ENSTA | INSTITUT POLYTECHNIQUE DE PARIS

Report - OPT2 - TP1

Implementation of the Newton method

Disclaimer: in the python file the functions calls are in comments write now

Tom Cuel

February 21, 2025

Contents

Introduction

This project explores the application of Newton's method for finding the static equilibrium of a chain composed of rigid bars in two-dimensional space. The problem is formulated as an optimization problem, with the constraints initially expressed as equalities. This formulation allows us to apply Newton's method to find the root of the non-linear system of equations, which in turn provides the desired static equilibrium. In the second part we add inequality constraints and the scipy.optimize tool to see a more complex problem and compare the results between the tool and the by-hand Newton method.

1 Problem to solve

1.1 Modelisation of the problem

We consider a chain with $N+1$ rigid bars, whose endpoints are (x_{i-1}, y_{i-1}) and (x_i, y_i) for $i \in [1, \dots, N+1]$. The decision variables are (x_i, y_i) for $i \in [1, \dots, N]$, with fixed endpoints $(0,0)$ and (a,b) for the whole chain. The bars are all supposed to be of the same length $L = \ell_i(x,y) = \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}$ for a bar i of extremities at location $i-1$ and i. N, L, a, b can be modified in the code. The problem can be modelized as the following optimization problem:

$$\min_{x \in \mathbb{R}^N, y \in \mathbb{R}^N} E(x,y) = \sum_{i=1}^{N+1} \ell_i(x,y) \frac{y_{i-1} + y_i}{2} \quad \text{subject to: } c_i(x,y) = l_i(x,y)^2 - L^2 = 0, \forall i \in [1, \dots, N+1] \tag{1}$$

that can be rewritten as, with the current conditions:

$$\min_{x \in \mathbb{R}^N, y \in \mathbb{R}^N} \sum_{i=1}^{N} y_i \quad \text{subject to: } c_i(x,y) = 0, \forall i \in [1, \dots, N+1] \tag{2}$$

The problem is not convex **(Q1)** because the constraint $\ell_i(x,y) = \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} = L$ defines a non-convex feasible set, meaning that the convex combination of two feasible solutions may not remain feasible (even though the constraint itself is not affine, it doesn't mean that the feasible set is not convex).

1.2 Optimality conditions

We introduce the Lagrangian of the problem: $L(x, y, \lambda) = \sum_{i=1}^{N} y_i + \sum_{i=1}^{N+1} \lambda_i c_i(x, y) = e^T z + \lambda^T c(z)$
with $z = (x, y) = (x_1, \ldots, x_N, y_1, \ldots, y_N)$ and $e = (0, \ldots, 0, 1, \ldots, 1)$
We then write the optimality conditions:

$$\begin{cases} \nabla_z L(z, \lambda) = 0 \\ \nabla_\lambda L(z, \lambda) = 0 \end{cases} \iff \begin{cases} e + \nabla_z^T c(z)\lambda = 0 \quad \textit{(stationarity condition)} \\ c(z) = 0 \quad \textit{(equality constraint)} \end{cases}$$

An optimality condition is a necessary condition that the variable z must satisfy to be considered a solution of an optimization problem. These conditions are useful for designing and analyzing optimization algorithms such as Newton's method, and provide a set of candidates for the minima of the objective function. The Karush-Kuhn-Tucker (KKT) conditions are necessary conditions for optimality, but only when the constraint qualification is satisfied. However, it is important to note that in non-convex optimization problems, there may be multiple local optima, and the KKT conditions may not be sufficient to uniquely identify the global optimum. Therefore, while the KKT conditions are a powerful tool for optimizing convex problems and provide a set of necessary conditions that must be satisfied at an optimal solution, but they have limitations in the case of non-convex problems. So, we may not find a global minimizer of the problem **(Q2)**.
In non-convex optimization problems, such as the one studied in this project, it is important to be aware that a solution may not exist for certain values of the problem parameters (a, b, N, L). Therefore, a solution to problem exists only when the end point (a, b) is located within the range of the chain, as expressed mathematically by the inequality: $\sqrt{(a^2 + b^2)} \leq (N + 1)L$ **(Q3)**. This ensures that the total length of the N+1 rigid bars is sufficient to connect the fixed endpoints (0,0) and (a,b). If this condition is not met, no feasible solution exists because the chain cannot physically reach the required position.

2 Newton's method

2.1 Description of the method

To start the iteration, we generate the initial guess x, let it sufficiently close to the actual solution. This may help to accelerate the convergence rate of the Newton method. However, it is important to note that the performance of the Newton method is highly sensitive to the choice of initial guess, for example choosing $(x, y) = (0, 0)$ doesn't work since the part in the algorithm where we need to invert a matrix is not possible. To ensure that the Newton step is a descent direction and that we found a local minimizer, we use the function `check_stationarity` to verify the positive definiteness of the Hessian matrix of the Lagrangian. Here is the algorithm of the Newton method:

Newton's method

Given N, L, a, b, ensure that $\sqrt{a^2 + b^2} \leq (N + 1)L$ *(we choose them that way in pratice)*
$z_0, \lambda_0 \leftarrow$ initial guess
$k \leftarrow 0$, error $\leftarrow \infty$
while $(k <$ max_iter) and (error $>$ tol) **do**
  Let $\xi_k = (z_k, \lambda_k)$, $\quad F(\xi_k) = (e + \nabla_z^T c(z_k)\lambda_k, c(z_k))$, $\quad$ we get : $d = (\Delta z, \Delta\lambda) = -(\nabla_\xi F(\xi_k))^{-1} F(\xi_k)$
  $\alpha_k \leftarrow 1$ *that's where the backtracking line is*
  $z_{k+1} \leftarrow z_k + \alpha_k \Delta z \quad \lambda_{k+1} \leftarrow \lambda_k + \alpha_k \Delta\lambda$
  $k \leftarrow k + 1 \quad$ error $\leftarrow \left\|(e + \nabla_z^T c(z_k)\lambda_k, c(z_k))\right\|$
end while

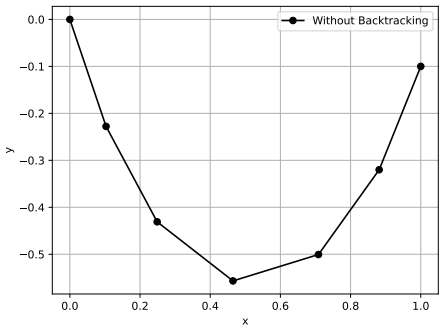
2.2 About the backtracking

The interest in the Newton method is that it's converging quadratically if the initial guess is close enough to the solution. Still, the convergence is not guaranteed. It is worth noting that the pure Newton method uses a fixed step size of $\alpha = 1$, which can cause the algorithm to overshoot the minimum and potentially diverge, leading to incorrect solutions. To avoid this, we use a backtracking line search to ensure that the step is appropriately chosen and that the algorithm moves in a direction that leads to convergence and the objective function decrease. Among the different method, I coded the damped Newton method in the python code, but here is the uniform gridding one. Here is the little algorithm that need to be added in the Newton method to use this backtracking method:

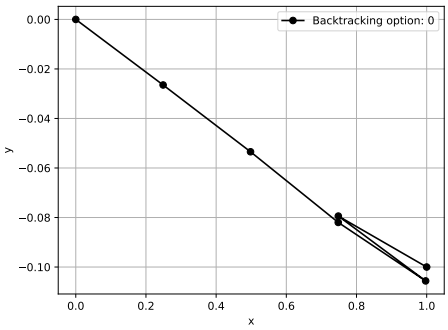
Backtracking Line Search

$\alpha_{\text{best}} \leftarrow 1, \quad$ min_res $\leftarrow \infty$
$\alpha_{\text{candidate}} \leftarrow [0, \ldots, 1]$ with $\lceil L/\epsilon \rceil^N$ points for an uniform grid
for all $\alpha \in \alpha_{\text{candidate}}$ **do**
  if $\left\|(e + \nabla_z^T c(z_k + \alpha\Delta z)(\lambda_k + \alpha\Delta\lambda), c(z_k) + \alpha\Delta z)\right\| <$ min_res **then**
    $\alpha_{\text{best}} \leftarrow \alpha, \quad$ min_res $\leftarrow \left\|(e + \nabla_z^T c(z_k + \alpha\Delta z)(\lambda_k + \alpha\Delta\lambda), c(z_k) + \alpha\Delta z)\right\|$
  end if
end for

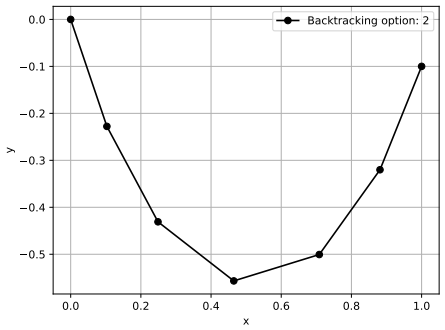
2.3 First results



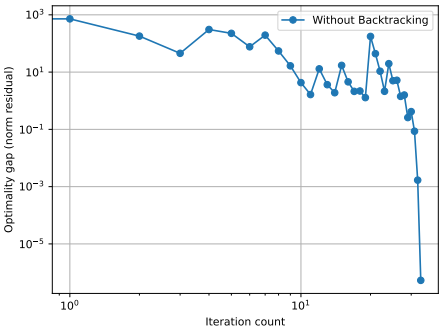
(a) converging without backtracking

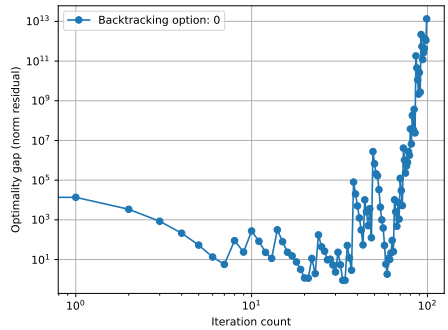


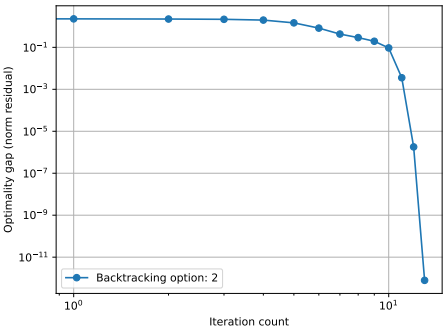
(b) diverging without backtracking



(c) converging with backtracking







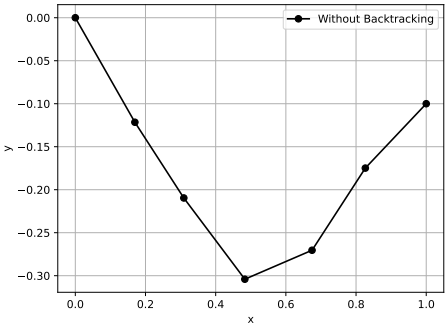
The result does confirm the trend that the Newton method converges faster with the backtracking line search, that the Newton's method can sometimes diverge without it. Overall, our results demonstrate the effectiveness of Newton's method with backtracking line search in solving the non-convex optimization problem studied in this project.

2.4 Influence of the different parameters

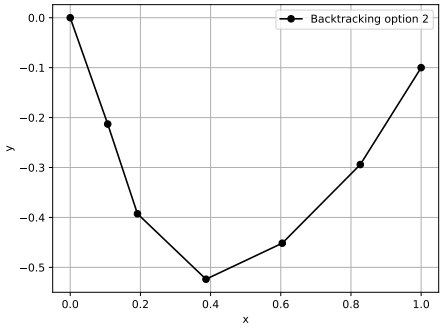
Let's now investigate the influence of the different parameters on the convergence. I already talked about the initial guess that is critical and also that needs to be feasible so that the matrix inversion is possible at some point and the code doesn't crash.

2.4.1 λ

To test the λ influence, I tried to make many different lambda values and see what it changes. Those values have two components, the first one is a weight coefficient and the seccond is the $N+1$ points vectors with all its values equal to 1. At the end of the day, I tried a wide range of λ values. Here are the results I obtained for one test in particular:



(a) average chain without backtracking



(b) average chain with backtracking

For different λ values, we can clearly see that the backtracking methods provide results that are way closer to the solution than the ones without it. But on average *(since those are average chains, made out the ponderated sum of the nb_λ chains, I didn't wanted to superpose 10 graph for them to be more readable)*, the results aren't that far from the optimal solution.

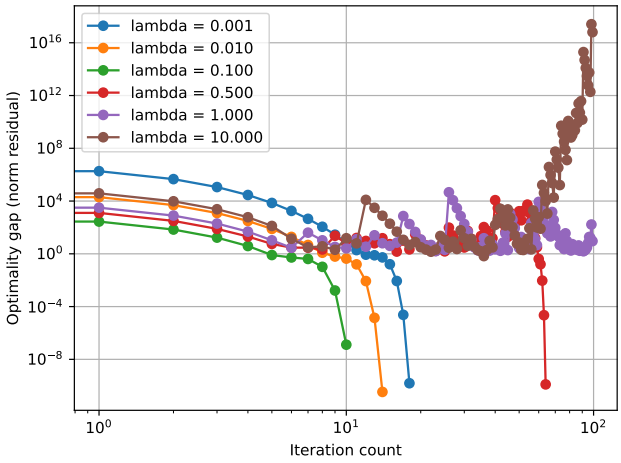
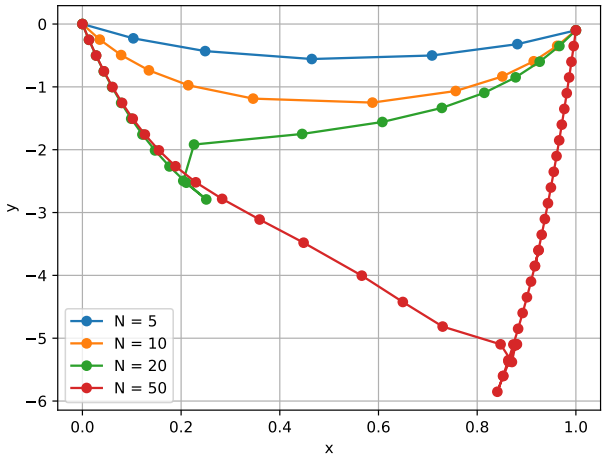


Figure 4: convergence with different λ

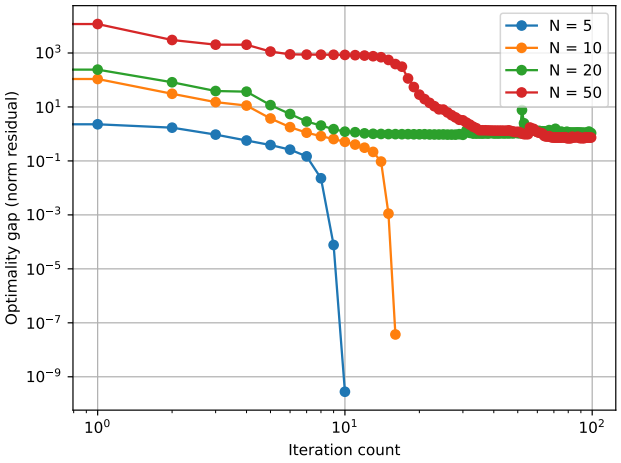
The convergence graph was made on the Newton's method without backtracking. This graph is one of the many I got. The results indicate that the convergence speed of the algorithm can be affected by the value of λ with faster convergence observed for smaller values of λ *($\lambda \leq 0.1$)*. However, we observed that when λ is big, the algorithm is unable to move in a descent direction and therefore cannot converge. These results suggest that the choice of the initial guess for λ can significantly affect the performance of the optimization algorithm, and that a good starting guess for λ is critical to ensure the convergence of the algorithm. Further research is needed to identify strategies for selecting an appropriate initial guess for λ that can improve the convergence properties and robustness of Newton's method with backtracking line search in solving non-convex optimization problems.

2.4.2 N

For a fixed chain bar lenght $L = 0.25$ and a fixed λ value, I tried different values of N to see how the convergence is affected by the number of bars.



(a) average chain with backtracking

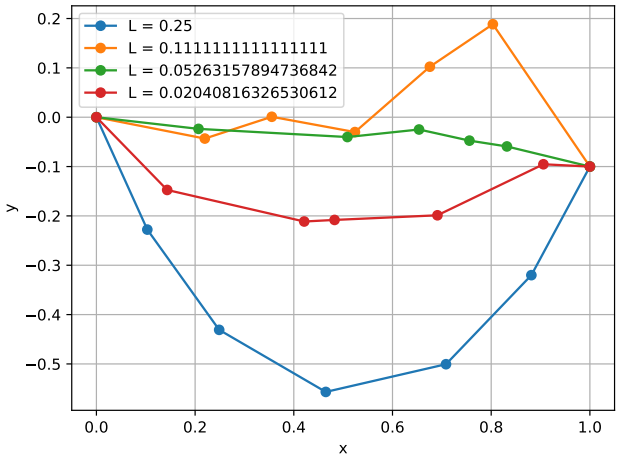


(b) convergence with different N

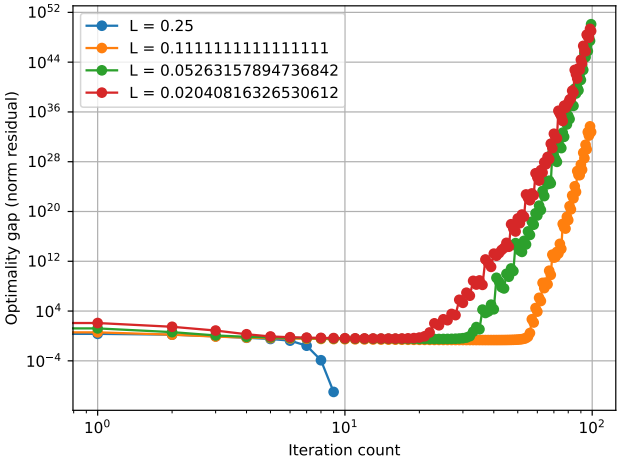
The results are clearly quite telling. Augmenting the number of chains without changing the other parameters can lead to false solutions, because we need to ensure that the condition $\sqrt{a^2 + b^2} \leq (N + 1)L$ is respected and that's not the case here. And all of that even though the algorithm converges. But overall adding more chains leads to higher computation times as expected.

2.4.3 L

For a fixed number of bars $N = 5$ and a fixed λ value, I tried different values of L to see how the convergence is affected by the bar length.



(a) average chain with backtracking



(b) convergence with different L

Same remarks for the results as for the N variation, we need to respecte the condition $\sqrt{a^2+b^2} \leq (N+1)L$ for the algorithm to converge and to work properly, changing one of the parameters without the others accordingly can lead to errors in the results.

3 Inequality constraints and `scipy.optimize`

3.1 New problem

Despite my option being this one, I did code the convex relaxation method in the python script just to see how it works, results won't be shown there but they can be computed. Back to the new problem for option 2, it's the same as before, but we add inequality constraints to the problem. The problem can be formulated as:

$$\min_{x\in\mathbb{R}^N, y\in\mathbb{R}^N} \sum_{i=1}^{N} y_i \quad \text{subject to: } c_i(x,y) = 0 \text{ and } g_i(z) = 0.2x_i + y_i + 0.1 \geqq 0, \quad \forall i \in [1,\dots,N+1] \tag{3}$$

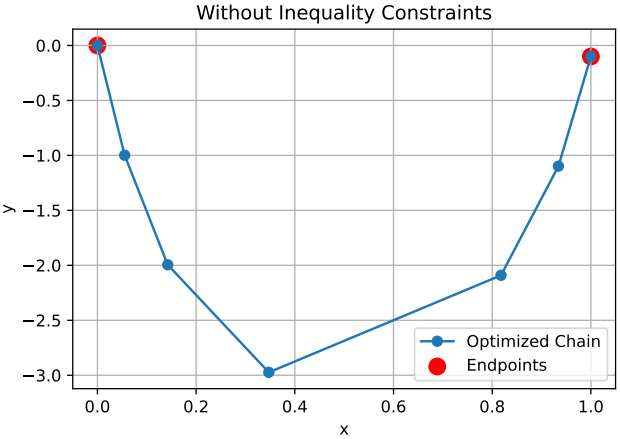
We can use the Sequential Least Squares Quadratic Programming (SLSQP) solver available in the `scipy.optimize.minimize` function. The SLSQP method is part of the broader family of Sequential Quadratic Programming (SQP) algorithms, which aim to solve nonlinear optimization problems, typically subject to equality and inequality constraints.

3.2 Comparison with the Newton method

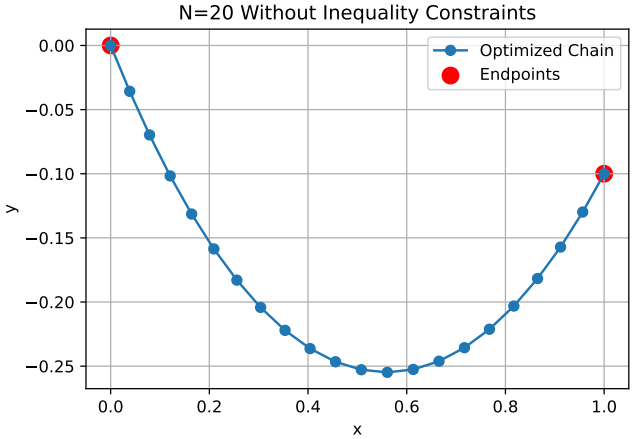
We can directly make a comparaison between the two method when we don't consider the inequality constraints. Here is what the terminal gives us:

```
Solving optimization without inequality constraints...
Optimization terminated successfully    (Exit mode 0)
            Current function value: -9.155454716584043
            Iterations: 20
            Function evaluations: 228
            Gradient evaluations: 20
Solution without inequality constraints: [ 0.05522486  0.14239791  0.34731719  0.81778884  0.9342592  -0.99847394
 -1.99466713 -2.97344601 -2.0910309  -1.09783673]
```

We have a number of iterations that is similar to what we got earlier, although there are possibly slightly more iterations in this case. But the results are quite similar to the newton's method one, obtained a bit faster though.



(a) scipy result for the base example, $N = 5$



(b) result for $N = 20$

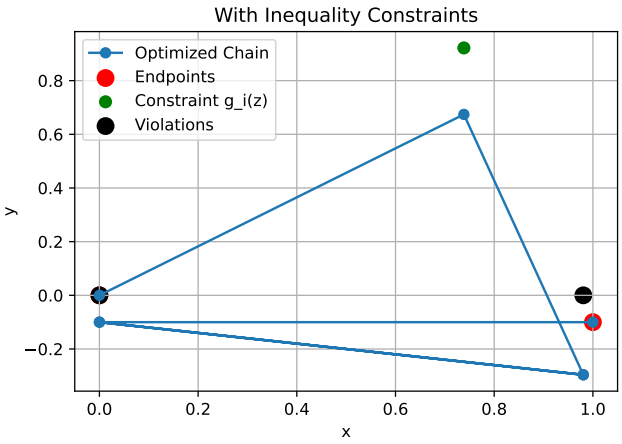
The main difference lies in the handling of constraints, which doesn't apply here. The SLSQP solver is based on the SQP algorithm, which iteratively solves a sequence of quadratic subproblems to approximate the nonlinear optimization problem. While the Newton method requires the Hessian matrix for each iteration, SQP approximates the problem by solving quadratic programming subproblems, which allows it to handle both equality and inequality constraints more flexibly. Since there are no constraints in this case, the performance of SLSQP and Newton's method are comparable. Both methods converge relatively quickly, given that they utilize second-order information *(either the Hessian matrix in the case of Newton's method or approximations of it in the case of SQP)*.

3.3 Inequality constraints influence

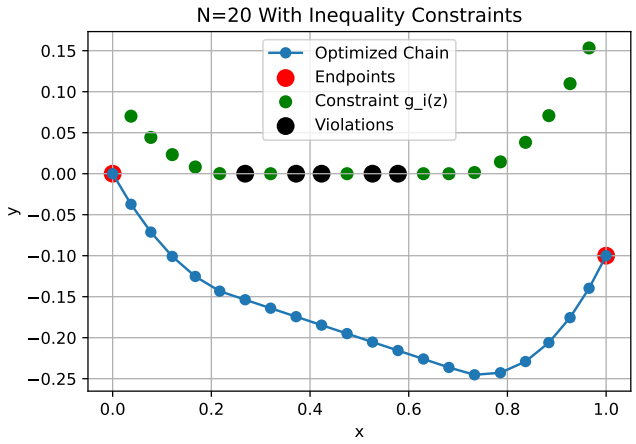
I tried to see how the inequality constraints influence the results, and here is what I observed:

```
Solving optimization with inequality constraints...
Optimization terminated successfully    (Exit mode 0)
            Current function value: -0.11807119818756562
            Iterations: 6
            Function evaluations: 67
            Gradient evaluations: 6
Constraint violations detected: [-1.58206781e-15 -2.08166817e-15]
Solution with inequality constraints: [ 7.38584395e-01  9.80580676e-01 -7.17409422e-12  9.80580676e-01
 -2.05938936e-12  6.74161072e-01 -2.96116135e-01 -1.00000000e-01
 -2.96116135e-01 -1.00000000e-01]
```

Interesting enough, the convergence does happens sooner with the inequality constraints, but the computation time is a bit longer. With inequality constraints, the solver converges in fewer iterations (6 vs. 20) likely because the constraints restrict the feasible region, guiding the optimization more directly to a solution. Without constraints, the solver explores a larger space, requiring more iterations to find the optimal point.



(a) scipy result with constraints



(b) result for $N = 20$

It's a bit unclear with $N = 5$ since it doesn't look like anything I've encountered before, the inequality constraints seems to mess things up. For $N = 5$, the results appear somewhat unexpected, suggesting that the constraints may be influencing the optimization in a nontrivial way. It is possible that the constraints introduce a local minimum or alter the gradient landscape, making convergence behavior less intuitive. On the other hand, for $N = 20$, the results are more consistent, with the constraints effectively shaping the solution in a predictable manner. This suggests that the impact of inequality constraints becomes clearer with a larger problem size. Therefore, I will focus solely on the $N = 20$ case to draw more reliable conclusions. While most of the constraints appear to be satisfied, some violations are explicitly marked in the plot. These violations indicate that at certain points along the optimized chain, the constraint function $g_i(z)$ dips below zero, meaning the constraints were not fully respected. This could be due to several reasons:

1. Tolerance issues: The solver might consider small violations acceptable within numerical precision limits.
2. Regularization trade-offs: The optimization process may have prioritized minimizing the objective function over strictly enforcing constraints.
3. Solver limitations: The SLSQP method may struggle with very strict or complex constraints, especially if the feasible region is small or if the constraints are nearly active.

We can also note that the curve doesn't have the same shap as before, where there is constraint violation, making the shape goes up a bit more. The deformation of the curve near constraint violations suggests that the solver did not fully enforce the inequality constraints, allowing the solution to drift into an infeasible region. This likely results from numerical tolerance issues, trade-offs between minimizing the objective function and satisfying constraints, or limitations of the SLSQP method in handling tight constraints. The unexpected shape change indicates that the solver may have slightly overshot feasible boundaries, altering the optimized chain's trajectory.

The Sequential Quadratic Programming (SQP) method, implemented via `scipy.optimize.minimize` with the SLSQP solver, proves to be an effective approach for constrained optimization. It efficiently balances objective minimization with constraint enforcement, often converging in fewer iterations when inequality constraints are present. However, as observed in the results, the method is not always perfect in strictly enforcing constraints, sometimes allowing small violations due to numerical tolerances or solver limitations. Despite this, SQP remains a powerful and flexible optimization technique, particularly suited for problems where constraints play a crucial role in shaping the solution. Fine-tuning solver parameters or exploring alternative optimization methods may be necessary in cases where strict feasibility is critical.