Projet d'OPT-201 : Équilibre d'une chaîne articulée

Leonardo Martins Bianco - Guillaume Lambert

7 février 2021

Table des matières

Introduction

Ce projet d'optimisation considère le problème de trouver la position d'équilibre statique d'une chaîne formée de barres rigides, contenue dans un plan vertical et fixée à ses deux bouts. Le langage de programmation utilisé est *Matlab*.

Dans un premier temps, on va modéliser le problème et écrire le simulateur calculant les différentes grandeurs du problème. Ensuite, on utilisera une méthode newtonienne pour trouver les points stationnaires du problème et on les déterminera grâce à l'implémentation d'un optimiseur. Enfin, l'algorithme de Newton ayant une convergence uniquement locale (sous des conditions de régularité), on globalisera cet algorithme par recherche linéaire à travers la règle d'Armijo. La qualité de la convergence de l'algorithme de Newton sera évaluée dans les cas de convergence.

1 Modélisation du problème et écriture du simulateur

Tout d'abord, nous introduisons la modélisation de la situation et étudions le problème. Ensuite, nous traitons l'implémentation du simulateur et sa vérification. Enfin, nous appliquons notre code sur un cas-test concret.

1.1 Modélisation du problème

Trouver la position d'équilibre statique de la chaîne revient à minimiser l'énergie potentielle de la chaîne sous des contraintes appropriées.

1.1.1 Détermination des variables du problème

On considère les coordonnées des noeuds décrivant la position de la chaîne : ses points d'articulation. Les points $(0,0)$ et (a,b) aux extrémités étant fixes, ils ne sont pas considérés dans le jeu de variables. On note N_b le nombre de barres, et ainsi on a $N_n = N_b - 1$ noeuds "libres". Lors de nos études, l'extrémité (a,b), le nombre de barres N_b et leurs longueurs pourront varier.

On note les coordonnées du noeud i par $(\mathrm{x}_i, \mathrm{y}_i)$. Si la barre i de la chaîne a pour extrémités les noeuds d'indices $i-1$ et i, sa longueur vaut

$$l_i(\mathrm{x},\mathrm{y}) = \sqrt{(\mathrm{x}_i - \mathrm{x}_{i-1})^2 + (\mathrm{y}_i - \mathrm{y}_{i-1})^2}$$

Les variables du problème sont donc les couples $(\mathrm{x}_i, \mathrm{y}_i)_{1 \leq i,j \leq N_b - 1}$. On note également $(\mathrm{x}_0, \mathrm{y}_0) = (0,0)$ et $(\mathrm{x}_{N_b}, \mathrm{y}_{N_b}) = (a,b)$.

1.1.2 Energie du problème

On fait, sans perte de généralité, l'hypothèse que la masse de chaque barre est égale à sa longueur et que la constante gravitationnelle vaut 1. Ainsi, l'énergie potentielle totale du système est donnée par

$$\mathrm{E}(\mathrm{x},\mathrm{y}) = \sum_{i=1}^{n_b} l_i(\mathrm{x},\mathrm{y}) \frac{\mathrm{y}_i + \mathrm{y}_{i-1}}{2}$$

1.1.3 Contraintes

Les barres de la chaîne sont rigides, donc les contraintes à respecter concernent l'égalité des longueurs des barres par rapport aux longueurs données. Rigoureusement, ces contraintes sont données par

$$\tilde{c}_i(\mathrm{x},\mathrm{y}) = l_i - L_i = \sqrt{(\mathrm{x}_{i-1}-\mathrm{x}_i)^2 + (\mathrm{y}_{i-1}-\mathrm{y}_i)^2} - L_i = 0$$

Cependant, à cause de la racine carrée qui apparaît dans la formule ci-dessus, $\tilde{c}_i$ n'a pas des bonnes propriétés de différentiabilité, qui sont très importantes pour que l'on puisse appliquer les théorèmes vus en cours.

Ainsi, on choisit des fonctions de contrainte équivalentes, c'est-à-dire, qui donnent le même ensemble admissible, mais qui ont des propriétés de différentiabilité souhaitées :

$$c_i(\mathrm{x},\mathrm{y}) = l_i^2 - L_i^2 = (\mathrm{x}_i - \mathrm{x}_{i-1})^2 + (\mathrm{y}_i - \mathrm{y}_{i-1})^2 - L_i^2 = 0$$

Ces fonctions sont des polynômes quadratiques, alors elles sont différentiables.

Vérifions qu'elles donnent le même ensemble admissible pour le problème :

$$\begin{aligned}
&\tilde{c}_i(\mathrm{x},\mathrm{y}) = 0 \implies l_i = L_i \implies l_i^2 = L_i^2 \implies c_i(\mathrm{x},\mathrm{y}) = 0 \\
&c_i(\mathrm{x},\mathrm{y}) = 0 \implies l_i^2 = L_i^2; \quad (x^2 \text{ injective pour } x > 0, \text{c'est le cas des longueurs}) \\
&\qquad\qquad \implies l_i = L_i \implies \tilde{c}_i(\mathrm{x},\mathrm{y}) = 0
\end{aligned}$$

C'est à dire, $\tilde{c}_i(\mathrm{x},\mathrm{y}) = 0 \iff c_i(\mathrm{x},\mathrm{y}) = 0$. Ainsi, l'ensemble admissible est exactement le même, et on peut utiliser les contraintes c_i.

1.1.4 Expression du problème

On peut maintenant poser le problème :

$$\begin{cases} \min \mathtt{E}(\mathrm{x},\mathrm{y}) \\ c_i(\mathrm{x},\mathrm{y}) = 0, \quad i = 1,\dots,N_b \end{cases}$$

On remarque que sur l'ensemble admissible l'expression de l'énergie $\mathtt{E}$ devient linéaire. Comme on s'intéresse à résoudre le problème juste sur l'ensemble admissible, on peut changer la fonction-objectif $\mathtt{E}$ par une autre fonction-objectif équivalente e. La forme finale du problème est donc la suivante :

$$(P) \quad \begin{cases} \min e(\mathrm{x},\mathrm{y}) \\ c_i(\mathrm{x},\mathrm{y}) = 0, \quad i = 1,\dots,N_b \end{cases}$$

où

$$e(\mathrm{x},\mathrm{y}) := \sum_{i=1}^{n_b} L_i \frac{\mathrm{y}_i + \mathrm{y}_{i-1}}{2}$$

On va maintenant se poser 2 questions sur la résolution de ce problème.

1.2 Analyse du problème

1.2.1 Existence de solutions

La première question à se poser une fois le problème posé est l'existence de solutions si l'ensemble admissible est non vide.

L'existence de solutions est un fait topologique de l'ensemble admissible. Dans notre cas, l'ensemble admissible correspond à l'ensemble où les contraintes d'égalité sont satisfaites. Nous allons utiliser le théorème de Weierstrass pour montrer l'existence d'une solution s'il existe une chaîne admissible. On travaille avec la norme usuelle $\|.\|$ sur $\mathbb{R}^{(N_b+1)\times 2}$. Toute chaîne est un vecteur de cet espace, si on garde les coordonnées de ses noeuds dans un vecteur $(x_0, x_1, \ldots, x_{N_b}, y_0, y_1, \ldots, y_{N_b})$.

L'ensemble admissible est un compact de $\mathbb{R}^{(N_b+1)\times 2}$: étant en dimension finie, on montre qu'il est borné et fermé.

L'ensemble admissible est borné :
Soit $x = \left((\mathrm{x}_i, \mathrm{y}_i)_{i=0}^{N_b}\right)$ une chaîne admissible. On pose L la longueur totale de la chaîne, et α la plus grande composante de x : $\alpha = \max\{\max_{i=0,\cdots,N_b+1}\{\mathrm{x}_i\}, \max_{i=0,\cdots,N_b+1}\{\mathrm{y}_i\}\}$. Etant donné que les barres sont de longueur finie et sont en nombre finie, on a toujours l'existence de α.

Par l'inégalité triangulaire :

$$\|x\| = \sqrt{\mathrm{x}_0^2 + \mathrm{x}_1^2 + \cdots + \mathrm{x}_{N_b}^2 + \mathrm{y}_0^2 + \mathrm{y}_1^2 + \cdots + \mathrm{y}_{N_b}^2} \leq \sqrt{2(N_b+1)}\alpha \leq \sqrt{2(N_b+1)}L$$

Donc l'ensemble admissible est borné.

L'ensemble admissible est fermé :
On utilise la caractérisation séquentielle. Soit $x = \left((\mathrm{x}_i, \mathrm{y}_i)_{i=0}^{N_b}\right)$ une chaîne admissible et $(x^{(n)})_n = \left((\mathrm{x}_i^{(n)}, \mathrm{y}_i^{(n)})_{i=0}^{N_b}\right)_n$ une suite de chaîne convergente vers x. Chaque chaîne est un vecteur de $\mathbb{R}^{2(N_b+1)}$ et on utilise la norme euclidienne usuelle de cet espace. On a

$$\lim_{n\to\infty} \|x - x^{(n)}\| = \lim_{n\to\infty} \sqrt{(\mathrm{x}_0 - \mathrm{x}_0^{(\mathrm{n})})^2 + \cdots + (\mathrm{x}_{N_b} - \mathrm{x}_{\mathrm{N_b}}^{(\mathrm{n})})^2 + (\mathrm{y}_0 - \mathrm{y}_0^{(\mathrm{n})})^2 + \cdots + (\mathrm{y}_{N_b} - \mathrm{y}_{\mathrm{N_b}}^{(\mathrm{n})})^2} = 0$$

Donc $\lim_{n\to\infty} |x_i - x_i^{(n)}| = 0$ et $\lim_{n\to\infty} |y_i - y_i^{(n)}| = 0$. C'est-à-dire, chaque noeud de la chaîne $x^{(n)}$ converge vers le noeud correspondant dans la chaîne x.

Cette convergence entraîne la convergence des longueurs entre les points, donc $x^{(\infty)} := \lim_{n\to\infty} x^{(n)}$ appartient au domaine admissible. On prouve cela par 2 méthodes différentes :

Méthode 1 :
Soit, par exemple, $p_1^{(n)}$ la suite formée par le premier noeud de chaque chaîne $x^{(n)}$ et p_1 le premier noeud da la chaîne limite (ces noeuds ne sont pas fixés). Or (0 correspond au noeud fixé à l'origine) :

$$p_1^{(n)} \longrightarrow p_1 \quad \text{alors} \quad p_1^{(n)} - 0 \longrightarrow p_1 - 0$$

$$\underbrace{\|p_1^{(n)} - 0\|}_{=l_1 \text{ constante } \forall n} \longrightarrow \underbrace{\|p_1 - 0\|}_{l_1} \quad \text{alors} \quad l_1^\infty = l_1$$

où on a utilisé la continuité de la norme dans un espace vectoriel normé, et où on note la longueur de la i-ème barre de la chaîne limite par l_i^∞. De manière identique, on prouve que toutes les longueurs sont préservées :

$$\underbrace{\|p_{i-1}^{(n)} - p_i^{(n)}\|}_{=l_i \text{ constante } \forall n} \longrightarrow \underbrace{\|p_{i-1} - p_i\|}_{l_i^\infty} \quad \text{alors} \quad l_i^\infty = l_i$$

Méthode 2 :
Par inégalité triangulaire :

$$\begin{aligned}
&|l_i^2 - l_i^{(n)2}| \\
&= |(x_{i-1} - x_i)^2 - (x_{i-1}^{(n)} - x_i^{(n)})^2 + (y_{i-1} - y_i)^2 - (y_{i-1}^{(n)} - y_i^{(n)})^2| \quad \text{puis on utilise} (a-b)(a+b) = a^2 - b^2 \\
&= |(x_{i-1} - x_{i-1}^{(n)} + x_i^{(n)} - x_i)(x_{i-1} - x_i + x_i^{(n)} - x_{i-1}^{(n)}) + (y_{i-1} - y_{i-1}^{(n)} + y_i^{(n)} - y_i)(y_{i-1} - y_i + y_i^{(n)} - y_{i-1}^{(n)})| \\
&\leq |(x_{i-1} - x_{i-1}^{(n)} + x_i^{(n)} - x_i)(x_{i-1} - x_i + x_i^{(n)} - x_{i-1}^{(n)})| + |(y_{i-1} - y_{i-1}^{(n)} + y_i^{(n)} - y_i)(y_{i-1} - y_i + y_i^{(n)} - y_{i-1}^{(n)})| \\
&\leq (\underbrace{|x_{i-1} - x_{i-1}^{(n)}|}_{\to 0} + \underbrace{|x_i^{(n)} - x_i|}_{\to 0}) (|x_{i-1} - x_i| + |x_i^{(n)} - x_{i-1}^{(n)}|) + (\underbrace{|y_{i-1} - y_{i-1}^{(n)}|}_{\to 0} + \underbrace{|y_i^{(n)} - y_i|}_{\to 0}) (|y_{i-1} - y_i| + |y_i^{(n)} - y_{i-1}^{(n)}|)
\end{aligned}$$

Donc $\lim_{n\to\infty} l_i^{(n)} = l_i$. Or $\forall n \in \mathbb{N}$, on a $l_i^{(n)} = l_i$ car la chaîne $x^{(n)}$ est admissible. Donc $\forall n \in \mathbb{N}$, $l_i^{(n)}$ est constante et les longueurs de la chaîne $\lim_{n\to\infty} x^{(n)} = x^{(\infty)}$ vérifient la condition d'admissibilité : $l_i^{(\infty)} = l_i$.

Alors la chaîne $(\tilde{x}, \tilde{y})$ est admissible, et donc l'ensemble admissible est fermé.

Voici une représentation graphique de la convergence de chaînes décrit ci-dessus, dans le cas test :

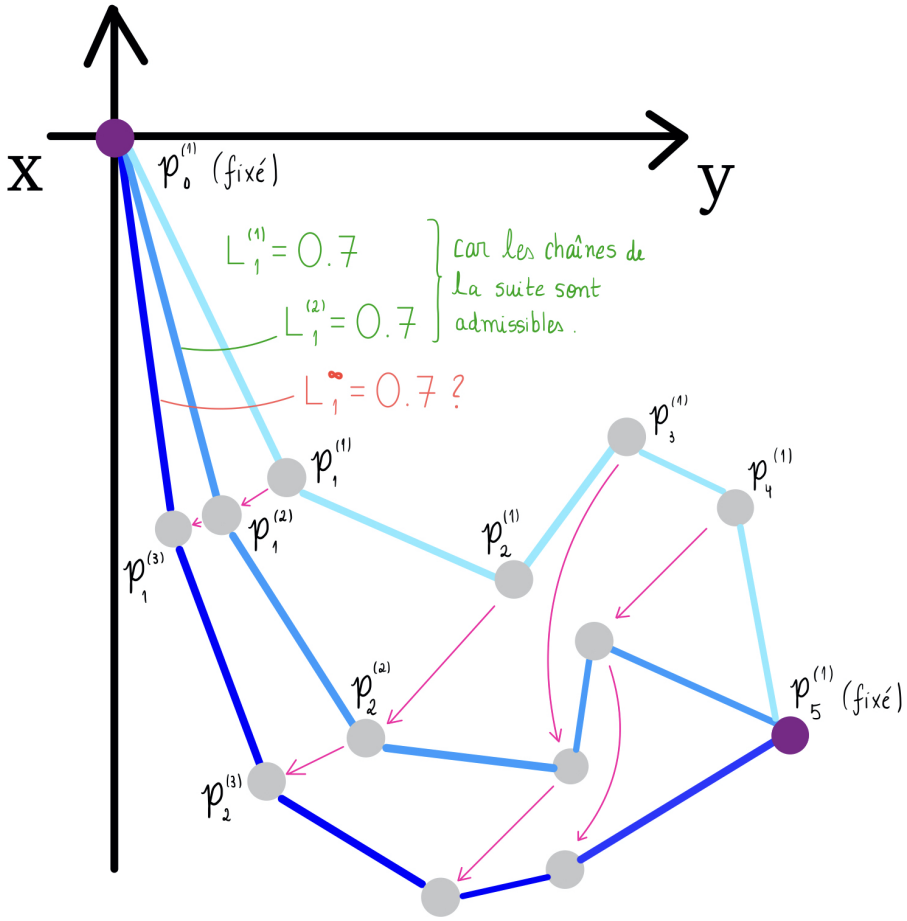


FIGURE 1 – Les chaînes bleu clair convergent vers la chaîne bleu foncé

On conclut ainsi que l'ensemble admissible est un borné fermé de $\mathbb{R}^{N_b+1}$, donc compact. On voit aussi que la fonction-objectif e est linéaire sur les (x, y), alors elle est continue. On peut donc appliquer le théorème de Weierstrass, qui dit que sous ces hypothèses, la fonction atteint son minimum en un certain point x_*. Ce point est une solution du problème d'optimisation.

1.2.2 Existence de multiplicateurs λ_*

On se demande maintenant si étant donné une solution x_* du problème primal, peut-on en déduire qu'il existe un multiplicateur optimal λ_* tel que $\nabla_x \ell(x_*, \lambda_*) = 0$? Autrement dit, peut-on appliquer la condition nécessaire d'ordre 1 de KKT ? Etant donné que le critère e et les contraintes c sont différentiables sur l'ensemble admissible, cela revient à se demander si les contraintes sont qualifiées sur l'ensemble admissible.

Intuitivement, cela revient à dire que la représentation analytique choisie pour l'ensemble

admissible via les fonctions c_i n'est pas rédondante. Or, si on avait plus de fonctions que nécessaire, il y aurait dans le cône tangent une relation de dépendence linéaire entre les gradients ∇c_i. Quand les contraintes sont qualifiées, ces gradients engendrent le cône tangent sans excès, c'est-à-dire, ils sont linéairement indépendants.

Le gradient ∇c_i s'écrit

$$\nabla c_i = \left(0, \dots, 0, \frac{\partial c_i}{\partial x_{i-1}}, \frac{\partial c_i}{\partial x_i}, 0, \dots, 0, \frac{\partial c_i}{\partial y_{i-1}}, \frac{\partial c_i}{\partial y_i}, 0, \dots, 0\right)$$

A cause des zéros dans le gradient ∇c_i, il suffit vérifier l'indépendance linéaire entre ∇c_{i-1} et ∇c_i. Supposons qu'il existe $\alpha \in \mathbb{R}$ tel que $\nabla c_{i-1} = \alpha \nabla c_i$, on obtient les équations

$$\begin{aligned} -2(x_{i-1} - x_{i-2}) &= 0 \\ 2(x_{i-1} - x_{i-2}) &= -2(x_i - x_{i-1}) \\ -2(y_{i-1} - y_{i-2}) &= 0 \\ 2(y_{i-1} - y_{i-2}) &= -2(y_i - y_{i-1}) \end{aligned}$$

Si l'on a $x_{i-1} - x_{i-2} \neq 0$ et $y_{i-1} - y_{i-2} \neq 0$, alors on voit que ce système n'a pas de solution à cause de la première et troisième équations (et les contraintes seraient donc qualifiées).

Cependant il faut noter que les conditions $x_{i-1} - x_{i-2} \neq 0$ et $y_{i-1} - y_{i-2} \neq 0$ ne sont pas toujours vraies. Elles correspondent à l'existence des barres verticales ou horizontales dans la chaîne en équilibre, et comme on verra plus tard, cela peut arriver (voir le cas-test 3.b). Dans ces cas, les contraintes ne seront pas qualifiées et on ne peut pas conclure l'existence de $\lambda_\star$ à partir de l'existence de $x_\star$.

On va maintenant expliquer le code que l'on a écrit pour le *simulateur* de ce problème.

1.3 Implémentation du code

La structure du code est la suivante :

— Le programme principal `ch` est un script contenant les données du problème et appelant le simulateur *chs*.

— Le simulateur `chs` calcule `e`, `c`, `g`, `a` et `hl`. Chacune de ces grandeurs sont calculées respectivement par les fonctions *calculatrices* : `fun_e`, `fun_c`, `fun_g`, `fun_a` et `fun_hl`. L'utilisation de telles fonctions facilite la réutilisation et généralisation du code.

— La fonction `test_derivees` affiche l'étude de l'exactitude du gradient de `e`.

— La fonction `verif_param` vérifie que les paramètres soient corrects avant d'effectuer les calculs.

Remarque : Les fonctions *calculatrices* peuvent prendre en paramètres des éléments non utiles à leur bon fonctionnement. Par exemple, `e` ne dépend pas de `x` mais reçoit `x` en paramètre. Ainsi, on préserve la notation en tant que fonction des coordonnées $(\mathtt{x}_i, \mathtt{y}_i)$, et le code pourra évoluer en cas de modifications du problème étudié.

1.3.1 Simulateur chs

De façon générale, le simulateur reçoit les données du problème du fichier principal `ch` et une variable `indic`, qui sélectionne une des fonctionnalités du simulateur. Voici la documentation de la fonction simulateur `chs` :

```
function [e,c,g,a,hl,indic] = chs(indic,xy,lm)
```

Paramètres d'entrée

- `indic` : entier pilotant le comportement du simulateur
  - 1 : tracé de la chaîne
  - 2 : calcule `e` et `c`
  - 4 : calcule `e`, `c`, `g` et `a`
  - 5 : calcule `hl`
  - 6 : affiche l'étude de l'exactitude du gradient de `e`
- `xy` : vecteur-colonne des abscisses puis des ordonnées des points d'articulation de la chaîne, $\mathrm{xy} = (\mathrm{x}_1, ..., \mathrm{x}_{n_n}, \mathrm{y}_1, ..., \mathrm{y}_{n_n})$. Il est de dimension $(2n_n, 1)$.
- lm : vecteur-colonne des multiplicateurs de Lagrange pour les contraintes d'égalité $\lambda = \{\lambda_i\}_{i=1}^{n_b}$. Il est de dimension $(n_b, 1)$.

Valeurs renvoyées

Nous présentons les valeurs renvoyées ainsi que leurs fonctions *calculatrices* correspondantes : `fun_e`, `fun_c`, `fun_g`, `fun_a` et `fun_hl`. De plus, nous détaillons les fonctions `fun_trace`, `verif_param` et `test_derivees`.

Calcul de l'énergie potentielle `e` est la valeur de l'énergie potentielle en `xy`. Son expression est

$$e(\mathrm{x}, \mathrm{y}) := \sum_{i=1}^{n_b} L_i \frac{\mathrm{y}_i + \mathrm{y}_{i-1}}{2}$$

Elle est calculée par la fonction `fun_e`.

Calcul des contraintes `c` est la valeur en `xy` des contraintes sur la longueur des barres. C'est un vecteur-colonne de dimension $(n_b, 1)$. Son expression est, pour $i \in \{1, ..., n_b\}$,

$$c_i(\mathrm{x}, \mathrm{y}) = l_i^2 - L_i^2 = (\mathrm{x}_i - \mathrm{x}_{i-1})^2 + (\mathrm{y}_i - \mathrm{y}_{i-1})^2 - L_i^2$$

Elle est calculée par la fonction `fun_c`.

Calcul du gradient de l'énergie potentielle `g` est le gradient de `e` en `xy`, noté $\nabla e(x)$. C'est un vecteur-colonne de dimension $(2n_n, 1)$. Il est calculé par la fonction `fun_g`.

Calcul théorique

Calculons le gradient de e. On voit que e ne dépend pas de $x_k, 1 \leq k \leq N_n$. Donc,

$$\frac{\partial e}{\partial x_k}(x,y) = 0$$

Les dérivées par rapport aux variables y_k sont facilement calculées, car e est linéaire par rapport à elles

$$\frac{\partial e}{\partial y_k}(x,y) = \frac{L_k + L_{k+1}}{2}$$

Ainsi on obtient que ∇e peut être calculé à partir des longueurs des barres seulement :

$$\nabla e(\mathrm{x},\mathrm{y}) = \Big(\underbrace{0,\ldots,0}_{N_n}, \frac{L_1+L_2}{2}, \cdots, \frac{L_{N_b}+L_{N_b-1}}{2}\Big)$$

Calcul de la Jacobienne des contraintes On note par `a` la jacobienne des contraintes d'égalité en `xy`, notée $A(x) = c'(x)$. C'est une matrice de dimension $(n_b, 2n_n)$. Elle est calculée par la fonction `fun_a`.

Calcul théorique

Rappelons que la matrice jacobienne de $c = (c_1, \ldots, c_{N_b})$ est donnée par

$$a = \begin{pmatrix} \frac{\partial c_1}{\partial x_1} & \frac{\partial c_1}{\partial x_2} & \cdots & \frac{\partial c_1}{\partial x_{N_n}} & \frac{\partial c_1}{\partial y_1} & \frac{\partial c_1}{\partial y_2} & \cdots & \frac{\partial c_1}{\partial y_{N_n}} \\ \frac{\partial c_2}{\partial x_1} & \ddots & & & & & & \\ \vdots & & & & & & & \\ \frac{\partial c_{N_b}}{\partial x_1} & & & & & & & \end{pmatrix}$$

Cette matrice de taille $N_b \times 2N_n$ peut être considérée comme la concatenation de deux sous-matrices $N_b \times N_n$, une avec les dérivées par rapport à x_k que l'on appelle a_x, l'autre avec les dérivées par rapport à y_k, que l'on appelle a_y. On calcule ces dérivées :

$$\frac{\partial c_i}{\partial x_k}(x,y) = \begin{cases} 2\,(x_i - x_{i-1}) & \text{si } k = i \\ 2\,(x_{i-1} - x_i) & \text{si } k = i-1 \\ 0 \quad \text{sinon} \end{cases}$$

et de façon identique

$$\frac{\partial c_i}{\partial y_k}(x,y) = \begin{cases} 2\,(y_i - y_{i-1}) & \text{si} \quad k = i \\ 2\,(y_{i-1} - y_i) & \text{si} \quad k = i-1 \\ 0 \quad \text{sinon} \end{cases}$$

Les matrices a_x et a_y sont donc nulles à l'exception des diagonales et sous-diagonales. Pour construire de manière efficace ces deux matrices, *Matlab* fournit la fonction spdiags. Elle est utile pour créer des matrices creuses dont les valeurs non nulles forment des diagonales. Enfin, on concaténe les matrices a_x et a_y pour former la matrice `a`. Le code est le suivant :

```
diag_x = sparse(Nn,1);  %correspond aux derivees par rapport a x_i
diag_y = sparse(Nn,1);  %correspond aux derivees par rapport a y_i

for i=1:Nn
  diag_x(i,1) = x(i+1) - x(i);
  diag_y(i,1) = y(i+1) - y(i);
endfor

a_x = spdiags([-2*diag_x 2*diag_x],[-1 0],Nb,Nn); %jacobienne par
    rapport aux x_i
a_y = spdiags([-2*diag_y 2*diag_y],[-1 0],Nb,Nn); %jacobienne par
    rapport aux y_i

%On assemble la jacobienne par rapport aux x_i et y_i
a = [a_x,a_y];
```

Calcul du Hessien du Lagrangien `hl` est le hessien du lagrangien en `xy` et lm, notée $\nabla^2\ell((\mathrm{x},\mathrm{y}),\lambda)$. C'est une matrice de dimension $(2n_n, 2n_n)$. Elle est calculée par la fonction `fun_hl`.

Calcul théorique

Le lagragien est donné par

$$\ell(x,\lambda) = e(x) + \lambda^\top c(x)$$

Le hessien du lagrangien, étant linéaire, est donc

$$\nabla^2_{xx}\ell(x,\lambda) = \nabla^2 e(x) + \sum_{i\in E}\lambda_i \nabla^2 c_i(x)$$

Tout d'abord, on a vu dans la partie 2.1.3 que le gradient de e ne dépend que des longueurs L_i. Ainsi, les dérivées secondes de e sont toutes nulles et $\nabla^2 e$ est la matrice nulle. On obtient que $\nabla^2\ell(x,\lambda) = \lambda^\top\lambda^2 c(x)$.

Calculons maintenant les dérivées secondes de c_i. À partir du calcul des dérivées de premier ordre dans la partie 2.1.4, on obtient

$$\frac{\partial c_i}{\partial x_l \partial x_k} = \begin{cases} 0 \\ 2\left(\delta_{il} - \delta_{(i-1)l}\right), & k = i \\ 2\left(\delta_{(i-1)l} - \delta_{il}\right), & k = i-1 \end{cases}$$

où δ_{ij} est le delta de Kronecker. Cette expression devient

$$\begin{cases} 2 & \text{si} \quad k = i = l \\ -2 & \text{si} \quad k = i, l = i-1 \\ 2 & \text{si} \quad k = i-1 = l \\ -2 & \text{si} \quad k = i-1, l = i \\ 0 & \text{sinon} \end{cases}$$

En faisant le même calcul, on vérifie que les dérivées $\frac{\partial^2}{\partial y_l \partial y_k}$ sont données par exactement les mêmes équations. Comme dans le cas de la Jacobienne, on considère alors une sous-matrice h_i de taille $N_n \times N_n$ telle que

$$\nabla^2 c_i = \begin{pmatrix} h_i & 0 \\ 0 & h_i \end{pmatrix}$$

Construction des matrices h_i :

— Pour $i \in \{1, ..., N_b - 1\}$, les matrices h_i partagent une même structure. Par exemple, pour $i = 3$, on observe

$$h_3 = \begin{pmatrix} 0 & 0 & 0 & 0 & \cdots \\ 0 & 2 & -2 & 0 & \\ 0 & -2 & 2 & 0 & \\ 0 & 0 & 0 & 0 & \\ \vdots & & & & \ddots \end{pmatrix}$$

— Pour $i = 1$, on observe

$$h_1 = \begin{pmatrix} 2 & 0 & \dots & 0 \\ 0 & 0 & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 \end{pmatrix}$$

— Pour $i = N_b$, on observe

$$h_1 = \begin{pmatrix} 0 & 0 & \cdots & \cdots & 0 \\ 0 & \ddots & & & \vdots \\ \vdots & & 0 & & 0 \\ \vdots & & & 2 & -2 \\ 0 & \vdots & 0 & -2 & 2 \end{pmatrix}$$

Ainsi

$$\sum_i \lambda_i h_i = \begin{pmatrix} 2(\lambda_1 + \lambda_2) & -2\lambda_2 & 0 & \cdots & 0 \\ -2\lambda_2 & 2(\lambda_2 + \lambda_3) & -2\lambda_3 & & \vdots \\ 0 & -2\lambda_3 & 2(\lambda_3 + \lambda_4) & & \vdots \\ \vdots & & & & 0 \\ \vdots & & & 2(\lambda_{N_b-1} + \lambda_{N_b}) & -2\lambda_{N_b} \\ 0 & \cdots & 0 & -2\lambda_{N_b} & 2\lambda_{N_b} \end{pmatrix}$$

Donc la matrice hessienne est donnée par

$$\begin{pmatrix}
2(\lambda_1+\lambda_2) & -2\lambda_2 & 0 & \cdots & 0 & 0 & \cdots & \cdots & \cdots & 0 \\
-2\lambda_2 & 2(\lambda_2+\lambda_3) & -2\lambda_3 & & \vdots & \vdots & & & & \vdots \\
0 & -2\lambda_3 & 2(\lambda_3+\lambda_4) & \ddots & 0 & \vdots & & & & \vdots \\
\vdots & & \ddots & \ddots & -2\lambda_{N_b} & \vdots & & & & \vdots \\
0 & \cdots & 0 & -2\lambda_{N_b} & 2\lambda_{N_b} & 0 & \cdots & \cdots & \cdots & 0 \\
0 & \cdots & \cdots & \cdots & 0 & 2(\lambda_1+\lambda_2) & -2\lambda_2 & 0 & \cdots & 0 \\
\vdots & & & & \vdots & -2\lambda_2 & 2(\lambda_2+\lambda_3) & -2\lambda_3 & & \vdots \\
\vdots & & & & \vdots & 0 & -2\lambda_3 & 2(\lambda_3+\lambda_4) & \ddots & 0 \\
\vdots & & & & \vdots & \vdots & & \ddots & \ddots & -2\lambda_{N_b} \\
0 & \cdots & \cdots & \cdots & 0 & 0 & \cdots & 0 & -2\lambda_{N_b} & 2\lambda_{N_b}
\end{pmatrix}$$

La matrice hessienne est creuse hormis la diagonale et les sous et sur diagonales. C'est pourquoi, comme dans la partie 2.1.4, la fonction spdiags permet de créer cette matrice efficacement. Voici le code :

```
diag = 2.*[lm(1:Nb-1,1)+lm(2:Nb,1);lm(Nb,1);lm(1:Nb-1,1)+lm(2:Nb,1);lm(
    Nb,1)];%diagonale
diag_inf = -2.*[[lm(2:Nb,1)];0;[lm(2:Nb,1);0]];%sous-diagonale
diag_sup = -2.*[0;lm(2:Nb,1);0;lm(2:Nb,1)];%sur-diagonale
hl = spdiags([diag_inf diag diag_sup],[-1,0,1],2*Nb,2*Nb);
```

Représentation graphique de la chaîne Pour représenter la chaîne, on appelle la fonction `fun_trace`. Par exemple, dans le cas-test où on a 5 barres de longueur `[0.7, 0.5, 0.3, 0.2, 0.5]`, avec un point de fixation à $(0,0)$, l'autre à $(1,-1)$. On trace la chaîne dans sa position initiale donnée par `xy = [ 0.2, 0.4, 0.6, 0.8, -1.0, -1.5, -1.5, -1.3]` et on obtient le graphe suivant :

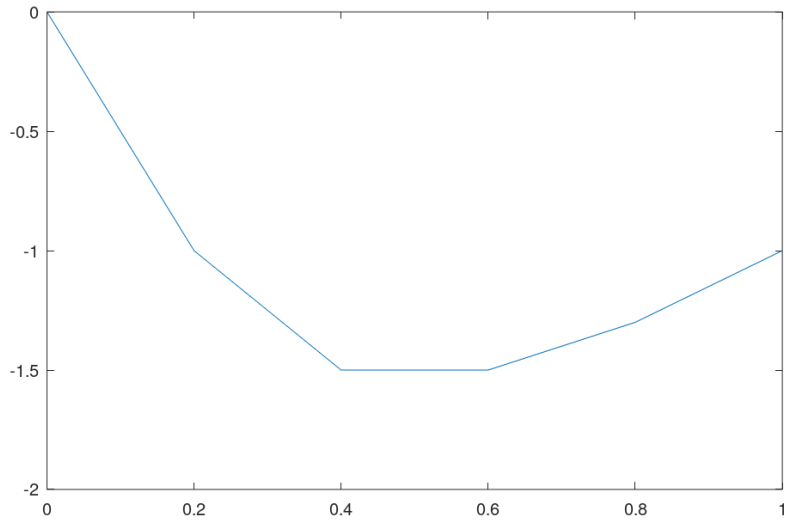


FIGURE 2 – Tracée de la position initiale de la chaîne dans le cas-test

L'indice de sortie La fonction verif_param s'occupe de vérifier la bonne conformité des paramètres et de déterminer l'indice de sortie de la fonction. Afin d'assurer le bon fonctionnement de la fonction `chs`, cette fonction est appelée en premier lieu.

Si les paramètres ne sont pas conformes, c'est-à-dire si $\text{indic\_sortie} = 1$, la fonction `chs` s'arrête instantannément et retourne des variables par défaut nulles. Si les paramètres sont conformes, c'est-à-dire si $\text{indic\_sortie} = 0$, alors le simulateur `chs` réalise le calcul demandé.

Les vérifications sur les paramètres sont
— `indic` $\in \{1, 2, 4, 5, 6\}$
— `xy` est un vecteur-colonne de taille $2N_n \times 1$
— `lm` est un vecteur-colonne de taille $N_b \times 1$

1.4 Vérification du code

1.4.1 Exactitude de l'énergie potentielle et des contraintes

On peut vérifier si notre programme calcule correctement les valeurs de e et c en calculant ces valeurs exactement pour quelques cas simples et comparer avec le résultat du programme.

Par exemple, une chaîne composées de 2 barres de longueur `L = [1, 1]`. Si un point fixé est $(0, 0)$ et l'autre est $(1, 0)$, un exercice simple de géometrie montre que les coordonnées y_i sont données par `y = [0, -sqrt(3)/2, 0]` (on sélectionne le cas où le point intermédiaire est plus bas que les points de fixation, par soucis de cohérence). Alors l'énergie devient

$$
\begin{aligned}
e &= L_1 \frac{y_1 + y_0}{2} + L_2 \frac{y_2 + y_1}{2} \\
&= y_1 = -\frac{\sqrt{3}}{2}
\end{aligned}
$$

Maintenant on exécute `ch` avec les paramètres correspondant à cet exemple, c'est-à-dire, `A = 1, B = 0,L = [1, 1]` et `xy = [0.5, -sqrt(3)/2]`). On obtient la valeur $e = -0.86603$ et le graphe suivant :

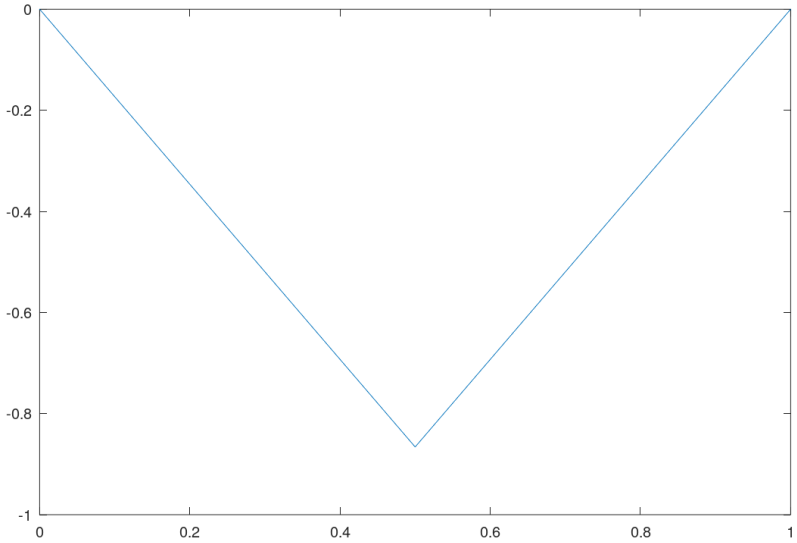


FIGURE 3 – Traçage de la chaîne composée seulement de deux barres de longueur unitaire

Or on vérifie que `>> -sqrt(3)/2 = -0.86603`, donc notre programme a bien calculé l'énergie.

De plus, on note que dans ce cas on a calculé la solution exacte du problème directement, les contraintes sont donc nulles. Le programme nous renvoie la valeur `-1.1102e-16` pour les deux contraintes, une valeur que l'on peut bien considérer nulle.

1.4.2 Exactitude des dérivées

On vérifie maintenant l'exactitude du calcul du gradient `g` de l'énergie `e` en utilisant les différences finies.

On compare les composantes du gradient avec le quotient différentiel correspondant :

$$
\frac{\phi(x + t_i e^i) - \phi(x)}{t_i} = \phi'(x).e^i + O(t_i)
$$

avec e^i le i-ème vecteur de base de $\mathbb{R}^n$ et $t^i = \epsilon^{1/2} \max(\tau_i, |x_i|)$ (ϵ est l'*epsilon-machine*).

Cette étude de l'exactitude du gradient est implémentée dans la fonction `test_derivees` et affiche le résultat suivant dans le cas-test défini dans la partie 1.5.

```
indice : 1  pas : 1.49e-08  composante : 0.000000e+00  DF : 0.000000e+00  erreur : 0.000000e+00  (erreur abs)
indice : 2  pas : 1.49e-08  composante : 0.000000e+00  DF : 0.000000e+00  erreur : 0.000000e+00  (erreur abs)
indice : 3  pas : 1.49e-08  composante : 0.000000e+00  DF : 0.000000e+00  erreur : 0.000000e+00  (erreur abs)
indice : 4  pas : 1.49e-08  composante : 0.000000e+00  DF : 0.000000e+00  erreur : 0.000000e+00  (erreur abs)
indice : 5  pas : 1.49e-08  composante : 6.000000e-01  DF : 6.000000e-01  erreur : 9.934107e-09  (erreur rel)
indice : 6  pas : 2.24e-08  composante : 4.000000e-01  DF : 4.000000e-01  erreur : 9.934108e-09  (erreur rel)
indice : 7  pas : 2.24e-08  composante : 2.500000e-01  DF : 2.500000e-01  erreur : 0.000000e+00  (erreur rel)
indice : 8  pas : 1.94e-08  composante : 3.500000e-01  DF : 3.500000e-01  erreur : 3.667978e-08  (erreur rel)
```

Remarque : Si la valeur de la i-ème composante du gradient est non nulle, alors on calcule l'erreur relative par rapport au quotient différentiel correspondant. Sinon, on se contente du calcul de l'erreur absolue.

Dans notre cas, l'énergie ne dépendant pas des coordonnées $(x_i)_{i=1}^{N_n}$, le gradient est nul sur les N_n-ème premières coordonnées et donc l'erreur calculée est absolue. De plus, cette non dépendance de l'énergie en $(x_i)_{i=1}^{N_n}$ entraîne la nullité des quotients différentiels correspondant et l'erreur absolue est donc nulle.

1.5 Application à un cas-test

Reprenons le cas test utilisé pour créer la représentation graphique 1.3.1 : on a 5 barres de longueur `[0.7, 0.5, 0.3, 0.2, 0.5]`, avec un point de fixation à $(0, 0)$, l'autre à $(1, -1)$, la position initiale de la chaîne étant donnée par `xy = [ 0.2, 0.4, 0.6, 0.8, -1.0, -1.5, -1.5, -1.3]`. On obtient les grandeurs calculées suivantes :

— `e =  -2.28`

— `c =  [0.55,0.04,-0.05,0.04,-0.12]'`

— `g =`

```
0.00000
0.00000
0.00000
0.00000
0.60000
0.40000
0.25000
0.35000
```

— `a =`

```
 0.40000   0.00000   0.00000   0.00000  -2.00000   0.00000   0.00000   0.00000
-0.40000   0.40000   0.00000   0.00000   2.00000  -1.00000   0.00000   0.00000
 0.00000  -0.40000   0.40000   0.00000   0.00000   1.00000   0.00000   0.00000
 0.00000   0.00000  -0.40000   0.40000   0.00000   0.00000   0.00000   0.40000
 0.00000   0.00000   0.00000  -0.40000   0.00000   0.00000   0.00000  -0.40000
```

En observant le vecteur des contraintes, on note que cette chaîne n'est pas admissible car les contraintes ne sont pas nulles.

2 Résolution locale par l'algorithme de Newton et écriture de l'optimiseur

Dans cette partie, on présente une méthode de résolution du problème par l'algorithme de Newton. On va coder un *optimiseur* utilisant les grandeurs calculées par le *simulateur*.

On va étudier 4 cas-tests, où on discute la convergence de l'algorithme, sa vitesse, et si la chaîne trouvée correspond à un point stationnaire minimiseur du problème. On discute aussi théoriquement comment déterminer la vitesse de convergence de notre algorithme (dans les cas convergents).

2.1 La méthode de Newton

Pour trouver un point stationnaire, on utilise les conditions de première ordre d'optimalité, aussi appellées les conditions KKT :

$$\begin{cases} \nabla f\left(x_*\right) + c'\left(x_*\right)^\top \lambda_* = 0 \\ c\left(x_*\right) = 0 \end{cases}$$

Dans notre cas, $f = e$, l'énergie potentielle de la chaîne. La première équation correspond à trouver un zéro du gradient du lagrangien du problème. Si on laisse $z = (x, \lambda)$, on peut écrire ce système comme $F(z) = 0$ en notant

$$F(z) := \begin{pmatrix} \nabla f(x) + c'(x)^\top \lambda \\ c(x) \end{pmatrix}$$

C'est un problème que l'algorithme de Newton est en mesure de résoudre. Pour cela, on doit calculer la dérivée de F :

$$F'(z) = \begin{pmatrix} \nabla^2_{xx} \ell(x, \lambda) & c'(x)^\top \\ c'(x) & 0 \end{pmatrix}$$

Donc, selon la théorie vue en cours, une itération de l'algorithme de Newton consiste à calculer la solution du système linéaire

$$\begin{pmatrix} L_k & A_k^\top \\ A_k & 0 \end{pmatrix} \begin{pmatrix} d_k \\ \mu_k \end{pmatrix} = - \begin{pmatrix} \nabla f\left(x_k\right) + A_k^\top \lambda_k \\ c\left(x_k\right) \end{pmatrix}$$

Cependant, on peut formuler ce système de manière équivalente et plus simple si l'on pose $\lambda_k^{\mathrm{PQ}} := \lambda_k + \mu_k$:

$$\begin{pmatrix} L_k & A_k^\top \\ A_k & 0 \end{pmatrix} \begin{pmatrix} d_k \\ \lambda_k^{\mathrm{PQ}} \end{pmatrix} = - \begin{pmatrix} \nabla f\left(x_k\right) \\ c\left(x_k\right) \end{pmatrix}$$

et donc le point z est mis à jour selon $x_{k+1} := x_k + d_k$ et $\lambda_{k+1} := \lambda_k^{\mathrm{PQ}}$.

C'est ce système que l'on a implémenté dans notre code, en observant que le simulateur `chs` nous permet de calculer L_k, A_k, $c(x_k)$, $\nabla f(x_k)$. Donc, la seule chose que l'optimiseur doit réaliser est une boucle, qui à chaque itération résout ce système avec les données que le simulateur lui

donne, et mettre à jour le point en conséquence.

Une propriété intéressante de l'algorithme de Newton est la convergence quadratique locale si les hypothèses de régularité suivantes sont respectées :

— $z_\star \in \mathbb{R}^n \times \mathbb{R}^m$ est tel que $F(z_\star) = 0$
— F est $\mathcal{C}^{1,1}$ dans un voisinage de $z_\star$
— $F'(z_\star)$ est inversible

Alors il existe un voisinage $\mathcal{V}$ de $z_\star$ tel que si l'itéré initial $z_1 \in \mathcal{V}$, alors l'algorithme de Newton est bien défini et génère une suite $(z_k)_k$ qui converge quadratiquement vers $z_\star$.

La première composante du point $z_\star = (x_\star, \lambda_\star)$ trouvé par la méthode de Newton est un point stationnaire de f. On doit ensuite étudier la nature de ce point : maximum, minimum ou point selle. On étudiera la matrice hessienne réduite.

Un point important à noter sur l'algorithme de Newton est que sa convergence est uniquement locale. Ainsi, si le point initial n'est pas dans un voisinage d'une solution de $F(z) = 0$, alors l'algorithme peut diverger. On verra dans la partie 3 une globalisation de l'algorithme de Newton afin de forcer la convergence quelque soit le point initial.

2.2 Implémentation du code

On ajoute deux fonctions à notre code :
— `sqp` est l'optimiseur qui calcule la solution du problème primal-dual : `x` et `lm`.
— La fonction `verif_param_sqp` vérifie les arguments donnés à la fonction `sqp`.

2.2.1 Optimiseur sqp

L'optimiseur est indépendant du problème et peut donc être utilisé pour d'autres problèmes. Voici la documentation de la fonction optimiseur `sqp` :

```
function [x,lm,info] = sqp(simul,x,lm,options)
```

Paramètres d'entrée

— `simul` : simulateur utilisé dans l'optimiseur
— `x` : vecteur-colonne des abscisses puis des ordonnées des points d'articulation de la chaîne, $\mathtt{xy} = (\mathtt{x}_1, ..., \mathtt{x}_{n_n}, \mathtt{y}_1, ..., \mathtt{y}_{n_n})$. Il est de dimension $(2n_n, 1)$.
— `lm` : vecteur-colonne des multiplicateurs de Lagrange pour les contraintes d'égalité $\lambda = \{\lambda_i\}_{i=1}^{n_b}$. Il est de dimension $(n_b, 1)$.
— `options` : structure spécifiant les paramètres de fonctionnement de l'algorithme, plus précisément, ils définissent le critère d'arrêt de l'algorithme de Newton.
  — `options.tol(1)` et `options.tol(2)` : les seuils de tolérance de $\|\nabla_x l(x_k, \lambda_k)\|_\infty$ et $\|c(x_k)\|_\infty$. Ils sont compris dans $]0, 1[$.
  — `options.maxit` : le nombre maximal d'itérations autorisées. C'est un entier positif.

Valeurs renvoyées

— `x` et `lm` : les vecteurs contenant les valeurs finales de x_k et λ_k.
— `info` : structure donnant les informations sur le comportement de l'optimiseur suivantes :
  — `info.status` : décrit le résultat de l'algorithme.
    — 0 si le seuil d'optimalité est atteint ie l'algorithme a convergé.
    — 1 si les paramètres donnés à `sqp` ont un mauvais format.
    — 2 si le nombre maximal d'itérations autorisées est atteint ie il y a non convergence de l'algorithme.
  — `info.niter` : le nombre d'itérations effectuées.

2.3 Analyse théorique

Dans cette partie, on répond aux questions théoriques posées dans l'énoncé. La première concerne l'analyse de vitesse de convergence et la seconde concerne une estimation pour la valeur initiale des multiplicateurs de Lagrange.

2.3.1 Détermination pratique de la vitesse de convergence.

D'abord, on rappelle que l'expression de F' est donnée par

$$F'(z) = \begin{pmatrix} \nabla^2_{xx}\ell(z) & c'(x)^\top \\ c'(x) & 0 \end{pmatrix}$$

Or, comme l'énergie potentielle et les contraintes sont des polynômes en les variables spatiales, le lagrangien est deux fois différentiable avec dérivées continues, ce qui implique que la hessienne est symétrique (car les dérivées de seconde ordre commutent). Ainsi, F' est symétrique. Par le théorème spectral, on a une base orthonormée de vecteurs propres de F' (à chaque point). En particulier, comme on est dans le cadre de l'algorithme de Newton, on peut assumer que $F'(z_*)$ est inversible, et donc toutes les valeurs absolues des valeurs propres sont strictement positives.

D'un autre côté, on note que, étant données les hypothèses de régularité, il existe un voisinage de z_* où on peut approcher F par son développement en Taylor autour de z_* :

$$F(z_k) \approx \underbrace{F(z_*)}_{=0} + F'(z_*)(z_k - z_*) = F'(z_*)(z_k - z_*)$$

Le théorème spectral nous donne une matrice orthogonale P de changement de base qui nous permet d'exprimer le vecteur $z_k - z_*$ dans la base de vecteurs propres de $F'(z_*)$:

$$v := P(z_k - z_*)$$

Ainsi, si D est la matrice diagonale avec les valeurs propres de $F'(z_*)$, le développement en Taylor d'avant, devient

$$F(z_k) = P^\top D P(z_k - z_*)$$

et on peut multiplier par P (l'inverse de $P^\top$) à gauche dans les deux côtés pour obtenir :

$$PF(z_k) = DP(z_k - z_*) = Dv$$

Donc, si on prend la norme euclidienne au carré de cette expression, on obtient

$$\|PF(z_k)\|^2 = \|Dv\|^2 = \sum_i |\lambda_i|^2 v_i^2$$

Cependant, comme P transforme une base orthonormée (la base usuelle de $\mathbb{R}^n$) en autre base orthonormée (la base spectrale), P préserve les normes :

$$\|PF(z_k)\| = \|F(z_k)\|$$

et

$$\|P(z_k - z_*)\| = \|v\| = \|z_k - z_*\|$$

donc

$$\|F(z_k)\|^2 = \|Dv\|^2 = \sum_i |\lambda_i|^2 v_i^2 \geqslant |\lambda_1|^2 \|v\|^2 = |\lambda_1|^2 \|z_k - z_*\|^2$$

où λ_1 est la valeur propre avec la plus petite valeur absolue non nulle. Or, on peut prendre la racine carrée et obtenir

$$\|F(z_k)\| \geqslant |\lambda_1| \|z_k - z_*\|$$

De même manière, on obtient aussi la borne

$$\|F(z_k)\| \leqslant |\lambda_n| \|z_k - z_*\|$$

où λ_n est la valeur propre avec la plus grande valeur absolue.

Avec ces observations, on peut montrer que si l'on connaît la vitesse de convergence de $F(z_k)$, alors on connaît la vitesse de convergence de z_k. Supposons que $F(z_k)$ a une vitesse de convergence p vers $F(z_*) = 0$:

$$\exists C > 0 \text{ tel que } \forall k,\ \|F(z_{k+1})\| \leqslant C \|F(z_k)\|^p$$

alors on peut minorer le terme à gauche, et majorer le terme à droite en utilisant les inégalités obtenues précédemment, et donc

$$|\lambda_1| \|z_{k+1} - z_*\| \leqslant \|F(z_{k+1})\| \leqslant C \|F(z_k)\|^p \leqslant C |\lambda_n|^p \|z_k - z_*\|^p$$

c'est-à-dire,

$$\|z_{k+1} - z_*\| \leqslant K \|z_k - z_*\|^p$$

où $K = \frac{C|\lambda_n|^p}{|\lambda_1|}$ est une constante ne dépendant pas de k.

Donc, ce que l'on vient de montrer est que si on vérifie que la vitesse de convergence de $F(z_k)$ est d'ordre p, alors la vitesse de convergence de z_k est aussi d'ordre p.

Comme toutes les normes pour l'espace de dimension finie sont équivalentes, on peut utiliser n'importe quelle norme pour vérifier l'ordre de convergence de $F(z_k)$, et c'est parfois pratique d'utiliser la norme $\|\cdot\|_\infty$.

2.3.2 Estimation des multiplicateurs initiaux

Dans le cas où on donne un point stationnaire (c'est-à-dire, un point où $\nabla\ell(z) = 0$) au programme, le multiplicateur de Lagrange est caractérisé comme la solution du système linéaire

$$c'(x_1)^\top \lambda_1 = -\nabla f(x_1)$$

Donc il est naturel d'utiliser ce même système pour estimer un λ_1, même dans le cas où on n'est pas dans un point stationnaire. On rappelle que dans notre cas, $f = e$, l'énergie potentielle de la chaîne.

2.4 Cas-tests

On utilise notre programme pour résoudre les cas-tests donnés dans l'énoncé. On utilise le critère d'optimalité suivant :

$$\|\nabla_x \ell(x_k, \lambda_k)\|_\infty < options.tol(1) = 10^{-8}$$
$$\|c(x_k)\|_\infty < options.tol(2) = 10^{-8}$$

avec un critère de nombre d'itérations maximales atteint (10.000). Comme l'algorithme de Newton a une convergence très rapide dans les bons cas, ce nombre maximal d'itérations est suffisament grand. Si l'algorithme s'arrête après avoir atteint le nombre maximal d'itérations, alors on conclut une non-convergence.

2.4.1 Cas 2a

On utilise les positions initiales $(0.2, -1), (0.4, -1.5), (0.6, -1.5),$ et $(0.8, -1.3)$ pour les noeuds intérieurs de la chaîne. Observer le graphe de cette chaîne initiale va nous aider, à chaque cas-test, à comprendre le résultat final obtenu. Dans ce premier cas, le graphe est le suivant :

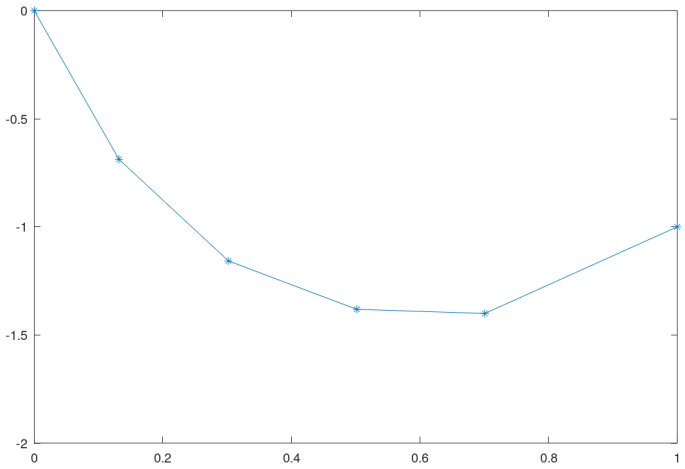


FIGURE 4 – Position initiale de la chaîne dans le cas-test 2a

L'algorithme **converge** presque instantanément, il prend juste **6 itérations**, et on obtient la solution suivante :

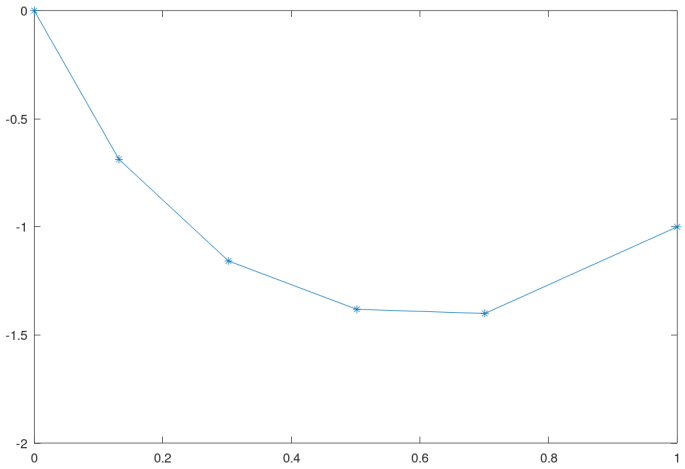


FIGURE 5 – Position finale de la chaîne dans le cas-test 2a

Ce résultat est attendu : la position initiale de la chaîne est très proche de la représentation physique et exacte d'une chaîne en équilibre. C'est-à-dire qu'on commence déjà très proche de la solution.

On obtient les valeurs des positions et des multiplicateurs suivantes :

— x =

```
 0.1317
 0.3020
 0.5017
 0.7008
-0.6875
-1.1576
-1.3815
-1.4006
```

```
   — lm =

 0.9261
 0.7162
 0.6107
 0.6126
 0.4076
```

On explique maintenant comment vérifier dans les cas-tests où l'algorithme converge si la chaîne trouvée est un minimiseur pour le problème, ou sinon un point selle ou un point maximiseur. Une condition suffisante pour l'optimalité s'écrit

$$\langle \nabla^2_{xx} \ell(x_*, \lambda_*) d, d \rangle > 0 \quad \forall d \in T_{x_*} X_E \setminus \{0\}$$

Pour des contraintes qualifiées, le cône tangent coïncide avec le cône linéarisé. Donc on peut donner une caractérisation alternative pour cette condition : si N est une matrice avec des colonnes qui sont une base de $\ker(c'(x_*))$, alors $\forall d \in T_{x_*} X_E, \exists v \in \mathbb{R}^n$ tel que $d = Nv$. La condition devient donc

$$\langle N^\top \nabla^2_{xx} \ell(x_*, \lambda_*) N v, v \rangle > 0 \quad \forall v \in \mathbb{R}^n$$

C'est-à-dire, on a transformé un problème de vérification de la positivité définie d'une matrice sur un cône tangent en un problème de vérification de la positivité définie d'une autre matrice sur tout l'espace, ce qui est un problème plus facile.

Pour ce cas-test, on obtient le spectre suivant pour $N^\top \nabla^2 \ell(x_*, \lambda_*) N$:

$$(\lambda_1, \lambda_2, \lambda_3) = (4.54112, 0.75595, 2.53282)$$

Comme elles sont toutes positives, on conclut par la condition suffisante précédente que la chaîne finale trouvée est un **minimiseur** pour l'énergie potentielle.

Pour faire l'analyse de vitesse de convergence pour ce cas (et pour les autres cas de convergence), on note que si l'ordre de convergence est quadratique, alors par l'analyse théorique d'avant et par la définition d'ordre de convergence, il existe une constante $C > 0$ telle que pour tout $k \geqslant 1$

$$\frac{\|F(z_{k+1})\|_\infty}{\|F(z_k)\|^2_\infty} \leqslant C$$

Dans le code, le rapport à gauche est facile à calculer pour chaque itération. Donc pour trouver la constante C, on prend la valeur maximale de ce rapport pour toutes itérations. Pour ce cas test on

trouve la valeur $C = 4.2363$, c'est-à-dire, on peut dire que la vitesse de convergence est **quadratique**.

Une question intéressante est de voir si l'on est "trop loin" d'une convergence cubique, c'est-à-dire, quelle est la constante que l'on trouve si la puissance dans le dénominateur du rapport est 3 au lieu de 2 ? Étonnamment, si l'on fait ça on obtient la valeur 2771459 ! Ça veut dire que l'on est bien loin d'une convergence cubique.

2.4.2 Cas 2b

On utilise les positions initiales $(0.2, 1), (0.4, 1.5), (0.6, 1.5),$ et $(0.8, 1.3)$ pour les noeuds intérieurs de la chaîne. Le graphe de la chaîne initiale est le suivant (observons que la chaîne initiale ne satisfait pas nécessairement les contraintes de longueur des barres) :

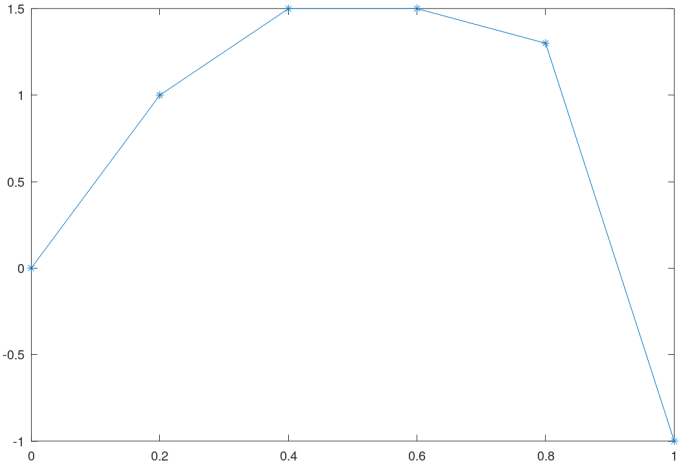


FIGURE 6 – Position initiale de la chaîne dans le cas-test 2b

L'algorithme **converge** presque instantanément, il prend juste **12 itérations**, et on obtient la solution suivante :

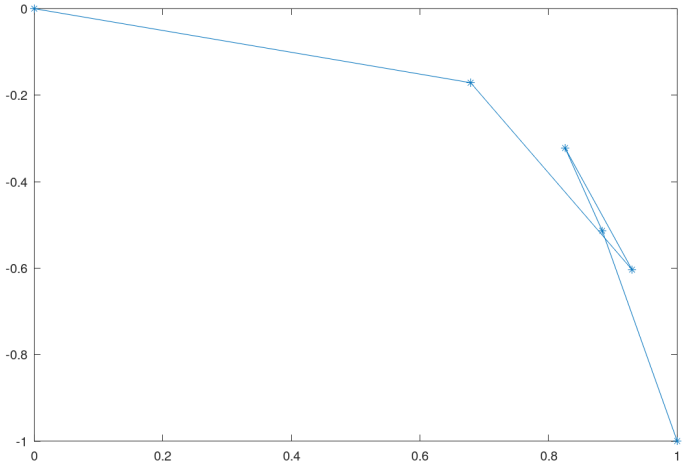


FIGURE 7 – Position finale de la chaîne dans le cas-test 2b

Le résultat final peut sembler un peu étrange de prime abord, mais après quelques instants d'observation, on note qu'il est raisonnable : la chaîne forme une espèce de "noeud" et elle ne change pas à cause des restrictions de longueur.

On obtient les valeurs des positions et des multiplicateurs suivantes :

— `x =`

```
 0.6787
 0.9298
 0.8257
 0.8834
-0.1713
-0.6036
-0.3223
-0.5138
```

— `lm =`

```
-0.3007
-0.8129
 1.9602
-3.5331
-1.7512
```

En faisant la même analyse du spectre de la hessienne réduite qui a été décrite dans le cas précédent, on obtient le spectre suivant :

$$(\lambda_1, \lambda_2, \lambda_3) = (4.9446, -1.8334, -15.0153)$$

Comme une valeur propre est positive et les autres sont négatives, on conclut que la chaîne finale trouvée par l'algorithme est un **point selle** pour l'énergie potentielle. Cela est attendu, car la représentation graphique de la configuration finale de la chaîne a un "noeud", que si on pouvait

défaire, alors la chaîne évoluerait vers une configuration d'énergie plus baisse.

L'analyse de vitesse de convergence est faite comme dans le cas 2a : on trouve la valeur maximale pour le rapport avec la puissance quadratique (2) dans le dénominateur. On obtient la constante 2.6858, ce qui indique que l'on a bien la convergence **quadratique** (si l'on essaie une convergence cubique la constante devient 11166 !).

2.4.3 Cas 2c

On utilise les positions initiales $(0.2, -1), (0.4, -1.5), (0.6, 1.5),$ et $(0.8, -1.3)$ pour les noeuds intérieurs de la chaîne. Le graphe de la chaîne initiale est le suivant :

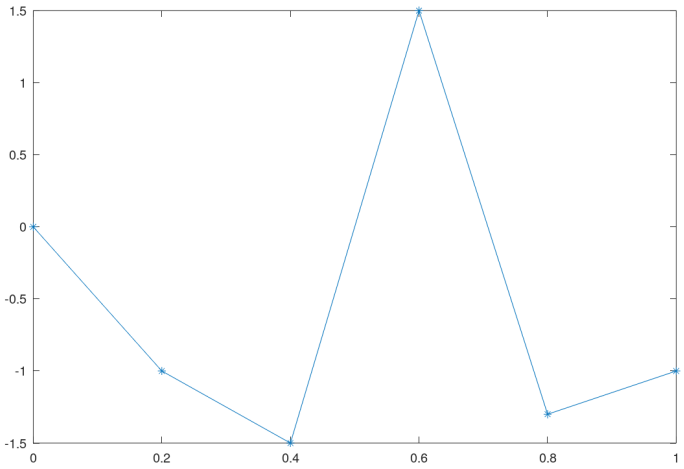


FIGURE 8 – Position initiale de la chaîne dans le cas-test 2c

L'algorithme **converge** presque instantanément, il prend juste **12 itérations**, et on obtient la solution suivante :

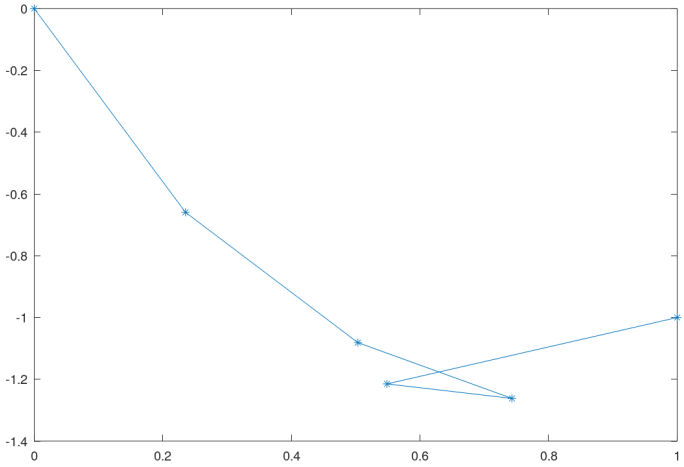


FIGURE 9 – Position finale de la chaîne dans le cas-test 2c

On peut utiliser la même explication du cas précédent pour justifier que ce résultat semble correct.

On obtient les valeurs des positions et des multiplicateurs suivantes :

— `x =`

```
  0.2352
  0.5034
  0.7430
  0.5486
 -0.6593
 -1.0813
 -1.2618
 -1.2149
```

— `lm =`

```
  1.0372
  0.9096
  1.0182
 -1.2548
  0.5404
```

En faisant la même analyse du spectre de la hessienne réduite qui a été décrite dans le cas précédent, on obtient le spectre suivant :

$$(\lambda_1, \lambda_2, \lambda_3) = (-2.1342, 2.0609, 5.7625)$$

Comme une valeur propre est négative, et les autres sont positives, on conclut que la chaîne finale trouvée par l'algorithme est un **point selle** pour l'énergie potentielle. Cela est attendu pour la même raison que dans le cas précédent.

L'analyse de la vitesse de convergence est faite exactement comme avant, et nous donne $C = 14.140$ pour une convergence quadratique. Ça semble un peu grand. Si on essaie une convergence plus faible, d'ordre linéaire au lieu de quadratique, on obtient une constante $C = 3.1378$, ce qui semble plus raisonnable. C'est-à-dire, on croît que pour ce cas, on a convergence, cependant une convergence plus lente d'ordre **linéaire**.

2.4.4 Cas 2d

On utilise les positions initiales $(0.2, 1), (0.4, -1.2), (0.6, 1.5),$ et $(0.8, -1.3)$ pour les noeuds intérieurs de la chaîne. Le graphe de la chaîne initiale est le suivant :

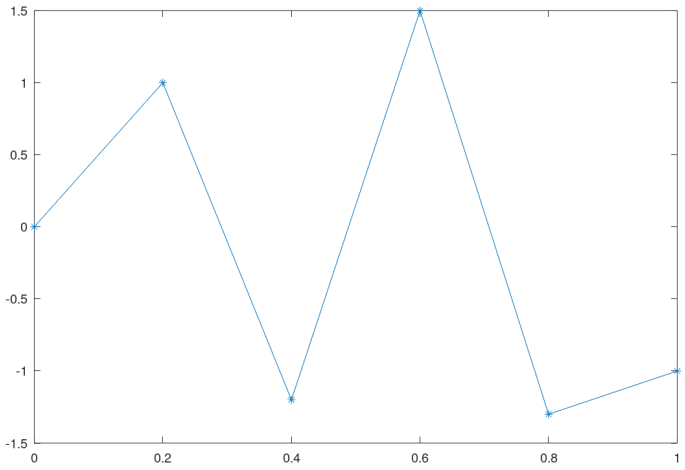
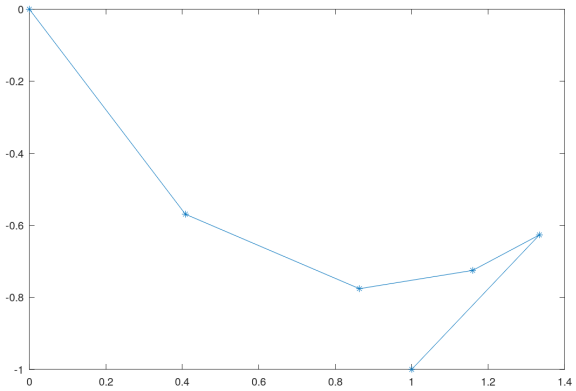
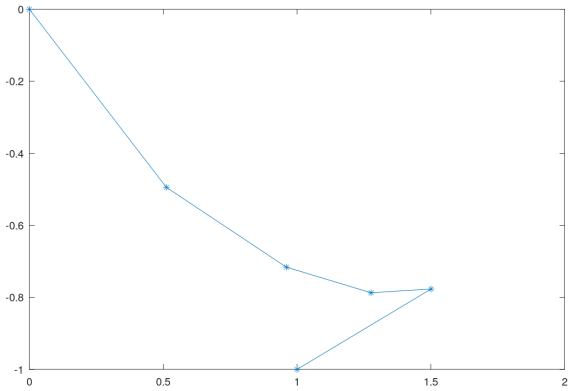


FIGURE 10 – Position initiale de la chaîne dans le cas-test 2d

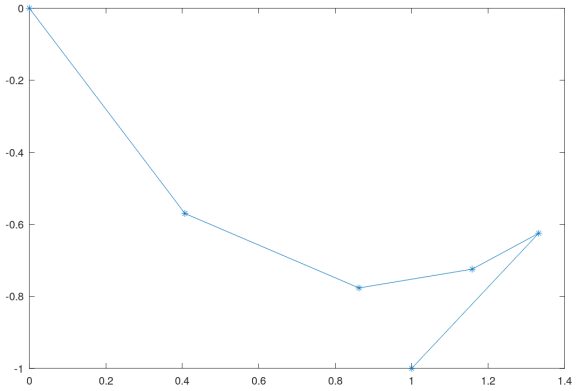
L'algorithme **ne converge pas**. Cependant, on note que pour un nombre maximal d'itérations $n_{\max} = 1000, 10.000, 20.000$, l'optimiseur s'approche d'une chaîne avec le même aspect, que l'on représente ci-dessous :



(a) Position de la chaîne après 1.000 itérations dans le cas-test 2d



(b) Position de la chaîne après 10.000 itérations dans le cas-test 2d



(c) Position de la chaîne après 20.000 itérations dans le cas-test 2d

Cette allure est raisonnable comme une configuration d'équilibre pour la chaîne. On note aussi

que pour 10.000 itérations, les contraintes pour le résultat valent $c = 0.05$, un nombre relativement petit (on peut voir ça graphiquement, les tailles des barres semblent bonnes). On trouve donc que le problème est dans le gradient du lagrangien, qui ne diminue pas.

On obtient les valeurs des positions et des multiplicateurs suivantes (lorsque le nombre d'itérations maximales est 1000) :

— `x =`

```
  0.4080
  0.8632
  1.1594
  1.3340
 -0.5691
 -0.7759
 -0.7252
 -0.6266
```

— `lm =`

```
  0.7798
  0.6975
  1.0730
  1.8227
 -0.9520
```

3 Globalisation par la recherche linéaire et la règle d'Armijo

Ce dernier cas-test montre que l'algorithme de Newton peut ne pas converger, notamment si les conditions de régularité ne sont pas respectées et/ou si le point initial n'est pas dans un voisinage d'une solution. On va donc globaliser l'algorithme de Newton, c'est-à-dire forcer sa convergence quel que soit le point initial.

3.1 Globalisation de notre méthode de résolution

3.1.1 La recherche linéaire

A l'itération k, une fois une direction de descente d_k déterminée, on cherche un pas $\alpha_k > 0$ afin de faire décroître f le long de d_k. On appelle cela faire de la *recherche linéaire*. Il faut respecter 2 objectifs :

— f doit décroître suffisamment pour que l'algorithme converge, notamment en temps raisonnable. On doit donc déterminer $\alpha_k > 0$ tel que

$$f(x_k + \alpha_k d_k) \leq f(x_k) + \text{ "un terme négatif"}$$

— Le pas α_k ne doit pas être trop petit sinon la suite $(x_k)_k$ est une suite de Cauchy convergente vers un point qui peut ne pas être stationnaire. On parle de *fausse convergence*.

3.1.2 La règle d'Armijo

Sous l'hypothèse que f est de classe C^1 (dans notre cas e est polynomiale donc est C^1), on peut écrire

$$f(x_k + \alpha_k d_k) \approx f(x_k) + \alpha_k \langle g_k, d_k \rangle$$

Or α_k étant strictement positif et $\langle g_k, d_k \rangle < 0$ car d_k est une direction de descente, on a $\alpha_k \langle g_k, d_k \rangle < 0$.

On voit donc que si on cherche α_k tel que le symbole $\approx$ est en fait $\leq$, alors on a $f(x_k + \alpha_k d_k) \leq f(x_k) +$ "un terme négatif" . Afin de rendre l'inégalité

$$f(x_k + \alpha_k d_k) \leq f(x_k) + \alpha_k \langle g_k, d_k \rangle$$

plus facilement réalisable, on multiplie le terme linéaire par une constante $\omega_1 \in]0, 1[$ très petite (dans notre cas $\omega_1 = 10^{-4}$). On cherche donc $\alpha_k > 0$ tel que

$$f(x_k + \alpha_k d_k) \leq f(x_k) + \omega_1 \alpha_k \langle g_k, d_k \rangle \qquad \text{(Armijo)}$$

On appelle cette condition la *condition d'Armijo* ou *condition de décroissance linéaire* car on fait décroître f au moins $\omega_1 \times$ "partie linéaire de f".

On montre dans la partie 3.2.2 que la condition (Armijo) est respectée pour un $\alpha_k > 0$ suffisamment petit. On a donc besoin d'un processus générant des pas de plus en plus petits et on sait qu'il générera sans aucun doute un $\alpha_k > 0$ satisfaisant (Armijo). Nous allons utiliser la *règle d'Armijo* ou la *règle de rebroussement* qui consiste à prendre $\alpha_k = \tau^{i_k} \in \{1, \tau, \tau^2, ...\}$ ($\tau \in]0, 1[$) avec i_k le plus petit entier tel que $\alpha_k = \tau^{i_k}$ vérifie (Armijo). On prend $\tau = \frac{1}{2}$.

On peut faire 2 premières remarques sur la règle d'Armijo :
- Etant donné que l'on teste les pas de manière décroissante, la règle d'Armijo choisit le pas le plus grand possible. Ainsi, la condition de recherche linéaire demandant un pas non trop petit semble être respectée dans la plupart des cas. Cependant, il n'y a pas de limite inférieure à la décroissance des τ^i et la suite des $(\tau^i)_i$ peut décroître très rapidement si τ est petit. Pour résoudre ce problème, on peut utiliser une autre règle : la *règle de Goldstein*.
- La règle d'Armijo est un processus simple et n'utilise donc pas toutes les informations à portée de main. Il est possible de tenir compte des valeurs de f en les points intermédiaires $x_k + \tau_i d_k$ et de procéder à une interpolation. Cela mène à la méthode de recherche linéaire appelée *pas d'Armijo*. Dans notre cas, nous n'utilisons pas cette amélioration de la règle d'Armijo.

3.1.3 Globalisation

On définit une *fonction de mérite* comme étant une fonction réelle qui atteint un minimum (si possible global) en une solution du problème à résoudre. Dans notre cas, on choisit la *fonction de moindres-carrés*

$$\varphi : \mathbb{R}^{n+m} \to \mathbb{R}$$

définie en $z = (x, \lambda)$ par

$$\varphi(z) := \frac{1}{2}\|F(z)\|_2^2$$

C'est bien une fonction de mérite : si $F(z) = 0$, alors $\varphi(z) = 0$ car $\|.\|_2$ est une norme. On remarque d'ailleurs que la réciproque est vraie.

On montre dans la partie 3.2.1 que la direction $p_k := (d_k, \mu_k) \in \mathbb{R}^{n+m}$ est une direction de descente en $z_k := (x_k, \lambda_k)$ de la fonction φ. On rappelle que $p_k := (d_k, \mu_k)$ est solution du système linéaire

$$\begin{pmatrix} L_k & A_k^\top \\ A_k & 0 \end{pmatrix} \begin{pmatrix} d_k \\ \mu_k \end{pmatrix} = - \begin{pmatrix} \nabla f(x_k) + A_k^\top \lambda_k \\ c(x_k) \end{pmatrix}$$

de la partie 2.1.

On applique maintenant la recherche linéaire par la règle d'Armijo pour trouver un pas $\alpha_k > 0$, puis on met à jour z_k par

$$x_{k+1} := x_k + \alpha_k d_k \qquad \text{et} \qquad \lambda_k := \lambda_k + \alpha_k \mu_k$$

On peut maintenant se demander si cet algorithme avec recherche linéaire d'Armijo possède bien la propriété de convergence globale. Tout comme pour la propriété de convergence locale de l'algorithme de Newton, on a besoin de conditions supplémentaires.

Si
— $\mathcal{K}_2(F'(z_k)) := \|F'(z_k)^{-1}\|_2 \|F'(z_k)\|_2$ est borné
— la condition de Zoutendijk est vérifiée : $\sum_{k\geq 1} \|\nabla\varphi(z_k)\|^2 \cos^2\theta_k < +\infty$ avec $\cos\theta_k = \frac{\langle -g_k, d_k\rangle}{\|g_k\|\|d_k\|}$

alors

— $\nabla\varphi(z_k) \to 0$ où φ est la méthode des moindres-carrés
— si, de plus, $(F'(z_k)^{-1})_k$ est bornée, alors $F(z_k) \to 0$

D'après ce théorème, on n'a donc pas forcément une convergence globale de notre algorithme avec règle d'Armijo. Il est cependant plus robuste par rapport à l'itéré initial que l'algorithme de Newton (lorsque les hypothèses sont vérifiées). C'est ce que l'on va observer dans l'étude des cas-tests.

3.2 Analyse théorique

3.2.1 Direction de descente de φ

On veut montrer que la direction $p_k := (d_k, \mu_k) \in \mathbb{R}^{n+m}$ est une direction de descente en $z_k := (x_k, \lambda_k)$ de la fonction φ.

On sait que p_k est par définition la solution du système linéaire $F'(z_k)p_k = -F(z_k)$. D'autre part, on a que $\nabla_z \varphi(z) = F(z)^\top \nabla_z F(z)$. En effet, si on note $F(x) = F(x_1, ..., x_m) = (F_1(x), ..., F_n(x))$

$$\varphi(x) = \frac{1}{2}\left[F_1(x)^2 + ... + F_n(x)^2\right]$$

$$\begin{aligned}
\nabla_x \varphi(x) &= \begin{pmatrix} \nabla_{x_1} F_1(x) F_1(x) & + ... + & \nabla_{x_1} F_n(x) F_n(x) \\ \vdots & & \vdots \\ \vdots & & \vdots \\ \nabla_{x_m} F_1(x) F_1(x) & + ... + & \nabla_{x_m} F_n(x) F_n(x) \end{pmatrix} \\
&= \begin{pmatrix} \nabla_{x_1} F(x) \cdot F(x) \\ \vdots \\ \nabla_{x_m} F(x) \cdot F(x) \end{pmatrix} \\
&= \left(\nabla_{x_1} F(x), ..., \nabla_{x_m} F(x)\right) F(x) \\
&= \nabla_x F(x)^\top F(x)
\end{aligned}$$

On a alors

$$\begin{aligned}
\nabla \varphi(z_k) \cdot p_k &= \nabla F(z_k)^\top F(z_k) \cdot p_k \\
&= F(z_k)^\top \nabla F(z_k) p_k \\
&= -F(z_k)^\top F(z_k) \quad \text{par construction de } p_k \\
&= -\|F(z_k)\|_2^2 \\
&< 0 \quad \text{si } F(z_k) \neq 0
\end{aligned}$$

Or on a toujours $F(z_k) \neq 0$, sinon l'algorithme se serait arrêté au moins à l'itération $k-1$ (voire avant).

On a bien montré que p_k est une direction de descente de φ en z_k.

3.2.2 Réalisation de la règle d'Armijo

Une fois la direction de descente p_k calculée, il faut déterminer le pas $\alpha_k > 0$ respectant la condition (Armijo).

On remarque que pour $i_k = 0$ ie $\alpha_k = 1$, la méthode précédente revient à l'algorithme de Newton appliqué à la fonction φ. Or on sait que sous des hypothèses de régularité, cet algorithme converge localement vers une solution de $\varphi(z) = 0$, donc converge vers un point stationnaire de f. Ainsi, on teste en premier le cas $i_k = 0$ car cela peut entraîner une convergence locale de l'algorithme vers un point stationnaire de f, à condition que le point initial se trouve dans un voisinage d'une solution de $\varphi(z) = 0$ et que les hypothèses de régularité sont respectées. D'autre part, on rappelle que l'algorithme avec la règle d'Armijo ne nous donne qu'un point stationnaire de φ, ce qui est un résultat moins fort. En effet, une solution de $F(z) = 0$ est un point stationnaire de φ (un minimiseur global de φ), alors qu'un point stationnaire de φ n'est pas forcément une solution

de $F(z) = 0$.

On se demande maintenant s'il existe $\alpha_k > 0$ tel que

$$f(x_k + \alpha_k d_k) \leq f(x_k) + \omega_1 \alpha_k \langle g_k, d_k \rangle$$

On va montrer qu'il en existe un suffisamment petit. Pour ce faire, on raisonne par l'absurde : supposons que $\forall \alpha_k > 0$, on a

$$f(x_k + \alpha_k d_k) > f(x_k) + \omega_1 \alpha_k \langle g_k, d_k \rangle$$

C'est équivalent à

$$\frac{f(x_k + \alpha_k d_k) - f(x_k)}{\alpha_k} > \omega_1 \langle g_k, d_k \rangle$$

Etant donné que $\alpha_k > 0$, on peut faire tendre α_k vers 0 et on a donc

$$\langle g_k, d_k \rangle \geq \omega_1 \langle g_k, d_k \rangle$$

C'est équivalent à $1 \leq \omega_1$. Or, $\omega_1 \in]0, 1[$, donc on aboutit à une contradiction.

On a bien montré qu'il existe $\alpha_k > 0$ tel que

$$f(x_k + \alpha_k d_k) \leq f(x_k) + \omega_1 \alpha_k \langle g_k, d_k \rangle$$

3.3 Implémentation du code

On ajoute l'option `options.rl` comme paramètre d'entrée de l'optimiseur `sqp` pour résoudre le problème avec ou sans recherche linéaire. La direction de descente p_k est donc différente selon la valeur de `options.rl`. Si `options.rl` $= 0$, alors on résout le problème avec recherche linéaire et si `options.rl` $= 1$ alors on le résout sans.

On ajoute une autre option utile à la vérification du code et à l'étude des cas-tests, que l'on présente dans la partie suivante.

3.4 Vérification du code

Pour que l'on sache ce qui se passe lors de l'optimisation, on affiche les valeurs de certaines grandeurs intéressantes. On ajoute une option `options.verb` qui caractérise l'affichage : 1 pour une étude simple (1 ligne par itération), 2 pour une étude poussée (autant de ligne que de α_k testés).

Pour `options.verb` $= 1$, on affiche les valeurs suivantes :

— `iter` : le numéro de l'itération k

— `|gl|` : la norme du gradient du lagrangien $\|\nabla_x \ell(x_k, \lambda_k)\|_\infty$

— `|ce|` : la norme des contraintes $\|c(x_k)\|_\infty$
On s'attend à ce que ces deux valeurs (`|gl|` et `|ce|`) tendent vers 0 pour respecter la condition d'optimalité pour l'arrêt de l'algorithme. D'ailleurs, si l'algorithme converge, on peut regarder si cette dernière est vérifiée en comparant `options.tol` à la dernière valeur de (`|gl|`,`|ce|`).

— `|x|` : la norme des noeuds $\|x_k\|_\infty$
— `|lm|` : la norme des multiplicateurs de Lagrange $\|\lambda_k\|_\infty$
Si l'algorithme converge, on s'attend à ce que les dernières valeurs de (`|x|`,`|lm|`) soient similaires.

— `alpha` : le pas α_k déterminé par la recherche linéaire
Ces valeurs doivent être de la forme $1/2^i$. De plus, dans le cas où l'algorithme converge vers une solution de $F(z) = 0$ respectant les conditions de régularité, les dernières valeurs de `alpha` doivent être égales à 1. En effet, dans ce cas, qu'importe le point initial, l'algorithme converge. Ainsi, il existe $k \in \mathbb{N}$ tel que l'itéré z_k appartient à un voisinage de la solution considérée. Or, on sait que l'algorithme de Newton converge localement (et on a supposé que les conditions de régularité sont respectées). Donc, à partir d'une certaine itération, dans la condition d'Armijo

$$\varphi(x_k + \alpha_k d_k) \leq \varphi(x_k) + \omega_1 \alpha_k \langle g_k, d_k \rangle$$

$$\iff \varphi(x_k + \alpha_k d_k) \leq \varphi(x_k) - \omega_1 \alpha_k \|F(z_k)\|_2^2$$

le terme $\|F(z_k)\|_2^2$ est négligeable et la condition d'Armijo devient

$$\varphi(x_k + \alpha_k d_k) \leq \varphi(x_k)$$

$$\iff \|F(z_k + \alpha_k d_k)\|_2^2 \leq \|F(z_k)\|_2^2$$

Ce qui est respectée lorsque $\alpha_k = 1$ car d_k est la direction de descente associée à l'algorithme de Newton sur $F(z) = 0$ et on a que les itérés $(z_j)_{j \geq k}$ convergent vers la solution considérée.

— `phi` : la valeur de $\varphi(z_k)$
L'algorithme convergent vers un point stationnaire de φ, on s'attend à ce que les valeurs de `phi` converge vers une valeur limite. On souhaite que cette valeur soit 0 car alors z_k tend vers un zéro de F.

Pour `options.verb` $= 2$, on affiche les valeurs suivantes :

— `iter` : le numéro du pas accepté, ie `iter` $= i_k$ tel que $\alpha_k = 1/2^{i_k}$ vérifie la condition d'Armijo à l'itération k.
— `simul` : le nombre d'appels au simulateur au début de la k-ième itération. On cherche à ce qu'il soit le plus petit possible car l'appel au simulateur est coûteux.
— `pente` : la valeur de la dérivée directionnelle $\varphi'(z_k) \cdot p_k$

Pour chaque numéro `iter` :

— `alpha` : le pas α_k considéré.
— `phip-phi` : la différence $\varphi(z_k + \alpha_k p_k) - (z_k)$.
— `DF(phi)` : la grandeur $\frac{\varphi(z_k+\alpha_k p_k)-(z_k)}{\alpha_k}$
On doit observer que les dernières valeurs de `phip-phi` sont négatives car cela est nécessaire pour que la condition d'Armijo soit respectée :

$$\varphi(x_k + \alpha_k d_k) - \varphi(x_k) \leq \omega_1 \alpha_k \langle g_k, d_k \rangle < 0$$

La valeur finale de `DF(phi)` est l'approximation par différences finies de $\varphi'(z_k) \cdot p_k$ s'il y a assez d'itérations dans la recherche de α_k, car alors les valeurs de α_k tendent vers 0. Cette situation n'est pas souhaitable car alors l'algorithme risque de procéder à une fausse convergence. Voici un exemple dans le cas-test 3c :

```
------------------------------------------------------------------------------
iter 1 ,   simul 0 ,       phi 1.19686e+01 ,      pente -2.39373e+01

Recherche Lineaire d'Armijo : |d| = 3.68e+01

  alpha          phip-phi        DF(phi)
1.0000e+00    1.30697e+05    1.30697e+05
5.0000e-01    8.14257e+03    1.62851e+04
2.5000e-01    4.98683e+02    1.99473e+03
1.2500e-01    2.71630e+01    2.17304e+02
6.2500e-02    2.46432e-02    3.94291e-01
3.1250e-02   -7.46157e-01   -2.38770e+01

|gl| = 2.937e+00 , |ce| = 3.622e+00
------------------------------------------------------------------------------
```

On a d'ailleurs que le nombre d'appels au simulateur `chs` est de 2 par recherche de pas α_k.

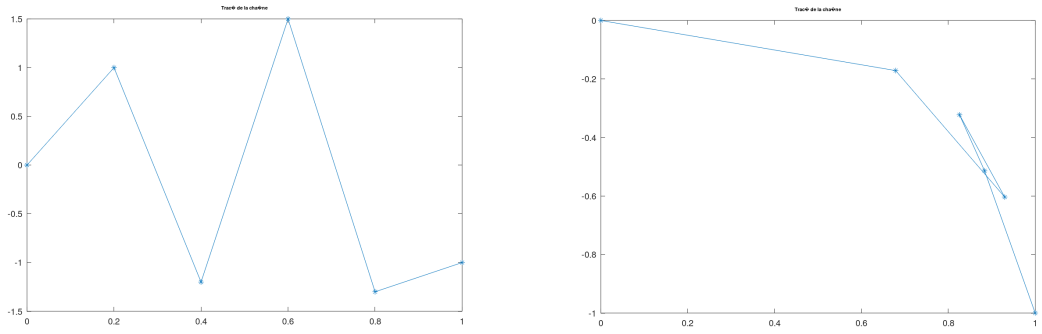
— `phi` : la valeur $\varphi(z_k)$.
— `|d|` : la norme de la direction de descente $\|d_k\|_\infty$.
— `|gl|` : la norme du gradient du lagrangien $\|\nabla_x \ell(x_k, \lambda_k)\|_\infty$.
— `|ce|` : la norme des contraintes $\|c(x_k)\|_\infty$.

3.5 Etude des cas-tests

3.5.1 Cas-test 2d

On reprend le cas-test de la partie 2.4.4, qui avec l'algorithme de Newton classique ne converge pas. Si la procédure de globalisation marche, on devrait observer une convergence vers un point stationnaire en utilisant l'algorithme avec la recherche linéaire.

La chaîne initiale et finale sont représentées dans la figure ci-dessous.



(d) (e)

FIGURE 11 – Chaîne initiale à gauche, et chaîne finale à droite

Le tableau obtenu en utilisant `options.verb = 1` nous indique que l'on a convergence après 11 itérations. On l'affiche ci-dessous. On voit bien que la composante maximale en valeur absolue du gradient du lagrangien tend vers zéro. On observe cela aussi pour les contraintes et pour $\varphi(z_k)$. On observe aussi que les dernières itérations ont une valeur unitaire d'`alpha`. Ces sont des bons comportements attendus.

```
-------------------------------------------------------------------------------
iter     |gl|          |ce|          |x|       |lm|       alpha        phi
 1    3.6847e+00    4.4896e+00    1.1e+00    2.5e+00    5.000e-01    1.10626e+02
 2    9.8741e-01    1.1125e+00    6.1e-01    5.5e-01    1.000e+00    4.62475e+01
 3    9.2892e-01    1.0475e+00    7.2e-01    6.3e-01    6.250e-02    2.50660e+00
 4    4.8543e-01    3.5621e-01    1.1e+00    7.1e-01    1.000e+00    2.40141e+00
 5    1.9198e-01    2.0952e-01    1.0e+00    1.1e+00    5.000e-01    4.20566e-01
 6    2.2373e-01    5.1887e-02    9.1e-01    1.8e+00    1.000e+00    1.28105e-01
 7    1.7684e-01    1.7109e-02    9.6e-01    2.8e+00    1.000e+00    8.39509e-02
 8    3.3076e-02    1.2590e-03    9.3e-01    3.4e+00    1.000e+00    3.64052e-02
 9    1.4402e-03    5.7390e-05    9.3e-01    3.5e+00    1.000e+00    1.96661e-03
10    1.2690e-06    3.5859e-08    9.3e-01    3.5e+00    1.000e+00    2.26161e-06
11    1.4942e-12    6.2728e-14    9.3e-01    3.5e+00    1.000e+00    2.36695e-12
-------------------------------------------------------------------------------
Le nombre maximal d'iterations autorisees est 500 et le nombre d'iterations effectuees est 11
Le comportement de l'optimiseur est 0
```

FIGURE 12 – Tableau de résultats pour le cas-test 2d

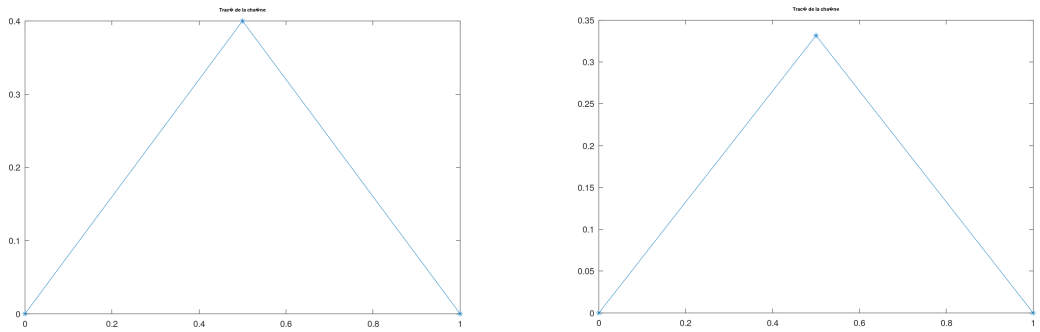
Le vecteur avec les coordonnées finales des points internes de la chaîne est `(0.678, 0.929, 0.825, 0.883, -0.171, -0.603, -0.322, -0.513)` et celui avec les multiplicateurs de Lagrange est `(-0.3007, -0.8129, 1.9602, -3.5331, -1.7512)`.

En comparant la figure de cette chaîne finale avec la chaîne finale 3.b, on note qu'elles sont bien les mêmes. On avait vérifié que cette chaîne est un point stationnaire pour F. Ainsi, **on conclut que ce cas-test, qui ne convergeait pas avant, maintenant converge en** 11 **itérations vers un point stationnaire de** f. C'est une bonne indication que la méthode de globalisation par recherche linéaire implémentée fonctionne.

On vérifie par le calcul que le point trouvé par l'algorithme est bien un point stationnaire de f. Le spectre de la hessienne réduite est `(4.9446,-1.8334,-15.0153)`. Deux valeurs propres sont négatives et une est positive, donc le point trouvé est un **point selle** de f.

3.5.2 Cas-test 3a

Prenons un cas simple : deux barres et le point fixé est $(a, b) = (1, 0)$. Les contraintes sont que les longueurs des barres finales doivent être 0.6. La position initiale de l'unique point intermédiaire est $(0.5, 0.4)$. On montre dessous la chaîne initiale et la chaîne finale après avoir exécuté l'algorithme.



(a) (b)

FIGURE 13 – Chaîne initiale à gauche, et chaîne finale à droite

Si on réfléchit un peu, on voit que le résultat est exactement ce que l'on pouvait attendre. La chaîne commence dans une position très stable, dans un maximum d'énergie potentiel et avec presque les longueurs recherchées, alors elle reste essentiellement la même. L'algorithme converge très rapidement : on fait juste 4 itérations.

```
------------------------------------------------------------------------------
iter     |gl|         |ce|         |x|       |lm|       alpha        phi
 1    1.9184e-01   3.9062e-03   5.0e-01   3.0e-01   1.000e+00   9.09315e-01
 2    3.4094e-03   3.3490e-05   5.0e-01   4.5e-01   1.000e+00   1.84158e-02
 3    5.3265e-07   2.5482e-09   5.0e-01   4.5e-01   1.000e+00   5.81303e-06
 4    6.2172e-15   0.0000e+00   5.0e-01   4.5e-01   1.000e+00   1.41867e-13
------------------------------------------------------------------------------

Le nombre maximal d'iterations autorisees est 500 et le nombre d'iterations effectuees est 4
Le comportement de l'optimiseur est 0
```

FIGURE 14 – Tableau de résultats pour le cas-test 3a

On vérifie bien que le gradient du lagrangien, les contraintes, et φ convergent vers zéro. Le vecteur des coordonnées finales de la chaîne est `(0.5, 0.331)` et celui des multiplicateurs de Lagrange est `(-0.4523,-0.4523)`.

Discutons maintenant l'optimalité de la chaîne trouvée. Notons tout d'abord qu'il n'y a pas de sens de parler de la hessienne réduite. En effet, comme on a un seul point (un seul degré de liberté), l'équation $c = 0$ a pour ensemble de solutions un ensemble de dimension zéro, c'est-à-dire, des points isolés. Bien sûr, on peut voir ça en regardant qu'il n'y a aucune chaîne voisine à la chaîne finale trouvée qui satisfait les contraintes. Notons que cela n'est pas le cas pour une chaîne constituée de plusieurs points car on peut alors continûment bouger la chaîne en respectant les contraintes.

Or, il n'y a pas de cône linéarisant ni tangent pour un point (sauf le point lui même), et alors il n'y a pas de sens de parler de la hessienne réduite. En fait, si on demande à notre programme de la calculer, on obtient un spectre vide (hors la valeur propre nulle).

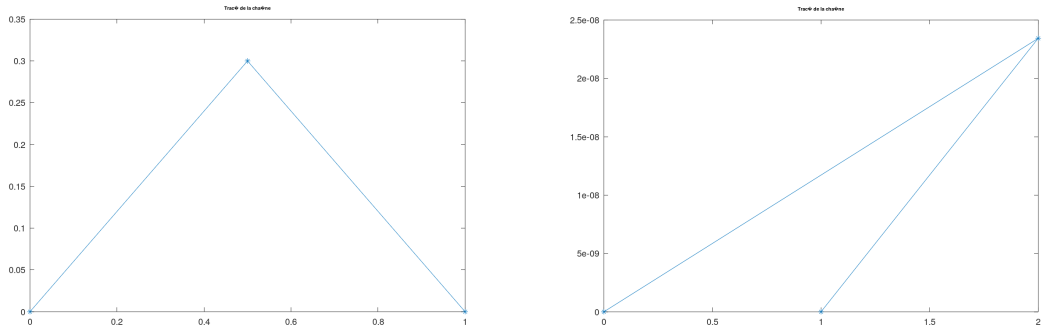
Ainsi pour conclure l'optimalité de la chaîne trouvée, on doit calculer manuellement toutes les autres chaînes admissibles, calculer la valeur de la fonction-objectif sur ces solutions, et enfin comparer. L'équation $c = 0$ nous donne le système

$$\begin{cases} x^2 + y^2 = 1 \\ (x-1)^2 + y^2 = 1 \end{cases}$$

et on calcule facilement que les seules solutions sont $x = \frac{1}{2}$ et $y = \pm\frac{\sqrt{3}}{2}$, c'est-à-dire, il y a une chaîne symétrique par rapport à l'axe des abscisses de celle que l'on a trouvée. On voit facilement que l'énergie potentielle diffère juste par un signe. Alors la chaîne vers laquelle on converge est un **maximum** sur l'ensemble admissible.

3.5.3 Cas-test 3b

Prenons une variation du cas-test précédent. On prend toujours deux barres, et comme deuxième point fixé $(a, b) = (1, 0)$. Les contraintes sont maintenant que les longueurs des barres finales doivent être 2 et 1. La position initiale de l'unique point intermédiaire est $(0.5, 0.3)$. On montre dessous la chaîne initiale et la chaîne finale après exécuter l'algorithme.



(a) (b)

FIGURE 15 – Chaîne initiale à gauche, et chaîne finale à droite

Les coordonnées de la chaîne finale sont `(2, 2.3428e-08)` et ceux des multiplicateurs de Lagrange sont `(3.2013e+07,-6.4027e+07)`.

Quand on tourne l'algorithme, on reçoit immédiatement des messages de précaution dans le code, qui indiquent une matrice singulière dans ce cas-test. On note aussi que la convergence arrive, mais elle n'est pas si rapide que précédemment : on arrête l'algorithme après 304 itérations. Voici les dernières lignes du tableau d'informations :

```
298    9.0520e-03   8.8818e-16   2.0e+00   6.0e+07   3.125e-02   4.19077e-05
299    8.9604e-03   8.8818e-16   2.0e+00   6.1e+07   3.125e-02   4.09696e-05
300    8.8804e-03   8.8818e-16   2.0e+00   6.1e+07   3.125e-02   4.01443e-05
301    8.8122e-03   8.8818e-16   2.0e+00   6.2e+07   3.125e-02   3.94303e-05
302    8.7562e-03   8.8818e-16   2.0e+00   6.3e+07   3.125e-02   3.88273e-05
303    8.7127e-03   8.8818e-16   2.0e+00   6.4e+07   3.125e-02   3.83354e-05
304    1.1585e-10   4.4409e-16   2.0e+00   6.4e+07   1.000e+00   3.79559e-05
-------------------------------------------------------------------------------
Le nombre maximal d'iterations autorisees est 500 et le nombre d'iterations effectuees est 304
Le comportement de l'optimiseur est 0
```

FIGURE 16 – Tableau de résultats pour le cas-test 3b

Ce tableau nous indique qu'on arrive bien à une chaîne qui satisfait les contraintes, et dans laquelle le gradient du lagrangien est très petit. Les coordonnées finales du point interne à la chaîne est `(2, 23.428e-9)`, et cela nous montre que **la barre tend vers une position horizontale**. On peut bien sûr vérifier que c'est le cas en prenant une tolérance `options.tol` plus petite dans l'algorithme et alors la seconde composante du point interne est d'ordre inférieur à 10^{-9}.

Maintenant on se demande pourquoi ces messages de précaution apparaissent. Pour trouver une direction de descente, on doit résoudre le système de Newton, dont le membre linéaire est $F'(z_k)$:

$$\begin{pmatrix} L_k & A_k^\top \\ A_k & 0 \end{pmatrix} \begin{pmatrix} d_k \\ \mu_k \end{pmatrix} = - \begin{pmatrix} \nabla f(x_k) + A_k^\top \lambda_k \\ c(x_k) \end{pmatrix}$$

On note A la matrice jacobienne des contraintes dans le système ci-dessus. Or, on l'a calculée dans la partie 1.3.1. Dans ce cas avec seulement deux barres, on peut l'écrire explicitement :

$$A = \begin{pmatrix} \frac{\partial c_1}{\partial x} & \frac{\partial c_1}{\partial y} \\ \frac{\partial c_2}{\partial x} & \frac{\partial c_2}{\partial y} \end{pmatrix} = \begin{pmatrix} 2(x-x_0) & 2(y-y_0) \\ 2(x-x_2) & 2(y-y_2) \end{pmatrix}$$

Ensuite on observe que la matrice $F'(z_k)$ est une matrice par blocs avec un bloc diagonal zéro. Ainsi le calcul du déterminant peut être considérablement simplifié (voir ici) :

$$\det \begin{pmatrix} L_k & A_k^\top \\ A_k & 0 \end{pmatrix} = \det(-A_k A_k^\top)$$

Mais comme on a seulement deux barres, ce déterminant peut être calculé facilement.

$$\det(-4) \begin{pmatrix} (x-x_0)^2 + (y-y_0)^2 & (x-x_0)(x-x_2) + (y-y_0)(y-y_2) \\ (x-x_2)(x-x_0) + (y-y_2)(y-y_0) & (x-x_2)^2 + (y-y_2)^2 \end{pmatrix}$$

On peut ignorer le facteur -4 car le déterminant est multilinéaire sur les colonnes d'une matrice. Dans notre cas, on peut considérer que $(y - y_0) \approx 0$ et $(y - y_2) \approx 0$. Alors, il faut juste calculer

$$\det \begin{pmatrix} (x-x_0)^2 & (x-x_0)(x-x_2) \\ (x-x_2)(x-x_0) & (x-x_2)^2 \end{pmatrix} \approx 0$$

C'est-à-dire, on a bien que la matrice $F'(z_k)$ est singulière si la chaîne finale obtenue a des barres horizontales, et on s'approche de ce cas là. On vérifie aussi que ce même argument vaut si la chaîne finale a des barres verticales.

Il y a une autre façon intéressante, moins algébrique, de voir cela. A est singulière si et seulement si $A^\top$ est singulière aussi, car leur déterminant sont égaux. Dans la matrice A, on voit facilement que si les barres finales sont verticales ou horizontales, alors A est singulière (car ainsi on a une colonne nulle dans la matrice). Or, dans la deuxième colonne par blocs de $F'(z)$, $A^\top$ est seule au-dessus d'un bloc de zéros. Alors si les colonnes de $A^\top$ sont linéairement dépendantes (ce qui implique $\det A = 0$) alors ajouter des zéros à la fin de ses vecteurs colonnes ne change pas cette dépendance linéaire. Autrement dit, les colonnes de $F'(z_k)$ correspondantes à la deuxième colonne par blocs seront linéairement dépendantes, et alors on aura $\det F'(z_k) = 0$.

C'est important de noter que lorsque les barres s'approchent de la position horizontale, on n'a pas exactement $y - y_0 = 0$ ni $y - y_2 = 0$. Ainsi, le déterminant n'est pas exactement zéro, mais presque. Cela signifie que si on prend une tolérance suffisamment haute alors le système de Newton peut trouver une solution, car les barres ne seront pas vraiment horizontales. En fait, même si on reçoit un message de précaution, on peut trouver une solution au système de Newton, dès que la méthode d'inversion de matrice par *Matlab* n'implique pas d'erreur stoppant le programme global. C'est exactement ce qui se passe dans notre cas : *Matlab* nous prévient de la singularité de $F'(z_k)$ mais poursuit les résolutions de l'équation matricielle car les tolérances `options.tol` arrêtent l'algorithme avant que l'inversion d'une ultime matrice fasse stopper l'algorithme global. C'est pour cela que l'algorithme converge quand même, mais à cause du mauvais conditionnement de la solution, cette convergence a un mauvais comportement : elle est lente et génère des "warnings". En effet, dans l'algorithme de Newton, l'une des conditions de régularité pour obtenir une convergence locale est que $F'(z_\star)$ doit être inversible.

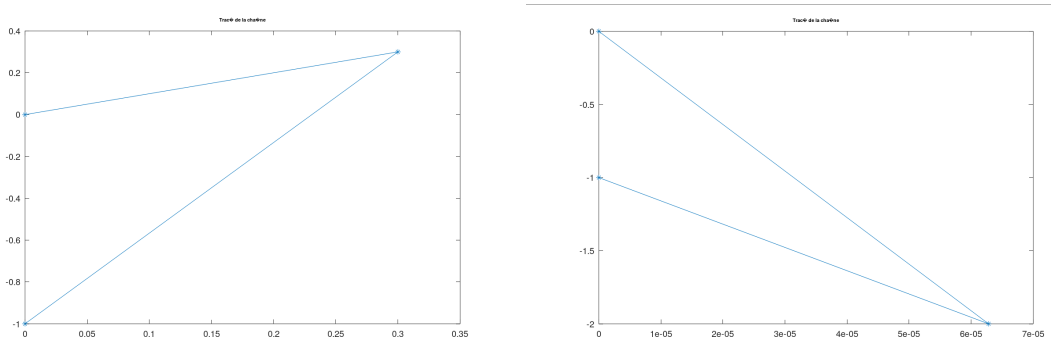
Discutons maintenant de l'optimalité de la chaîne trouvée. Comme dans le cas précédent, il n'y a pas de sens de parler de la hessienne réduite, car les chaîne admissibles sont des points isolés. Ainsi, pour conclure l'optimalité de la chaîne trouvée, on doit calculer manuellement toutes les autres solutions, calculer la valeur de la fonction objectif sur ces solutions, et enfin comparer. En fait, la chaîne horizontale vers laquelle on converge est l'unique solution de $c = 0$:

$$\begin{cases} x^2 + y^2 = 4 \\ (x-1)^2 + y^2 = 1 \end{cases}$$

On a que l'unique solution est $(x, y) = (2, 0)$ pour le point intermédiaire de la chaîne. **Il n'a pas donc de sens de parler de minimum ou maximum sur l'ensemble admissible**, cette chaîne est la seule possible.

3.5.4 Cas-test 3c

Finalement, faisons un dernier cas-test. On prend toujours deux barres, mais maintenant on prend comme deuxième point fixé $(a, b) = (0, -1)$. Les contraintes sont toujours que les longueurs des barres finales doivent être 2 et 1. La position initiale de l'unique point intermédiaire est $(0.3, 0.3)$. On montre ci-dessous la chaîne initiale et la chaîne finale après avoir exécuté l'algorithme.



(a) (b)

FIGURE 17 – Chaîne initiale à gauche, et chaîne finale à droite

Les coordonnées de la chaîne finale sont `(6.2814e-05, -2)` et ceux des multiplicateurs de Lagrange sont `(0.75,-0.75)`.

On a une convergence assez rapide en 18 itérations. Voici le tableau avec les informations de chaque itération :

```
-------------------------------------------------------------------------------
iter     |gl|          |ce|          |x|      |lm|       alpha         phi
 1    2.9368e+00   3.6221e+00   5.7e-01   7.0e-01   3.125e-02   1.19686e+01
 2    2.3413e+00   1.9677e+00   1.6e+00   4.6e-01   2.500e-01   1.12225e+01
 3    1.1150e+00   1.6839e+00   1.6e+00   2.3e-01   5.000e-01   5.64142e+00
 4    3.1499e-02   1.0591e+00   2.0e+00   7.9e-01   1.000e+00   2.04411e+00
 5    5.6627e-03   2.6478e-01   2.0e+00   7.6e-01   1.000e+00   1.12252e+00
 6    1.4157e-03   6.6196e-02   2.0e+00   7.6e-01   1.000e+00   7.01266e-02
 7    3.5392e-04   1.6549e-02   2.0e+00   7.5e-01   1.000e+00   4.38291e-03
 8    8.8479e-05   4.1373e-03   2.0e+00   7.5e-01   1.000e+00   2.73932e-04
 9    2.2120e-05   1.0343e-03   2.0e+00   7.5e-01   1.000e+00   1.71208e-05
10    5.5300e-06   2.5858e-04   2.0e+00   7.5e-01   1.000e+00   1.07005e-06
11    1.3825e-06   6.4645e-05   2.0e+00   7.5e-01   1.000e+00   6.68780e-08
12    3.4562e-07   1.6161e-05   2.0e+00   7.5e-01   1.000e+00   4.17987e-09
13    8.6406e-08   4.0403e-06   2.0e+00   7.5e-01   1.000e+00   2.61242e-10
14    2.1601e-08   1.0101e-06   2.0e+00   7.5e-01   1.000e+00   1.63276e-11
15    5.4004e-09   2.5252e-07   2.0e+00   7.5e-01   1.000e+00   1.02048e-12
16    1.3501e-09   6.3129e-08   2.0e+00   7.5e-01   1.000e+00   6.37798e-14
17    3.3752e-10   1.5782e-08   2.0e+00   7.5e-01   1.000e+00   3.98624e-15
18    8.4381e-11   3.9456e-09   2.0e+00   7.5e-01   1.000e+00   2.49140e-16
-------------------------------------------------------------------------------
Le nombre maximal d'iterations autorisees est 500 et le nombre d'iterations effectuees est 18
Le comportement de l'optimiseur est 0
```

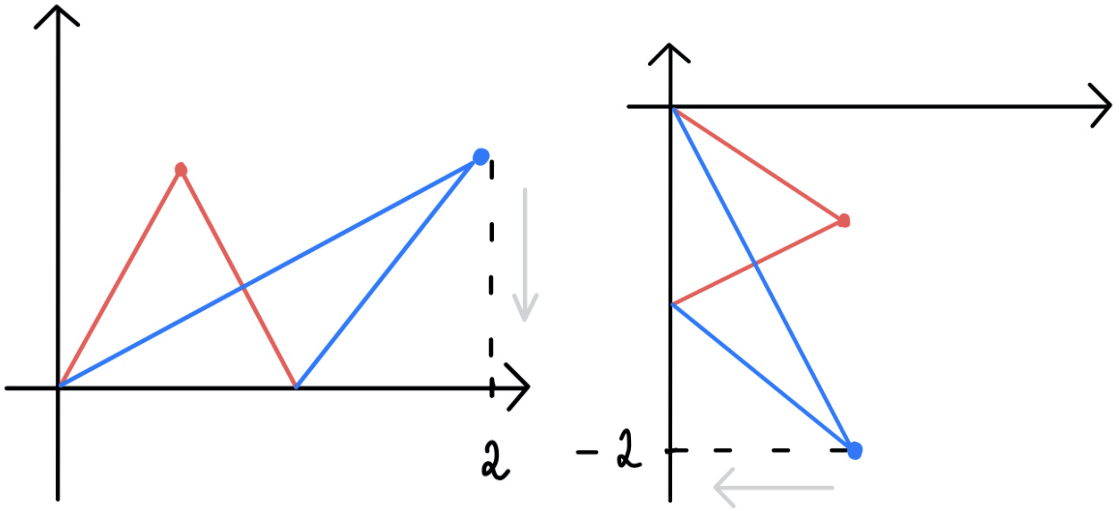
FIGURE 18 – Tableau de résultats pour le cas-test 3c

La coordonnée finale du point interne à la chaîne est `(62.814e-6, -2)`, c'est-à-dire, cette fois **on tend vers une chaîne verticale.**

On observe quelque chose étrange à première vue. Selon la discussion dans 3.b, on voit que l'on s'approche d'une solution où les barres sont verticales et cela devrait donner aussi des messages d'erreur à cause de la singularité de la matrice $F'(z_\star)$. En fait, en utilisant le même calcul dans ce cas là avec $x - x_0 \approx 0$ et $x - x_2 \approx 0$, on voit que $\det F'(z_k) = 0$ pour des barres verticales. Cependant, on ne voit pas de "warnings".

Après quelques expérimentations, l'explication de cette différence est que le problème arrive dans le cas 3b car $\det F'(z_k) \to 0$ plus vite que dans le cas 3c. Voici une explication heuristique pourquoi cela arrive.

Si l'on analyse le tableau donné par `options.verb = 1`, on observe que dans les deux cas la colonne `|x|` atteint dans les toutes premières itérations la valeur 2 et reste constante pour les reste des itérations. Cela correspond au fait que la chaîne va tout d'abord vers la coordonnée $x = 2$ ou $y = -2$ pour le cas 3b et 3c respectivement. Une fois qu'elle atteint ces coordonnées, le point intérieur de la chaîne descend vers l'axe x dans le cas 3b, ou va vers l'axe y dans le cas 3c. Le dessin ci-dessous représente ce phénomène :



(a) (b)

FIGURE 19 – Les premières itérations dans le cas 3b et 3c

Or, $\det F'(z_k) \propto \det(A^\top A)$ qui est juste une sommes des produits des différences $(x - x_0), (x - x_2), (y - y_0), (y - y_2)$. Donc le fait que le déterminant dans le cas 3b va plus vite vers zéro que le cas 3c signifie que les différences $(y - y_0)$ et $(y - y_2)$ vont vers zéro plus vite que les différences $(x - x_0)$ et $(x - x_2)$ dans le cas 3c.

La figure ci-dessus donne l'intuition que ça arrive car c'est "plus évident" dans le cas 3b que la direction de descente pour l'énergie potentielle (et alors pour le lagrangien) est vers le bas. Physiquement ça correspond au fait que la gravité pointe vers le bas. Dans le cas 3c, c'est relativement "moins évident" que la bonne direction est l'horizontale. Alors la convergence $\det F'(z_k) \to 0$ est plus lente. Pour discuter l'optimalité de la chaîne trouvée, on remarque que comme dans le cas précédent l'équation $c = 0$ donne un système que l'on résout facilement et ainsi on vérifie que la chaîne verticale trouvée est l'unique chaîne admissible. Ainsi, **il n'a pas de sens de dire si elle est un maximum ou un minimum dans l'ensemble admissible**, car elle est unique.

Conclusion

Ce projet nous a permis de mettre en application les notions vues en cours d'OPT-201. On a étudié un problème concret à travers une étude théorique et des résolutions pratiques. On a fait en sorte d'écrire un code indépendant du problème avec des fonctions les plus indépendantes possibles et facilement modifiables. Ainsi, on peut réutiliser le code pour résoudre d'autres problèmes.

On peut se demander comment étudier un problème avec non seulement des contraintes d'égalités mais aussi d'inégalités. C'est ce que l'on verra dans la suite du projet en OPT-202 avec l'ajout d'un plancher sous la chaîne.