Projet d'optimisation en Matlab

Équilibre d'une chaîne articulée

Séance 3 : Globalisation par recherche linéaire

On reprend le problème de trouver une position d'équilibre d'une chaîne au repos. Rappelons que, sur l'ensemble admissible, il s'agit de minimiser l'énergie potentielle de la chaîne sur un ensemble défini au moyen de contraintes d'égalité (respect des longueurs des barres). La chaîne est constituée de n_b barres rigides contenues dans un plan vertical (x, y) et s'articule en $n_n = n_b - 1$ nœuds, dont les positions sont à déterminer. Après modélisation, le problème s'écrit

$$(P) \quad \begin{cases} \min f(x) \\ c(x) = 0, \end{cases} \tag{3.9}$$

où $x \in \mathbb{R}^n$ est le couple (x, y) des abscisses $\mathrm{x} \in \mathbb{R}^{n_n}$ et ordonnées $\mathrm{y} \in \mathbb{R}^{n_n}$ des nœuds de la chaîne (donc $n = 2n_n$), $f(x) \in \mathbb{R}$ est, sur l'ensemble admissible, l'énergie potentielle de la chaîne dans la configuration donnée par x et $c(x) \in \mathbb{R}^m$ donne la valeur des contraintes sur la longueur des barres (il y a donc $m := n_b$ contraintes d'égalité). Le lagrangien du problème est la fonction $\ell : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ qui prend en $(x, \lambda) \in \mathbb{R}^n \times \mathbb{R}^m$ la valeur

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

1 Rappel de l'algorithme de Newton

On rappelle que l'algorithme de Newton (local) consiste à trouver une solution primale-duale $z_* := (x_*, \lambda_*) \in \mathbb{R}^n \times \mathbb{R}^m$ de son système d'optimalité du premier ordre qui s'écrit

$$F(z) = 0,$$

où $F : \mathbb{R}^{n+m} \to \mathbb{R}^{n+m}$ est définie en $z = (x, \lambda) \in \mathbb{R}^{n+m}$ par

$$F(z) := \begin{pmatrix} \nabla f(x) + c'(x)^\mathsf{T} \lambda \\ c(x) \end{pmatrix}.$$

Étant donné un itéré $z_k := (x_k, \lambda_k) \in \mathbb{R}^{n+m}$, l'itéré suivant $z_{k+1} = (x_{k+1}, \lambda_{k+1}) \in \mathbb{R}^{n+m}$ est mis à jour par

$$x_{k+1} := x_k + d_k \qquad \text{et} \qquad \lambda_{k+1} := \lambda_k + \mu_k, \tag{3.10a}$$

où (d_k, μ_k) est solution du système linéaire

$$\begin{pmatrix} L_k & A_k^\mathsf{T} \\ A_k & 0 \end{pmatrix} \begin{pmatrix} d_k \\ \mu_k \end{pmatrix} = - \begin{pmatrix} \nabla f(x_k) + A_k^\mathsf{T} \lambda_k \\ c(x_k) \end{pmatrix}. \tag{3.10b}$$

On y a noté $L_k := \nabla^2_{xx} \ell(x_k, \lambda_k)$ et $A_k := c'(x_k)$.

2 Globalisation par recherche linéaire

On sait que la direction

$$p_k := (d_k, \mu_k) \in \mathbb{R}^{n+m} \tag{3.11}$$

est une direction de descente en $z_k := (x_k, \lambda_k)$ de la *fonction de moindres-carrés*

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

définie en $z = (x, \lambda)$ par

$$\varphi(z) := \frac{1}{2}\|F(z)\|_2^2. \tag{3.12}$$

De manière non classique (il y a une autre méthode, demandant d'en savoir plus sur les techniques de pénalisation, vues dans le cours OPT202), on peut forcer la convergence de l'algorithme de Newton (on dit qu'on le *globalise*) en faisant de la recherche linéaire sur φ. Au lieu de mettre à jour z_k par (3.10a), on le fait par

$$x_{k+1} := x_k + \alpha_k\, d_k \qquad \text{et} \qquad \lambda_{k+1} := \lambda_k + \alpha_k\, \mu_k, \tag{3.13}$$

où $p_k = (d_k, \mu_k)$ est toujours déterminé par (3.10b) et $\alpha_k > 0$ est un *pas* déterminé par recherche linéaire sur φ le long de p_k.

Il y a beaucoup de règles de recherche linéaire, mais la plus adaptée à l'algorithme de Newton est la *règle d'Armijo*, que l'on simplifie ici en n'incluant pas d'interpolation : on prend α_k de la forme 2^{-i_k}, où i_k est le plus petit entier positif tel que l'on ait

$$\varphi(z_k + \alpha_k p_k) \leqslant \varphi(z_k) + \omega\, \alpha_k\, \varphi'(z_k) \cdot p_k, \tag{3.14}$$

où ω est pris dans $]0, \frac{1}{2}[$ (typiquement $\omega = 10^{-4}$ pour que (3.14) ne soit pas trop contraignant).

3 Schéma de l'algorithme

Une itération de l'algorithme de Newton avec recherche linéaire sur φ pour résoudre le problème (P), permettant de passer de l'itéré primal-dual (x_k, λ_k) au suivant (x_{k+1}, λ_{k+1}), peut être schématisée par les étapes suivantes :

1) tester l'optimalité éventuelle de $z_k = (x_k, \lambda_k)$, avec arrêt si z_k est « satisfaisant »,
2) appel du simulateur pour calculer $f(x_k)$, $c(x_k)$, $\nabla f(x_k)$, $c'(x_k)$ et L_k, qui interviennent dans (3.10b) et (3.14),
3) résoudre le système linéaire (3.10b),
4) déterminer le pas $\alpha_k > 0$ par la recherche linéaire comme décrit autour de (3.14),
5) mettre à jour z_k par (3.13).

4 Implémentation

4.1 Le paramètre `options.rl`

On pourra construire un unique solveur `sqp`, avec une *option* définie par le paramètre d'entrée `options.rl` permettant de sélectionner soit l'algorithme de Newton, soit l'algorithme de Newton avec recherche linéaire (il est bien de sauvegarder sa version de l'algorithme sans recherche linéaire, toutefois):

```
options.rl = 0: avec recherche linéaire,
           = 1: avec pas unite.
```

4.2 Soigner la présentation des résultats

Pour que l'on sache ce qui se passe pendant la phase d'optimisation (elle permet de détecter les incohérences du simulateur), il faut que l'optimiseur parle à l'utilisateur du solveur. Voici une version minimale des sorties à chaque itération (`options.verb = 1`, une ligne par itération), permettant à un œil exercé de comprendre le comportement de l'optimiseur. Celles-ci sont obtenues par l'instruction `fprintf`.

```
-----------------------------------------------------------------------
iter     |gl|        |ce|       |x|     |lm|      alpha        phi
   1  2.4956e-01  7.7026e+00  1.5e+00  6.4e-01  3.125e-02  6.57603e+01
   2  2.9238e-01  2.0650e+00  9.7e-01  1.1e+00  1.000e+00  5.60225e+00
   3  4.4940e-01  1.5837e+00  7.4e-01  1.3e+00  5.000e-01  3.01134e+00
   4  5.5588e-01  1.6550e+00  8.6e-01  1.8e+00  5.000e-01  2.66321e+00
   5  3.2658e-01  1.6276e+00  7.2e-01  2.5e+00  2.500e-01  2.17794e+00
   6  3.4954e-01  1.5776e+00  7.5e-01  6.1e-01  3.125e-02  2.12390e+00
   7  7.6374e-01  1.4987e+00  1.1e+00  1.1e+00  2.500e-01  2.08650e+00
   8  6.7127e-01  8.8351e-01  1.3e+00  6.4e-01  1.000e+00  9.24207e-01
   9  6.3062e-01  3.8424e-01  8.5e-01  1.3e+00  1.000e+00  4.41996e-01
  10  2.1263e-01  1.4549e-01  9.6e-01  1.5e+00  1.000e+00  5.25454e-02
  11  8.1372e-02  8.1474e-02  9.4e-01  1.7e+00  5.000e-01  1.51907e-02
  12  8.7556e-02  4.4406e-02  9.3e-01  2.1e+00  5.000e-01  9.04604e-03
  13  7.5649e-02  2.3725e-02  9.3e-01  2.5e+00  5.000e-01  5.76332e-03
  14  6.9040e-02  3.4034e-03  9.3e-01  3.2e+00  1.000e+00  4.55810e-03
  15  3.1844e-03  8.4017e-05  9.3e-01  3.5e+00  1.000e+00  1.13268e-05
  16  2.0759e-06  2.8775e-08  9.3e-01  3.5e+00  1.000e+00  4.86858e-12
  17  1.9476e-12  3.1086e-14  9.3e-01  3.5e+00  1.000e+00  3.86589e-24
-----------------------------------------------------------------------
```

La première colonne donne le numéro `iter` $= k$ de l'itération, la seconde donne la norme `|gl|` $= \|\nabla_x \ell(x_k, \lambda_k)\|_\infty$ du gradient du lagrangien (qui doit tendre rapidement vers zéro asymptotiquement), la troisième donne la norme `|c|` $= \|c(x_k)\|_\infty$ des contraintes (qui doit également tendre rapidement vers zéro asymptotiquement), les quatre et cinquième colonnes donnent les normes `|x|` $= \|x_k\|_\infty$ et `|lm|` $= \|\lambda_k\|_\infty$ de x_k et λ_k respectivement, la sixième colonne est le pas `alpha` $= \alpha_k$ déterminé par la recherche linéaire et la dernière est la valeur `phi` $= \varphi(z_k)$ définie en (3.12).

Si l'on veut plus d'information (en particulier sur la recherche linéaire, c'est souvent utile), on pourra avoir un mode plus bavard (`options.verb = 2`), qui donne plus de détails à chaque itération comme dans l'affichage ci-dessous.

```
-------------------------------------------------------------
iter   6,  simul  15,  phi  2.17794e+00,  pente -4.35589e+00

  recherche lineaire d'Armijo: |d| =  6.46e+00
    alpha        phip-phi      DF(phi)
  1.0000e+00   1.27880e+05   1.27880e+05
  5.0000e-01   7.98544e+03   1.59709e+04
  2.5000e-01   4.96505e+02   1.98602e+03
  1.2500e-01   3.01036e+01   2.40829e+02
```

```
  6.2500e-02   1.52496e+00   2.43994e+01
  3.1250e-02  -5.40482e-02  -1.72954e+00

  |gl| = 3.495e-01,  |ce| = 1.578e+00
-------------------------------------------------------------
```

Quelques explications : `simul` donne le nombre d'appels au simulateur au début de la k-ième itération; `pente` est la valeur de la dérivée directionnelle $\varphi'(z_k) \cdot p_k$.

La recherche linéaire fournit une ligne par essai de pas α_k. On voit ici que le sixième pas a été accepté ($\alpha_k = 3.1250e\,10^{-2}$) ; c'est plutôt exceptionnel de faire autant d'essais de pas (nous avons bien choisi l'itération, voir le premier tableau), car le plus souvent le pas unité est accepté et cela devrait être le cas dans les dernièrees itérations (pourquoi ?). La signification des sorties est la suivante. La première colonne (`alpha`) donne le pas essayé α_k, la seconde (`phip-phi`) donne

$$\varphi(z_k + \alpha_k p_k) - \varphi(z_k)$$

(que l'on veut rendre négatif) et la dernière (`DF(phi)`) donne l'estimation de cette pente par différences finies :

$$\frac{\varphi(z_k + \alpha_k p_k) - \varphi(z_k)}{\alpha_k},$$

Cette dernière valeur permet de vérifier la valeur de $\varphi'(z_k) \cdot p_k$ lorsque les pas essayés tendent vers zéro (ce qui n'est pas une situation recherchée...).

5 Questions

3.1. Pourquoi la direction p_k donnée par (3.11) est elle une direction de descente en z_k de la fonction φ définie en (3.12) ?

3.2. Dans la règle de recherche linéaire d'Armijo décrite autour de (3.14), pourquoi essaye t-on d'abord le pas $\alpha_k = 1$ (en prenant $i_k = 0$ en premier) ? Est-on sûr de trouver un pas $\alpha_k > 0$ vérifiant l'inégalité (3.14) ?

6 Cas-tests

- **Cas-test 2d :** Reprendre le cas-test 2d avec de la recherche linéaire et comparer avec les résultats obtenus avec le pas unité.
- **Cas-test 3a :** 2 barres de longueurs 0.6 et 0.6. Deuxième point de fixation de la chaîne : $(a, b) = (1, 0)$. Position initiale du nœud : $(0.5, 0.4)$.
- **Cas-test 3b :** 2 barres de longueurs 2 et 1. Deuxième point de fixation de la chaîne : $(a, b) = (1, 0)$. Positions initiales des nœuds : $(0.5, 0.3)$.
- **Cas-test 3c :** 2 barres de longueurs 2 et 1. Deuxième point de fixation de la chaîne : $(a, b) = (0, -1)$. Positions initiales des nœuds : $(0.3, 0.3)$.