Rapport OPT202 Project 1 : An articulated chain

Alexandre LAPRERIE, Antoine GERMAIN

14 February 2024

1 Introduction

This study investigates the use of Newton's method to determine the static equilibrium of a chain made up of rigid bars within a two-dimensional framework. The issue is approached as an optimization challenge, where the constraints are first defined as equalities. Such a setup enables the application of Newton's method to identify the solution to the non-linear equations system, thereby achieving the sought-after static equilibrium.
With reference to Figure 1, we consider a chain formed by $N+1$ equal rigid bars, and whose extremities are (x_i-1, y_i-1),(x_i, y_i) for bar $i \in [1, ..., N+1]$. We take as decision variables the position of the extremities of the bar (x_i, y_i) for $i \in [1, ..., N+1]$, and we fix the two ends at $(0, 0)$ and (a, b), respectively :

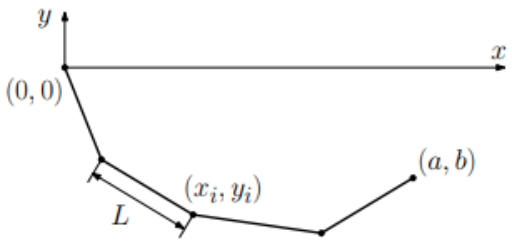


Figure 1: The articulated chain.

We keep N, a, b as free parameters to vary case by case, as well as the length of the bars (supposed the same for all of them), L. We have now a problem in 2N variables, the (x_i, y_i) for $i \in [1, ..., N]$, since the $N+1$ is constrained to be (a, b).

2 Part I: Set up

2.1 Proprieties of the problem

The problem is modelled as an optimization problem as below:

$$\min_{x \in \mathbb{R}^n,\ y \in \mathbb{R}^n} E(x, y) \quad \text{s.t. } c_i(x, y) = 0,\ i \in [1, ..., N+1], \tag{P_0}$$

where $E(x, y) = \sum_{i=1}^{N+1} l_i(x, y)\frac{y_{i-1}+y_i}{2}$, $l_i(x, y)$ is the length between (x_{i-1}, y_{i-1}) and (x_i, y_i), $c_i(x, y) = l_i(x, y)^2 - L^2 = (x_i - x_{i-1})^2 + (y_i = y_{i-1})^2 - L^2$. So ($P_0$) is equivalent as

$$\min_{x \in \mathbb{R}^n,\ y \in \mathbb{R}^n} \sum_{i=1}^{N} y_i \quad \text{s.t. } c_i(x, y) = 0,\ i \in [1, ..., N+1]. \tag{P}$$

Therefore, the optimization problem is not convex, as the constraint equality function c is not affine but quadratic considering x and y.

The problem not being convex, there is no global optimizer a priori.

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). For instance, if the end point (a, b) is located outside of the range of the chain, it will be impossible to reach and thus no solution

exists. For example, with $N = 10$ and $L = 0.05$, the algorithm produces an incorrect result, as illustrated in Figure 2, where several bars are clearly violating the $L = 0.05$ rule.

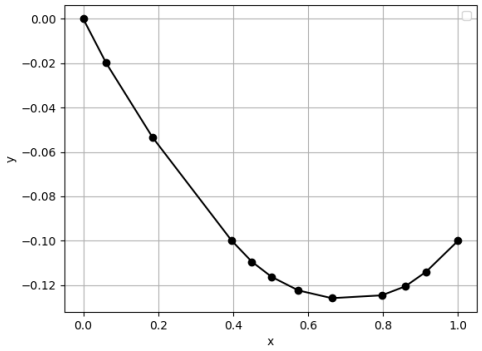


Figure 2: A problem without proper solution.

Therefore, a solution to problem (P) exists only when the end point (a, b) is located within the range of the chain, as expressed mathematically by the inequality:

$$\sqrt{a^2 + b^2} \leq NL. \tag{1}$$

This condition must be satisfied in order for the optimization problem to have a feasible solution.

2.2 Optimality conditions

To begin, we express the problem in terms of matrices and vectors as follows:

$$z = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \\ \hline y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}, \; c(z) = \begin{bmatrix} c_1(x,y) \\ c_2(x,y) \\ \vdots \\ c_{N+1}(x,y) \end{bmatrix}, \; e = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ \hline 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix},$$

so that the Langrangian can be written as: $\mathcal{L} = e^T z + \lambda c(z)$.
By Theorem, the optimality conditions for (P) is:

$$\begin{cases} \nabla_z \mathcal{L}(z,\lambda) & = 0 \\ \nabla_\lambda \mathcal{L}(z,\lambda) & = c(z) = 0 \end{cases} \iff \begin{cases} e + \nabla_z^T c(z)\lambda & = 0 \\ \nabla_\lambda \mathcal{L}(z,\lambda) & = c(z) = 0 \end{cases} \tag{2}$$

where

$$\nabla_z^T c(z) = \left[\begin{array}{ccc} \nabla_{x_1} c_1(x,y) & \dots & \nabla_{x_1} c_{N+1}(x,y) \\ \nabla_{x_2} c_1(x,y) & \dots & \nabla_{x_2} c_{N+1}(x,y) \\ \vdots & \vdots & \vdots \\ \nabla_{x_N} c_1(x,y) & \dots & \nabla_{x_N} c_{N+1}(x,y) \\ \hline \nabla_{y_1} c_1(x,y) & \dots & \nabla_{y_1} c_{N+1}(x,y) \\ \nabla_{y_2} c_1(x,y) & \dots & \nabla_{y_2} c_{N+1}(x,y) \\ \vdots & \vdots & \vdots \\ \nabla_{y_N} c_1(x,y) & \dots & \nabla_{y_N} c_{N+1}(x,y). \end{array}\right] \in \mathbf{R}^{2N \times N+1}$$

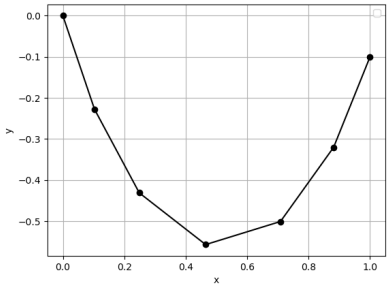
3 Part II: Newton's method

3.1 Without backtracking

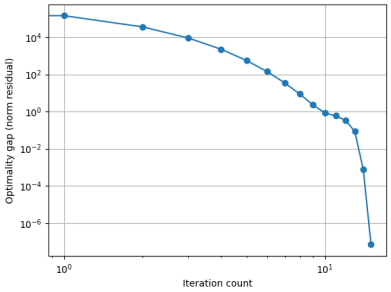
We begin by implementing Newton's method to solve problem (P) using the optimality conditions expressed in (2). To start the iteration, we generate the initial x close to the actual solution.

To ensure that the step is in a descent direction, we use the function $\texttt{check}_\texttt{s}\texttt{tationarity}(\texttt{z}, \texttt{lbd}, \texttt{gap})$ to verify the positive definiteness of the Hessian matrix $\nabla^2 f(x)$.

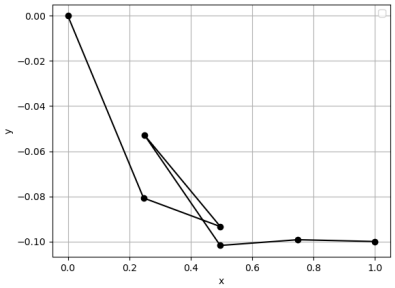
Here are a few results for $N = 5$ and $L = 0.25$ for 2 random values of λ :



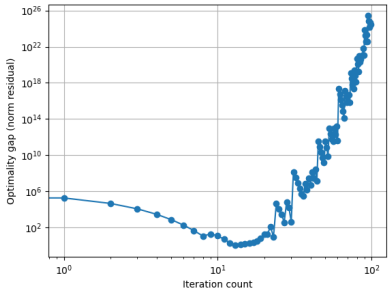
(a) Plot of solution without constraints without backtracking for N=5



(b) Plot of convergence without constraints without backtracking for N=5



(a) Plot of solution without constraints without backtracking for N=5



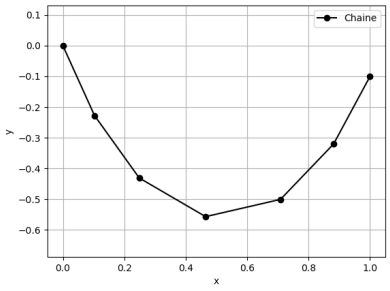
(b) Plot of convergence without constraints without backtracking for N=5

The two first solution correspond to what we could expect as an optimal solution. The third iteration does not converge to any solution.

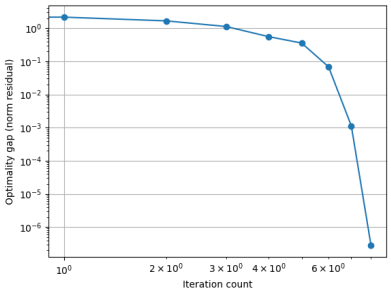
3.2 With backtracking

In the pure Newton method, convergence is not guaranteed and the algorithm can diverge or oscillate between points without approaching a solution. To address this issue, we introduce a backtracking line search to determine the appropriate step size t in each iteration, using the update rule $x_{k+1} = x_k - t(\nabla^2 f(x))^{-1}\nabla f(x)$.

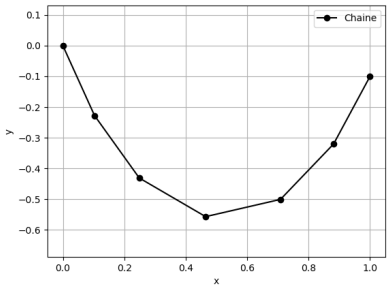
Here are a few results for $N = 5$ and $L = 0.25$ for 2 random values of λ :



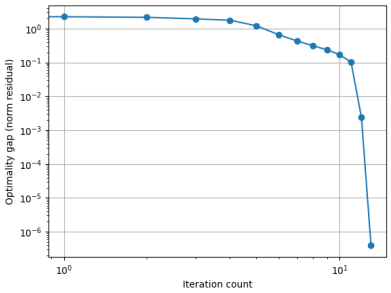
(a) Plot of solution without constraints with backtracking for N=5



(b) Plot of convergence without constraints with backtracking for N=5



(a) Plot of solution without constraints with backtracking for N=5

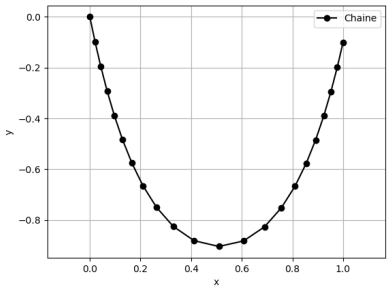


(b) Plot of convergence without constraints with backtracking for N=5

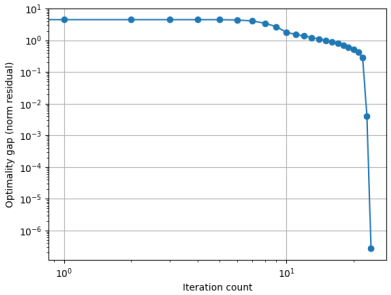
We notice that the algorithm with backtracking always converges, for any λ. Moreover, the convergence is way faster than without.

3.3 Other results with backtracking

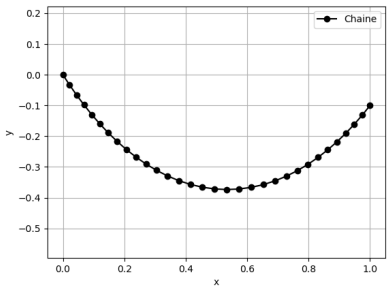
Here are different results using several values for N, L, (a, b) and random values for λ.



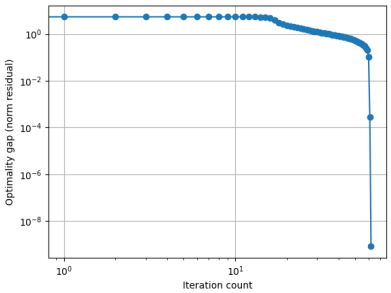
(a) Plot of solution without constraints with backtracking for N=20 and L=0.1.



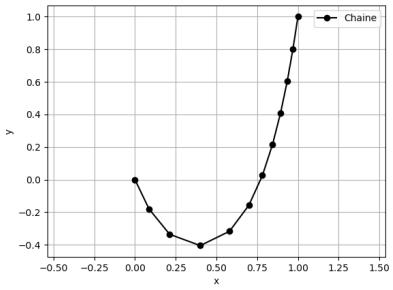
(b) Plot of convergence without constraints with backtracking for N=20 and L=0.1.



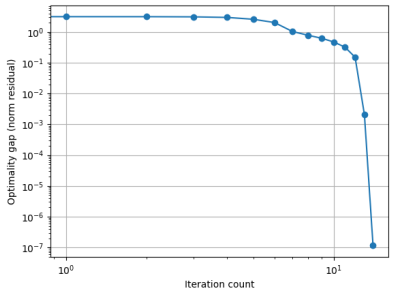
(a) Plot of solution without constraints with backtracking for N=30 and L=0.04.



(b) Plot of convergence without constraints with backtracking for N=30 and L=0.04.



(a) Plot of solution without constraints with backtracking for N=10, L=0.2 and $(a, b) = (1, 1)$.



(b) Plot of convergence without constraints with backtracking for N=10, L=0.2, and $(a, b) = (1, 1)$.

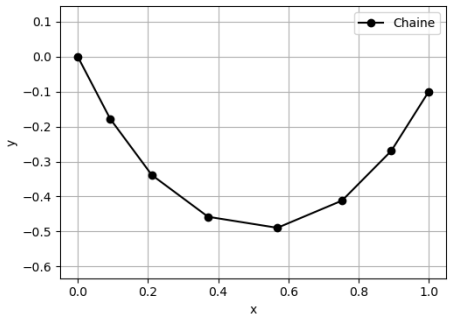
Convergence always occurs with backtracking following the same pattern, and its speed mainly depends on the value of λ and not on the values of N, L, a or b.

4 Part III: Inequality constraint

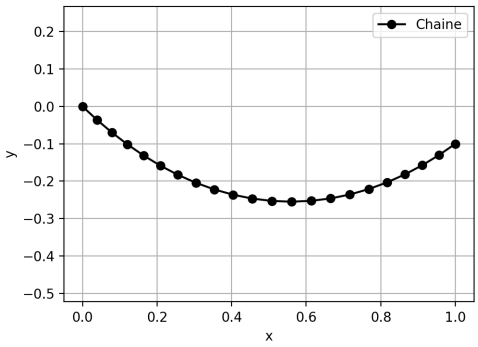
We reconsider the problem (P) and now add the constraints :

$$g_i(z) = 0.2x_i + y_i + 0.1 \geq 0$$

To begin with, we will first consider the problem without the g_i constraint and solve problem (P) using the "SLSQP" solver in scipy's optimize.minimize function. We therefore implement our code and obtain these results by taking N=6 and N=20. We obtain the following curves:



(a) Plot of solution without constraints for N=6



(b) Plot of solution without constraints for N=20

Intuitively, it's easy to see the effect of the weight on the chain. We can see that we have the same form of solution as in the first part and if we plot the optimal solution curves for the two methods, the solutions are confused (the solution vectors returned are equal):

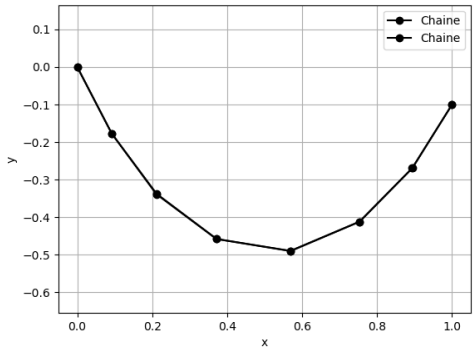
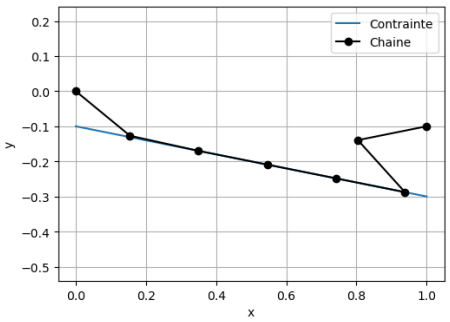
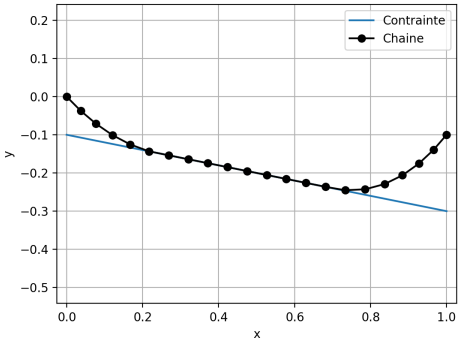


Figure 11: Solutions with the two methods for N=6

Now we're going to add the g_i constraint to the (P) problem. The g_i constraint represents a straight line below which the chain must not go.We add this constraint to the python code and solve the problem using the scipy.minimize function. We then plot the solution and the constraints for N=6, then for N=20. For the g_i constraint, we plot the straight line, and the admissible values are the values above the straight line.



(a) Plot of solution with constraints for N=6



(b) Plot of solution with constraints for N=20

We can see that the g_i constraint is respected because the chain does not go lower than the blue straight line representing the constraint for all values of N. We can see that the chain will come to rest on the line and we can easily imagine intuitively that this is an equilibrium position with the minimum potential energy. The problem is therefore well solved.