Numerical Project in Python n.1

An articulated chain

Andrea Simonetto

version: January 4, 2023

We will consider the problem of finding the static equilibrium of a chain formed by rigid bars in 2D. This will be found via an optimization problem with equality constraints.

The goal of this project is for you to code a Newton's method and experience what it means to deal with non-convex problems, as well as convex relaxations.

Some instructions.

- You can do the project in pairs, as long as everybody does a comparable amount of work. Since you will have a second numerical project, keep the same partner in both projects and send the files at the same time.
- You need to submit a single .pdf file, of no more than 6 pages, that is readable (for example, label all your graphs in a readable manner, with a reasonable fontsize), and it has inside all the elements to assess your work. Name the file: `LastName1_LastName2_Project1_Final.pdf`
- You also need to submit a jupyter notebook that we can run to assess the correctness of your plots and graphs. The notebook will have to generate the results that you have in the report. And it has to be readable. Name the file: `LastName1_LastName2_Project1_Final.ipynb`
- The deadline for sending us the **TWO** files is **Thursday February the 23nd, at 12H00 Paris time**, via the Moodle page of the course.
- As you figured already, you will use python, and in particular you will need the following packages (please refrain from using any other packages that are not strictly needed).

```
import numpy as np
import matplotlib.pyplot as plt
import cvxpy as cp
```

- Do as much as you can during the TP session (3 hour) at ENSTA, but it is possible that you will need to do extra work in addition to it.
- Don't wait the last day to put together the project.

Getting started

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 $i \in [1, \dots, N+1]$. We take as decision variables the position of the extremities of the bar (x_i, y_i) for $i \in [1, \dots, N+1]$, and we fix the two ends at $(0, 0)$ and (a, b), respectively.

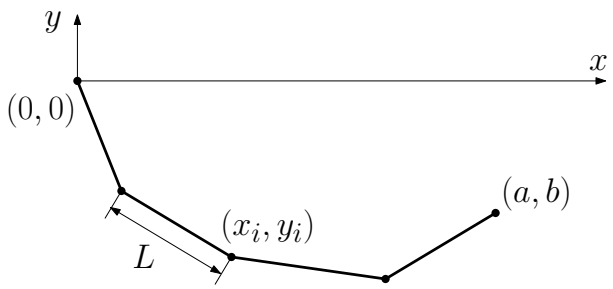

Figure 1: The articulated chain. (x_i, y_i) are the decision variables.

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 $i \in [1, \dots, N]$, since the $N+1$ is constrained to be (a, b).

If bar i has for extremities the coordinates at locations $i-1$ and i, then its length is

$$\ell_i(x, y) = \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2},$$

and we set $(x_0, y_0) = (0, 0)$, and $(x_{N+1}, y_{N+1}) = (a, b)$. Therefore, up to a constant, the potential energy of the chain is

$$E(x, y) = \sum_{i=1}^{N+1} g m_i(x, y) \frac{y_{i-1} + y_i}{2},$$

where g is the gravity constant and $m_i(x, y)$ is the mass of the i-th bar, supposed to be homogeneous. For the sake of simplicity, we take $g = 1$ and $m_i = \ell_i$, which yields,

$$E(x, y) = \sum_{i=1}^{N+1} \ell_i(x, y) \frac{y_{i-1} + y_i}{2}.$$

As constraints, we will have the lengths of the bars, which we can write as

$$c_i(x, y) = \ell_i(x, y)^2 - L^2 = (x_i - x_{i-1})^2 + (y_i - y_{i-1})^2 - L^2,$$

where we consider the squares, so that the contraints are differentiable.

We are then solving the problem,

$$(P_0): \qquad \min_{x \in \mathbf{R}^N, y \in \mathbf{R}^N} E(x, y) \qquad \text{subject to: } c_i(x, y) = 0,\ i \in [1, \dots, N+1],$$

or, since at optimality the constraints are verified,

$$E(x, y) = \sum_{i=1}^{N+1} \ell_i(x, y) \frac{y_{i-1} + y_i}{2} = \sum_{i=1}^{N+1} L\ \frac{y_{i-1} + y_i}{2},$$

so we can solve equivalently the problem,

$$(P): \qquad \min_{x \in \mathbf{R}^N, y \in \mathbf{R}^N} \sum_{i=1}^{N} y_i \qquad \text{subject to: } c_i(x, y) = 0,\ i \in [1, \dots, N+1],$$

Goal of the project

The goal of the project is to write an algorithm in Python to solve the problem (P) for the variables (x, y), both in $\mathbf{R}^N$. Introducing the Lagrangian multiplier $\lambda \in \mathbf{R}^{N+1}$, then the Lagrangian function will be

$$\mathcal{L}(x, y, \lambda) = \sum_{i=1}^{N} y_i + \sum_{i=1}^{N+1} \lambda_i c_i(x, y).$$

Setting up

To solve Problem (P), we proceed step by step. First, we pack the decision variables in a **column** vector $z \in \mathbf{R}^{2N} := (x, y)$, so that you will have all the x's first and then all the y's.

Then we define a **column** vector function $c(z) : \mathbf{R}^{2N} \to \mathbf{R}^{N+1}$, that given the coordinates z, return the length squared of each bars minus L^2, i.e., $c(z) = (c_1(x, y), \dots, c_{N+1}(x, y))$.

The cost function of our problem $\sum_i y_i$, can also be written in terms of z, as $e^\top z$, where e is a vector of N zeros stacked with a vector of N 1's.

Remark: at this point you should have three vectors of the form,

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

Finally, the Lagrangian can be written as

$$\mathcal{L}(z, \lambda) = e^\top z + \lambda^\top c(z).$$

Optimality conditions

We write the optimality conditions for (P) as

$$\begin{cases} \nabla_z \mathcal{L}(z, \lambda) = 0 \\ \nabla_\lambda \mathcal{L}(z, \lambda) = c(z) = 0 \end{cases}$$

And we will solve the non-linear system via a Newton's method.
First, let's look at the derivatives. We know that

$$\nabla_z \mathcal{L}(z, \lambda) = e + \nabla_z^\top c(z)\lambda \in \mathbf{R}^{2N},$$

where

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

and, for example, for each $\nabla_{x_i} c_i(x, y)$ we have

$$\nabla_{x_i} c_i(x, y) = 2(x_i - x_{i-1})$$

and $c(z)$ is actually a linear matrix in z.
Then, we also write a function to compute the Hessian of the Lagrangian for checking which stationary point we will get.

Questions

1. **[1 point]** Is the problem convex?
2. **[1 point]** What does the optimality conditions tell you? (I.e., comment on constraint qualification, and the meaning of (z, λ) to be a KKT point).
3. **[1 point]** Would you expect to have a solution, for all parameters (a, b, N, L)? Write a simple condition, that, if verified, a solution of (P) exists.

The Newton's method

Given optimality conditions

$$\begin{cases} e + \nabla_z^\top c(z)\lambda = 0 \\ c(z) = 0 \end{cases}$$

we want to solve for z, λ via a Newton's method.
We recall here the steps. Let $F(\xi) = 0$ represent the above non-linear system of equation, with $\xi = (z, \lambda)$. Then a Newton's method would start with an initial condition ξ^0, and perturb $F(\xi) = 0$ as

$$F(\xi^k + d) \approx F(\xi^k) + \nabla_\xi F(\xi^k) d = 0$$

and solve for d as

$$d = -(\nabla_\xi F(\xi^k))^{-1} F(\xi^k)$$

then $\xi = \xi + d$, or possibly with backtracking $\xi = \xi + \alpha d$, $\alpha \in (0, 1]$.

Let's start with computing $\nabla_\xi F(\xi^k)$ for any ξ^k. As said,

$$F(\xi) : \mathbf{R}^{2N+N+1} \to \mathbf{R}^{2N+N+1}, \quad F(\xi) = \begin{bmatrix} e + \nabla_z^\top c(z)\lambda \\ c(z) \end{bmatrix},$$

so

$$\nabla_\xi F(\xi) \in \mathbf{R}^{2N+N+1 \times 2N+N+1}, \qquad \nabla_\xi F(\xi) = \begin{bmatrix} \nabla_z\Big(\nabla_z^\top c(z)\lambda\Big) & \nabla_z^\top c(z) \\ \nabla_z c(z) & 0 \end{bmatrix}$$

Where you could verify that

$$\nabla_z\Big(\nabla_z^\top c(z)\lambda\Big) = \left[\nabla_{x_j} \sum_{k=1}^{N+1} \nabla_{x_i} c_k(x,y)\lambda_k\right]_{ij} \in \mathbf{R}^{2N\times 2N} = \nabla_{zz}\mathcal{L}(z,\lambda).$$

In this project n.1 you have access to a python notebook that allows you to code directly a Newton's method.

1. **[5 points]** Code a Newton's method to deliver to find the stationary points of (P) and a routine that plots the resulting solution (i.e., the chain). The Newton's method can have the option to have a backtracking strategy, and you need to be able to specify the stopping criterion, and the maximum number of iterations.

   For this you use the python notebook `Project-chain.ipnb` and in particular you need to code the function `newton_iteration`. You have already the gradient of F as `A` and F itself as `rhs`.

   You need to return also the optimality gap as the norm of F. Indeed, the more F is different from zero, the farther you are from the solution of the non-linear system and the farther from optimality.

   For the backtracking strategy **you need to be careful!**. We are optimizing a constrained problem, so a backtracking on the cost function is pointless. A backtracking on the Lagrangian is dangerous (since we have not proven that the Lagrangian has to diminish), and therefore perform a backtracking on $\|F(\xi)\|$. Check Algorithm 10.2 of Boyd's Convex Optimization Book for an example.

2. **[3 points]** Plot the solution, convergence, type of stationary point, and relevant metrics for the following starting setting:

   ```
   import numpy as np
   N = 5
   L = 0.25
   a,b = 1,-.1
   x = np.zeros(2*N)
   x[0:N] = np.linspace(1./(N+1), a-1./(N+1), N)
   x[N:2*N] =[b*i -.1*(i-a/2.)**2 for i in x[0:N]]
   lmbda = 0.1*np.random.random(N+1)
   ```

   with and without a backtracking strategy, and for different realizations of λ. What do you observe? Is what you see, what you expect?

3. **[2 points]** Explore the solutions for different values of N, L and initializations. Comment on convergence.

Convex relaxation

We look now at a different way to solve the same problem, with the concept of convex relaxation. The aim of this part is to show that convexity can be useful to solve non-convex problems as well, in special cases.

What you should have seen in the first part of the project is that, in this non-convex problem, Newton's method is very sensitive to initialization and it can diverge pretty quickly, so it is hard to find good solution for large N.

Here we show how to deal with this.

Stack the decision variables in a matrix as

$$u = [x, y] = \left[\begin{array}{c|c} x_0 & y_0 \\ x_1 & y_1 \\ \vdots & \vdots \\ x_{N+1} & y_{N+1} \end{array}\right] \in \mathbf{R}^{N+2\times 2},$$

here for simplicity we have also added x_0, y_0 and x_{N+1}, y_{N+1}, which you will have to constrain to $(0,0)$ and (a,b).

Define now a matrix as

$$X = uu^\top = \left[\begin{array}{ccc} \ddots & & \\ & x_i^2 + y_i^2 \quad x_i x_{i+1} + y_i y_{i+1} & \cdots \\ & & \ddots \\ \star & & \ddots \end{array}\right] \in \mathbf{S}^{N+2}$$

and consider the problem (equivalent to (P))

$$\begin{array}{lll} (P'): & \min\limits_{u\in\mathbf{R}^{N+2\times 2}, X\in\mathbf{S}^{N+2}} & \sum(u[:,1]) \\ & \text{subject to:} & X_{i,i} - 2X_{i,i+1} + X_{i+1,i+1} - L^2 = 0,\ i \in [0,\ldots,N], \\ & & X = uu^\top, \\ & & u[0,:] = (0,0), u[N+1,:] = (a,b), \quad X_{0,0} = 0, X_{N+1,N+1} = a^2 + b^2 \end{array}$$

where $u[:,1]$ represents the second column of u.

Question

4. **[1 point]** Show that $(P) \equiv (P')$.

Problem (P') is still non-convex, but if you relax the constraint $X = uu^\top$, as $X \succeq uu^\top$, and then write this via Schur's complement as:

$$\left[\begin{array}{cc} I_2 & u^\top \\ u & X \end{array}\right] \succeq 0.$$

then the resulting problem,

$$\begin{array}{lll} (P''): & \min\limits_{u\in\mathbf{R}^{N+2\times 2}, X\in\mathbf{S}^{N+2}} & \sum(u[:,1]) \\ & \text{subject to:} & X_{i,i} - 2X_{i,i+1} + X_{i+1,i+1} - L^2 = 0,\ i \in [0,\ldots,N], \\ & & \left[\begin{array}{cc} I_2 & u^\top \\ u & X \end{array}\right] \succeq 0., \\ & & u[0,:] = (0,0), u[N+1,:] = (a,b), \quad X_{0,0} = 0, X_{N+1,N+1} = a^2 + b^2 \end{array}$$

is the convex relaxation of (P').

The solution of (P'') is not in general a solution for (P'), but it could be a good initialization for the Newton's method. In addition, if you find a solution for (P'') for which $rank(X^*) = 2$, then you can decompose $X^* = uu^\top$, and therefore the solution X^* leads to a solution in terms of x^*, y^* for (P').

Questions

5. **[1 point]** Show that (P'') is convex.
6. **[2 points]** Show formally that if you find a solution for (P'') for which $rank(X^*) = 2$, then $X^* = uu^\top$, and therefore the solution X^* leads to a solution in terms of x^*, y^* for (P'). But if $rank(X^*) > 2$ this is in general not true.
7. **[4 points]** Use `cvxpy` to solve (P'') for the same instances you tried for the Newton's method and show numerically that the convex relaxation is good (delivering X^* with rank very close to 2, meaning the eigenvalues from the 3rd on are very small compared to the first two).
8. **[2 points]** Use the solutions you obtain from (P'') to initialize the Newton's method, and show that now you can solve a larger variety of problems.

Total: /23 **Points.**

Sample solutions

Below a sample of the solutions you are expected to get.

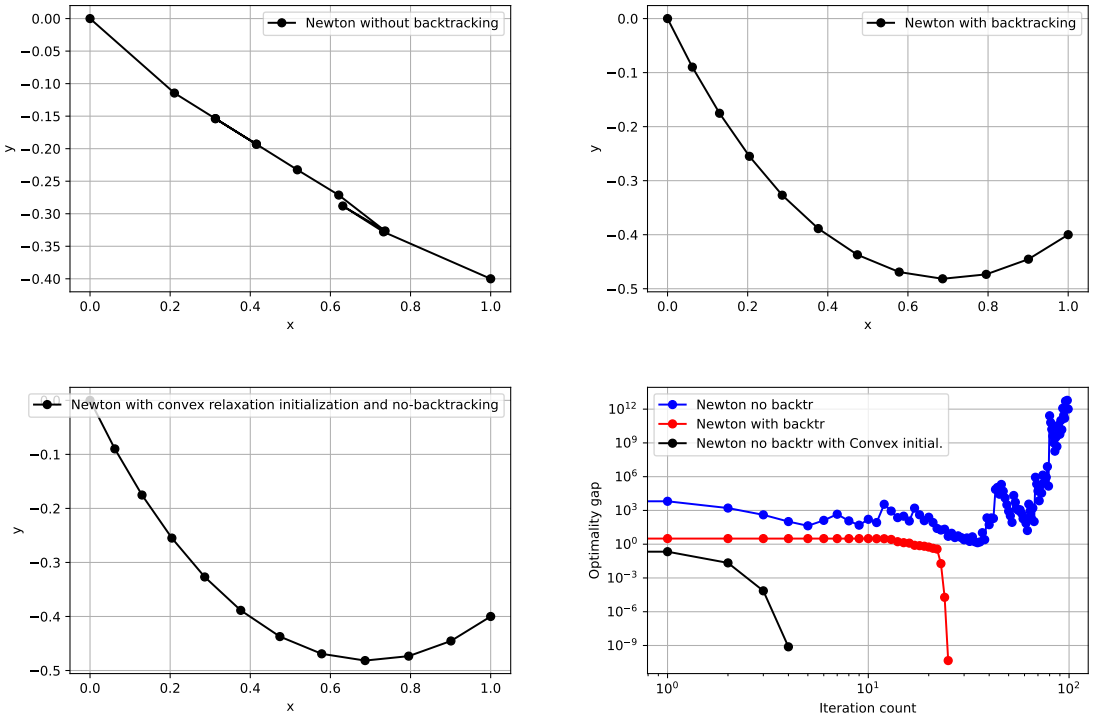

Figure 2: A sample of what you should obtain in your project.