Projet d'optimisation en Matlab

Équilibre d'une chaîne articulée

Séance 1 : Modélisation et écriture du simulateur

On 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. Cette position est obtenue en minimisant l'énergie potentielle de la chaîne sous des contraintes appropriées.

1 Modélisation du problème

On décide de prendre comme variables décrivant la position de la chaîne, les coordonnées des nœuds de celle-ci, c'est-à-dire de ses points d'articulation. S'il y a n_b barres, il y a $n_n = n_b - 1$ nœuds, car on ne compte pas les extrémités de la chaîne, supposées fixées aux points $(0, 0)$ et (a, b). L'extrémité (a, b) pourra varier d'un cas-test à l'autre, de même que le nombre de barres n_b et leurs longueurs. On se référera à la figure 1 pour les notations.

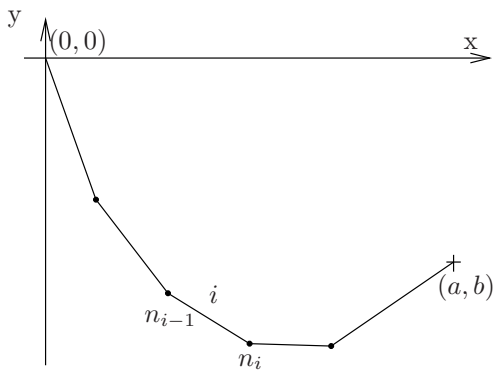

Figure 1: Chaîne formée de barres

Les coordonnées horizontale et verticale du nœud i sont notées $(\mathrm{x}_i, \mathrm{y}_i)$. Ce sont les variables du problème, qui sont donc au nombre de $n = 2n_n$.

Si la barre i a pour extrémités les nœuds d'indices $i - 1$ et i, sa longueur vaut

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

On note $(\mathrm{x}_0, \mathrm{y}_0) := (0, 0)$ et $(\mathrm{x}_{n_b}, \mathrm{y}_{n_b}) := (a, b)$. Alors, à une constante additive près, l'énergie potentielle de la chaîne s'écrit

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

où $\gamma > 0$ est la constante de gravité et $m_i(\mathrm{x}, \mathrm{y})$ la masse de la i-ième barre supposée uniformément répartie le long de celle-ci. Pour simplifier, on supposera que $\gamma = 1$ et $m_i(\mathrm{x}, \mathrm{y}) = l_i(\mathrm{x}, \mathrm{y})$, ce qui donne

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

Les contraintes à respecter portent sur la longueur des barres : L_i pour la barre i. La i-ième contrainte s'écrit donc $c_i(\mathrm{x},\mathrm{y}) = 0$, où

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

Elle porte sur le carré des longueurs de manière à ce que les contraintes soient différentiables.

On a compris que l'on cherche à résoudre le problème d'optimisation sous contraintes d'égalité en $(\mathrm{x}_i, \mathrm{y}_i)$, $i = 1, \ldots, n_n$:

$$\begin{cases} \min \mathrm{E}(\mathrm{x},\mathrm{y}) \\ c_i(\mathrm{x},\mathrm{y}) = 0, \quad i = 1, \ldots, n_b. \end{cases}$$

Notons que cela revient au même de résoudre le problème plus simple (le critère devient linéaire) :

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

où

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

En effet sur l'ensemble admissible, $E = e$ et donc les deux problèmes ont les mêmes solutions (ne pas oublier que l'on cherche un équilibre statique). C'est donc ce dernier problème que l'on résoudra.

2 Le but du projet

Le projet consiste à écrire en Matlab un *simulateur* (on dit aussi un *oracle*) et un *optimiseur* pour le problème (P) : trouver l'équilibre statique de la chaîne articulée formée de barres rigides.

Afin de fixer les notations, le problème que nous cherchons à résoudre peut s'écrire formellement de la manière suivante :

$$\begin{cases} \min e(x) \\ c_i(x) = 0, \quad i \in E, \end{cases}$$

où $x \in \mathbb{R}^n$ est le couple (x,y), $n = 2n_n$ et $E = \{1, \ldots, n_b\}$. On écrira par la suite le lagrangien du problème de la manière suivante :

$$\ell(x, \lambda) = e(x) + \lambda^{\mathsf{T}} c(x),$$

où $\lambda \in \mathbb{R}^{|E|}$.

3 Écriture du programme Matlab

Le programme principal

Écrire un programme principal (un script), appelé `ch`, contenant les données du problème (voir § 6) et appelant le simulateur `chs` (voir ci-dessous) pour calculer `e`, `c`, `g`, `a` et `hl`. On appellera `ch` depuis Matlab.

Le simulateur

On écrira un simulateur sous la forme d'une fonction Matlab se présentant comme suit

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

En entrée :

`indic` : pilote le comportement du simulateur :
- $= 1$: `chs` fait un tracé de la chaîne (fonction `plot` de Matlab);
- $= 2$: `chs` calcule `e` et `c`;
- $= 4$: `chs` calcule `e`, `c`, `g` et `a`;
- $= 5$: `chs` calcule `hl`.

`xy` : vecteur-colonne contenant d'abord les abscisses $\{\mathrm{x}_i\}_{i=1}^{n_n}$ des nœuds, puis leurs ordonnées $\{\mathrm{y}_i\}_{i=1}^{n_n}$, i.e., $\mathtt{xy} = (\mathrm{x}_1, \ldots, \mathrm{x}_{n_n}, \mathrm{y}_1, \ldots, \mathrm{y}_{n_n})$. C'est le vecteur x à optimiser.

`lm` : vecteur-colonne de dimension n_b contenant les multiplicateurs de Lagrange pour les contraintes d'égalité, $\lambda = \{\lambda_i\}_{i=1}^{n_b}$.

En sortie :

`e` : valeur de l'énergie potentielle en `xy` (c'est-à-dire pour les nœuds dont les coordonnées sont dans `xy`).

`c` : valeur en `xy` des contraintes sur la longueur des barres, `c` est un vecteur-colonne de dimension n_b, que nous noterons c.

`g` : gradient de e en `xy`, c'est un vecteur-colonne de dimension $2n_n$, on notera $g = \nabla\, e(x)$.

`a` : jacobienne des contraintes d'égalité en `xy`, matrice de n_b lignes et $2n_n$ colonnes, que nous noterons $A(x) = c'(x)$.

`hl` : hessien du lagrangien en `xy` et `lm` :

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

(les λ_i sont dans `lm`).

`indic` : décrit le résultat de la simulation :
- $= 0$: sortie normale (ce qui a été demandé a été fait),
- $= 1$: paramètre(s) d'entrée non correct(s).

L'optimiseur que l'on va écrire est général, il ne dépend pas du problème traité. Il s'attend donc à ce que tout simulateur se présente sous la forme décrite ci-dessus. Pour évaluer les quantités `e`, `c`, `g`, `a` et `hl`, il manque des données que l'on passera au simulateur en tant que « variables globales » (instruction `global` en Matlab). Il s'agit de :

- `L` : vecteur donnant les longueurs désirées des barres.
- `A` : a, abscisse de la deuxième extrémité de la chaîne.
- `B` : b, ordonnée de la deuxième extrémité de la chaîne.

Quelques conseils sur l'implémentation du problème

- La variable `xy` contient tous les paramètres à optimiser. L'optimiseur va les modifier à chaque itération. Il ne faut donc pas y mettre les extrémités de la chaîne qui sont des nœuds fixes, des données du problème.

Quelques conseils sur l'utilisation de Matlab

- En Matlab, gardez la structure de l'algèbre linéaire standard. En particulier, il est préférable que tous les vecteurs soient des vecteurs-colonnes.
- Matlab est un *interpréteur*. Il est donc peu efficace sur des programmes contenant des boucles (`for` et `while`), d'autant moins qu'elles sont imbriquées. Il faut en effet que les instructions soient interprétées à chaque tour de boucle (leur signification ayant pu changer), souvent à un coût non négligeable. D'autre part, Matlab présente l'avantage de pouvoir manipuler très efficacement les vecteurs et les matrices. Il faut en profiter.

  Dès lors, lorsqu'on évalue une fonction, on tirera avantage d'une formulation vectorielle des expressions à évaluer, de manière à éviter l'utilisation de boucles. Ceci demande souvent que l'on utilise davantage de place-mémoire.
- Dans ce problème, on aura intérêt à utiliser des matrices représentées comme structures creuses (voir les fonctions `sparse`, `spdiags`...). Cela permet de réduire l'espace-mémoire et d'accélérer les calculs.

  Si le simulateur génère des jacobiennes et des hessiens creux, l'optimiseur héritera de ces structures, sans qu'il faille y faire attention.
- En optimisation il est souvent nécessaire de programmer des boucles (processus itératifs), que l'on quitte lorsqu'un test est vérifié, disons `test_verifié` est vrai. De manière à ne pas devoir évaluer ce test à l'entrée de la boucle (c'est parfois contraignant et il peut y avoir plusieurs raisons de quitter la boucle des itérations), on utilise souvent le codage suivant

```
while true
   ...
   if test_verifié; break; end
   ...
end
```

  L'utilisation de `continue` plutôt que `break` aurait pour effet de ne pas exécuter le code qui suit le test et de passer au tour suivant.
- Pour les débogage du code, il sera parfois approprié d'utiliser la commande `keyboard`, qui permet d'avoir la main à l'endroit où cette instruction apparaît dans le programme. On peut alors, à partir de la fenêtre Matlab, examiner la valeur des variables à cet endroit, y détecter les erreurs, les inconsistances...

  Pour reprendre l'exécution, taper `return`. Pour arrêter l'exécution, taper `dbquit`.

4 Vérification des dérivées

Une tâche délicate et cruciale pour le succès des séances suivantes est de s'assurer de l'exactitude du simulateur. Deux aspects de celui-ci sont à vérifier.

- *Le calcul exact des fonctions e et c.* Ceci pourra se vérifier sur des cas simples pour lesquels on connaît la valeur de ces fonctions (chaîne formée de deux ou trois barres de longueurs égales, par exemple).

- *L'exactitude des dérivées.* Traditionnellement, l'exactitude de la dérivée d'une fonction $\varphi : \mathbb{R}^n \to \mathbb{R}^m$ se vérifie par différences finies. Pour $i = 1, \ldots, n$, on compare les valeurs des dérivées $\varphi'(x) \cdot e^i = \partial\varphi(x)/\partial x_i \in \mathbb{R}^m$ (e^i désigne le i-ième vecteur de base de $\mathbb{R}^n$) données par le simulateur avec l'un des quotients différentiels

$$\begin{aligned} \frac{\varphi(x+t_i e^i) - \varphi(x)}{t_i} &= \varphi'(x) \cdot e^i + O(t_i), \\ \frac{\varphi(x+t_i e^i) - \varphi(x-t_i e^i)}{2t_i} &= \varphi'(x) \cdot e^i + O(t_i^2), \end{aligned}$$

où le pas t_i est « bien choisi » (ni trop grand ni trop petit). Le reste indique que la seconde formule est plus précise que la première. On sait que le choix de t_i est délicat. Il est standard de prendre

$$t_i = \varepsilon^{1/2} \max(\tau_i, |x_i|),$$

où ε est l'*epsilon-machine* (eps en Matlab), qui est le plus petit nombre flottant positif tel que $\mathrm{fl}(1+\varepsilon) \neq 1$, et τ_i est une valeur « typique » de x_i (ici on peut prendre $\tau_i = 1$).

En suivant cette procédure dans notre code, nous avons obtenu pour les dérivées de e, le tableau suivant :

```
i      pas        f'(i)          DF           erreur
1   1.49e-08   0.00000e+00   0.00000e+00   0.00000e+00 abs
2   1.49e-08   0.00000e+00   0.00000e+00   0.00000e+00 abs
3   1.49e-08   0.00000e+00   0.00000e+00   0.00000e+00 abs
4   1.49e-08   0.00000e+00   0.00000e+00   0.00000e+00 abs
5   1.49e-08   6.00000e-01   6.00000e-01   9.93411e-09 rel
6   2.24e-08   4.00000e-01   4.00000e-01   9.93411e-09 rel
7   2.24e-08   2.50000e-01   2.50000e-01   0.00000e+00 rel
8   1.94e-08   3.50000e-01   3.50000e-01   3.66798e-08 rel
```

La première colonne donne l'indice de la composante du gradient, la seconde le pas t_i choisi, la troisième la composante du gradient calculé, la quatrième les quotients différentiels correspondants et la cinquième l'erreur relative ou absolue entre les deux valeurs de la dérivée (une erreur relative de 10^{-8} est excellente et ne laisse pas de doute sur la justesse des dérivées calculées).

Une autre manière de vérifier la bonne valeur des dérivées sera discutée dans une autre séance.

5 Questions

1.1. Le problème (P) a t-il (au moins) une solution, s'il existe une chaîne admissible ?

1.2. Si le problème (P) a une solution x_*, peut-on en déduire qu'il existe un multiplicateur optimal λ_* tel que $\nabla_x \ell(x_*, \lambda_*) = 0$?

6 Cas-tests

Cas-test 1 : 5 barres de longueur

```
L = [0.7 0.5 0.3 0.2 0.5]';
```

Deuxième point de fixation de la chaîne : $(a, b) = (1, -1)$. Position initiale des nœuds :

```
xy = [ 0.2    0.4    0.6    0.8 ...
      -1.0   -1.5   -1.5   -1.3 ]';
```