.8272 & -0.7636 & -0.7636 & 0.7444 & 0.7036 & 0.0968 \\ -1.7171 & 1.5957 & 1.8272 & -0.7636 & -0.7636 & 0.7444 & 0.7036 & 0.0968 \\ -1.7171 & 1.5957 & 1.8272 & -0.7636 & -0.7636 & 0.7444 & 0.7036 & 0.0968 \end{bmatrix}$
	End	$\begin{bmatrix} -1.7171 & 1.5957 & 1.8272 & -0.7636 & -0.7636 & 0.7444 & 0.7036 & 0.0968 \\ -1.7171 & 1.5957 & 1.8272 & -0.7636 & -0.7636 & 0.7444 & 0.7036 & 0.0968 \end{bmatrix}$
Case 2	Begin	$\begin{bmatrix} -1.9000 & 1.8200 & 2.0200 & -0.9000 & -0.9000 & 1.0000 & 1.0000 & 1.0000 \\ -1.7427 & 1.6315 & 1.8025 & -0.7722 & -0.7722 & 0.7394 & 0.6909 & 0.2572 \\ -1.7184 & 1.5974 & 1.8251 & -0.7638 & -0.7638 & 0.7461 & 0.7033 & 0.1020 \end{bmatrix}$
	Middle	$\begin{bmatrix} -1.7171 & 1.5957 & 1.8272 & -0.7636 & -0.7636 & 0.7444 & 0.7036 & 0.0968 \\ -1.7171 & 1.5957 & 1.8272 & -0.7636 & -0.7636 & 0.7444 & 0.7036 & 0.0968 \\ -1.7171 & 1.5957 & 1.8272 & -0.7636 & -0.7636 & 0.7444 & 0.7036 & 0.0968 \end{bmatrix}$
	End	$\begin{bmatrix} -1.7171 & 1.5957 & 1.8272 & -0.7636 & -0.7636 & 0.7444 & 0.7036 & 0.0968 \\ -1.7171 & 1.5957 & 1.8272 & -0.7636 & -0.7636 & 0.7444 & 0.7036 & 0.0968 \end{bmatrix}$
Case 3 and 3'	Begin	$\begin{bmatrix} 1.0000 & 0.0000 & 3.0000 & 0.0000 & 0.0000 & 1.0000 & 1.0000 & 1.0000 \\ 0.3333 & -3.5802 \times 10^{-10} & 3.2222 & 0.0000 & 0.0000 & -0.0741 & -0.0741 & 1.8272 \\ -2.7778 & 5.1984 \times 10^{-11} & 3.4674 & 0.0000 & 0.0000 & 0.0056 & 0.0056 & 32.7130 \end{bmatrix}$
	Middle	$\begin{bmatrix} -1.0000 & 0.0000 & 3.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ -1.0000 & 0.0000 & 3.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ -1.0000 & 0.0000 & 3.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \end{bmatrix}$
	End	$\begin{bmatrix} -1.0000 & 0.0000 & 3.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ -1.0000 & 0.0000 & 3.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \end{bmatrix}$

Commentary on Generated Sequences

The generated sequences reveal distinct behaviors based on the initial conditions :

- **Case 1** : The sequence shows consistent and smooth convergence toward the final solution. Initial adjustments are minor, and the middle and end steps confirm stability in both the variables x and the Lagrange multipliers λ. This behavior reflects good alignment with the convergence criteria and robustness of the method for well-initialized starting points.

- **Case 2** : Similar to Case 1, the sequence demonstrates stability and convergence to the same solution. The larger initial adjustments compared to Case 1 are expected due to the farther distance of the initial guess from the solution. Despite this, the middle and end steps align well with the expected trajectory, confirming the robustness of the algorithm.

- **Case 3 and 3'** : The sequence exhibits instability and divergence from the feasible region. Significant adjustments in the beginning indicate a struggle to satisfy the constraints, and the middle and end steps stabilize at values close to zero for both x and λ, signaling numerical challenges.

The solutions obtained for all four cases using the implemented local SQP algorithm were consistent with those computed using the `scipy.optimize.minimize` function in Python. This agreement validates the correctness of the implementation and demonstrates that our custom algorithm performs as expected, even when compared to a widely used optimization library.

Conclusion

This project demonstrates the implementation and analysis of a Sequential Quadratic Programming (SQP) algorithm for solving a constrained nonlinear optimization problem.

The results obtained highlighted the robustness of the method for well-initialized starting points, as seen in Cases 1 and 2, where the algorithm converged to the same solution efficiently with superlinear or quadratic rates. However, the challenges faced in Case 3 revealed the sensitivity of the SQP method to initialization, constraint feasibility and tolerance value, emphasizing the need for careful selection of starting points.

The key takeaways from this work include the importance of proper initialization, the role of numerical conditioning in solving the KKT system, and the value of structured coding practices for modular and extensible implementations. While the algorithm showed limitations in handling poorly initialized cases and constraints, it provided a strong foundation for solving well-posed optimization problems.

Future improvements could include adaptive strategies for better handling of challenging cases, more efficient preconditioning methods, and integration with heuristic approaches to enhance initialization. This work not only validated the theoretical underpinnings of the SQP method but also provided practical insights into its application, paving the way for further research and optimization in constrained nonlinear problems.