

Projet d'optimisation en Matlab

Équilibre d'une chaîne articulée

Séance 4 : Prise en compte de contraintes d'inégalité

Dans cette séance, on souhaite améliorer le solveur de problème d'optimisation en lui donnant la possibilité de prendre en compte des contraintes d'inégalité. Cela permettra par exemple de trouver la position d'équilibre de la chaîne en présence d'un plancher.

1 Une chaîne au-dessus d'un plancher

On veut à présent prendre en compte la présence d'un plancher convexe linéaire par morceaux que l'on exprime comme l'enveloppe supérieure d'un nombre *fini* p de fonctions affines. La chaîne, qui doit rester au-dessus de ce plancher, ne pourra donc pas, en général, prendre sa position d'équilibre précédente. On suppose le plancher infiniment glissant. Il n'y a donc pas lieu d'introduire un modèle décrivant le contact entre la chaîne et le plancher. Ceci se traduit par le fait que l'on cherche toujours la position d'énergie potentielle minimale, la présence du plancher se traduisant simplement par l'ajout de contraintes d'inégalité portant sur la position des nœuds.

Le plancher est supposé décrit dans le plan (x, y) , au moyen d'une fonction affine $\varphi : \mathbb{R} \rightarrow \mathbb{R}^p$ définie par

$$\varphi(x) = r + xs.$$

où les *vecteurs* $r \in \mathbb{R}^p$ et $s \in \mathbb{R}^p$ sont donnés mais pourront varier en dimension et en valeur d'un cas-test à l'autre. Tous les points de la chaîne doivent se trouver dans le convexe

$$C = \{(x, y) \in \mathbb{R} \times \mathbb{R} : ye \geq \varphi(x)\},$$

où, sous forme matricielle, $e := (1 \ 1 \ \dots \ 1)^T \in \mathbb{R}^p$. On a représenté à la figure 3 la po-

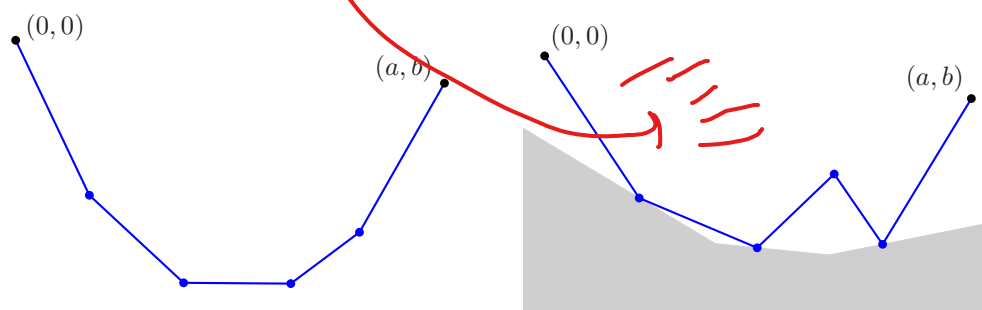


Figure 3: Positions d'équilibre de la chaîne, sans et avec plancher

sition d'équilibre d'une chaîne, *sans* (à gauche) et *avec* (à droite) cette contrainte (dans ce cas, $p = 3$). On supposera que les points de fixation $(0,0)$ et (a,b) se trouvent dans le domaine C , ce qui revient à supposer que les vecteurs r et s (des données) vérifient les inégalités vectorielles

$$r \leq 0 \quad \text{et} \quad r + as \leq be.$$

(a,b) > plancher

Comme la chaîne est affine entre ses nœuds et que le plancher est convexe, elle sera entièrement contenue dans C si tous ses nœuds s'y trouvent, c'est-à-dire si pour tout $i \in [1 : n_n]$ et tout $j \in [1 : p]$, on a

$$c_{n_b+(j-1)n_n+i}(x, y) \equiv r_j + x_i s_j - y_i \leq 0.$$

Le problème à résoudre est donc de la forme

$$(P_2) \quad \begin{cases} \min e(x, y) \\ c_i(x, y) = 0, & i \in [1 : n_b] \\ c_i(x, y) \leq 0, & i \in [n_b + 1 : n_b + pn_n], \end{cases}$$

où les contraintes d'égalité (les n_b premières) concernent la longueur des barres et les contraintes d'inégalité (les suivantes) expriment la présence du plancher.

2 Modification du simulateur

Le simulateur doit à présent fournir à l'optimiseur les informations décrivant les contraintes d'inégalité. Les modifications à apporter sont les suivantes :

- en entrée, on trouvera donc en plus un multiplicateur associé à ces contraintes d'inégalité,
- en sortie, il faudra calculer $c_I(x)$ (dans **ci**), $c'_I(x)$ (dans **ai**) et peut-être modifier le calcul du hessien du lagrangien (dans **hl**).

Le simulateur pourra donc se présenter comme suit:

$$\text{function } [\underline{\mathbf{e}}, \underline{\mathbf{ce}}, \underline{\mathbf{ci}}, \mathbf{g}, \underline{\mathbf{ae}}, \underline{\mathbf{ai}}, \mathbf{hl}, \text{indic}] = \text{chs}(\text{indic}, \mathbf{xy}, \underline{\mathbf{lme}}, \underline{\mathbf{lmi}})$$

En entrée :

indic : pilote le comportement du simulateur :

- = 1 : **chs** fait un tracé de la chaîne;
- = 2 : **chs** calcule **e**, **ce** et **ci**;
- = 4 : **chs** calcule **e**, **ce**, **ci**, **g**, **ae** et **ai**;
- = 5 : **chs** calcule **hl**.

xy : vecteur-colonne contenant d'abord les abscisses $\{x_i\}_{i=1}^{n_n}$ des nœuds, puis leurs ordonnées $\{y_i\}_{i=1}^{n_n}$, i.e., $\mathbf{xy} = (x_1, \dots, x_{n_n}, y_1, \dots, y_{n_n})$. C'est le vecteur x à optimiser.

lme : vecteur-colonne de dimension n_b contenant les multiplicateurs de Lagrange pour les contraintes d'égalité, $\lambda_E = \{\lambda_i\}_{i=1}^{n_b}$ à utiliser dans le calcul du hessien **hl**.

lmi : vecteur-colonne de dimension n_n contenant les multiplicateurs de Lagrange pour les contraintes d'inégalité, $\lambda_I = \{\lambda_i\}_{i=n_b+1}^{n_b+n_n}$ à utiliser (éventuellement) dans le calcul du hessien **hl**.

En sortie :

- e** : valeur de l'énergie potentielle en x : $-\mathbf{xy}$ (c'est-à-dire pour les nœuds dont les coordonnées sont dans **xy**).
- ce** : valeur en **xy** des contraintes sur la longueur des barres, **ce** est un vecteur-colonne de dimension n_b , que nous noterons c_E .
- ci** : contient la valeur des pn_n contraintes d'inégalité, c'est donc un vecteur-colonne de dimension pn_n , que nous noterons mathématiquement c_I .
- g** : gradient de e en **xy** ; c'est un vecteur-colonne de dimension $n := 2n_n$; on notera $g = \nabla e(x)$.

- ae**: jacobienne des contraintes d'égalité en **xy**, matrice de n_b lignes et $2n_n$ colonnes, que nous noterons $A_E(x) = c'_E(x)$.
ai: jacobienne des contraintes d'inégalité en **xy**, matrice de pn_n lignes et $2n_n$ colonnes, que nous noterons $A_I(x) = c'_I(x)$.
hl: hessien du lagrangien en **xy**, **lme** et **lmi**:

$$\nabla_{xx}^2 \ell(\lambda) = \underbrace{\nabla^2 e(x)}_{=0} + \sum_{i \in E} \lambda_i \nabla^2 c_i(x) + \underbrace{\sum_{i \in I} \lambda_i \nabla^2 c_i(x)}_{=0}$$

(les λ_i sont dans **lme** et **lmi**).

- 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).

Il y a deux variables globales en plus:

- **R**: r , vecteur correspondant au plancher $\varphi(x) = r + s x$.
- **S**: s , idem.

3 Modification de l'optimiseur

3.1 L'optimisation quadratique successive

Le problème (P_2) est un problème d'optimisation avec contraintes d'égalité et d'inégalité de la forme

$$(P_{EI}) \quad \begin{cases} \min f(x) \\ c_E(x) = 0 \\ c_I(x) \leq 0, \end{cases} \quad (4.1)$$

où $x \in \mathbb{R}^n$ est le couple (x, y) des abscisses $x \in \mathbb{R}^{n_n}$ et ordonnées $y \in \mathbb{R}^{n_b}$ 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 , $c_E(x)$ donne la valeur des contraintes sur la longueur des barres et $c_I(x)$ donne la valeur des contraintes définissant le plancher convexe au-dessus duquel doit se trouver la chaîne. On a $E = [1 : n_b]$ et $I = [n_b + 1 : n_b + pn_n]$, si le plancher est défini par l'enveloppe supérieure de p fonctions affines. On note $m_E = |E| = n_b$, $m_I = |I| = pn_n$ et $m = m_E + m_I$.

Le lagrangien du problème (P_{EI}) s'écrit :

$$\ell(x, \lambda) = f(x) + \lambda^T c(x),$$

où $\lambda = (\lambda_E, \lambda_I) \in \mathbb{R}^m$, avec $\lambda_E \in \mathbb{R}^{|E|}$ et $\lambda_I \in \mathbb{R}^{|I|}$, et $c(x)$ est le couple $(c_E(x), c_I(x)) \in \mathbb{R}^m$.

On se propose d'améliorer l'optimiseur local construit précédemment (celui sans recherche linéaire de la séance 2) en remplaçant la méthode Newton (qui fonctionne pour les problèmes avec contraintes d'égalité seulement) par un algorithme connu sous le nom d'*optimisation quadratique successive* (OQS ou SQP en anglais pour *Sequential Quadratic Programming*).

Une itération de cet algorithme remplace la résolution du système linéaire de la méthode de Newton par la résolution d'un problème d'optimisation quadratique. Il génère toujours une suite primale-duale $\{(x_k, \lambda_k)\} \subseteq \mathbb{R}^n \times \mathbb{R}^m$ de la manière suivante. Au début de

l'itération k , on connaît (x_k, λ_k) . On calcule alors une solution primale-duale $(d_k, \lambda_k^{\text{PQ}})$ du problème quadratique osculateur

$$\begin{cases} \min_{d \in \mathbb{R}^n} \nabla f(x_k)^\top d + \frac{1}{2} d^\top M_k d \\ c_E(x_k) + c'_E(x_k) d = 0 \\ c_I(x_k) + c'_I(x_k) d \leq 0. \end{cases} \quad (4.2)$$

et M_k le hessien du lagrangien $\nabla_{xx}^2 \ell(x_k, \lambda_k)$ ou une approximation de celui-ci (voir ci-dessous). Le nouvel itéré est alors

$$x_{k+1} = x_k + d_k \quad \text{et} \quad \lambda_{k+1} = \lambda_k^{\text{PQ}}.$$

Le nom d'*optimisation quadratique successive* vient de ce qu'à chaque itération, on résout le problème quadratique (4.2).

3.2 Modification définie positive du hessien

Il y a de bonnes raisons de prendre pour M_k dans (4.2) une *approximation* définie positive du hessien du lagrangien $H_k := \nabla_{xx}^2 \ell(x_k, \lambda_k)$.

- Le problème quadratique osculateur (4.2) est alors beaucoup plus facile à résoudre (certains algorithmes peuvent alors le résoudre en un nombre polynomial d'itérations, alors qu'il est NP-ardu si $M_k \not\geq 0$) et a au plus une solution (il peut encore être non réalisable cependant).
- La solution d_k du problème quadratique osculateur (4.2) est alors une direction de descente d'une « fonction de mérite bien choisie », ce qui permet la globalisation de l'algorithme OQS/SQP par recherche linéaire.
- On perd la convergence quadratique de l'algorithme de Newton, mais on peut toutefois avoir une vitesse de convergence superlinéaire, pourvu que M_k approxime H_k « dans les bonnes directions ».

Les deux méthodes les plus couramment utilisées pour remplacer H_k par une matrice définie positive sont les *techniques de quasi-Newton* et l'approximation de H_k par sa *factorisation de Cholesky modifiée*. Les techniques quasi-newtoniennes seront vues dans une autre séance. Nous utilisons ici la factorisation de Cholesky modifiée.

Observons d'abord que la factorisation de Cholesky de H_k ne fonctionnera pas si $H_k \not\geq 0$, car on ne peut pas écrire $H_k = L_k L_k^\top$ dans ce cas ($L_k L_k^\top \geq 0$). La technique consiste alors à modifier H_k au cours de sa factorisation de Cholesky et d'en déduire une matrice E_k (un écart, une erreur) telle que

$$H_k + E_k = L_k D_k L_k^\top,$$

où E_k est diagonale semi-définie positive, L_k est triangulaire inférieure et D_k est diagonale définie positive. On prend alors $M_k := L_k D_k L_k^\top$ dans le problème quadratique osculateur (4.2).

Cette opération de factorisation de Cholesky modifiée est réalisée par le code `cholmod`, soit en ajoutant le chemin

`addpath ~jgilbert/qpalm`

soit par téléchargement à partir du site pédagogique du cours ou encore directement à l'adresse

<https://who.rocq.inria.fr/Jean-Charles.Gilbert/ensta/cours2a/notes/cholmod.m>

Le fonctionnement de ce code est expliqué par le `help` en ligne (entrer «`help cholmod`» dans la fenêtre de commande de `Matlab`). On prendra typiquement

```
small = 1.e-5;  
big   = 1.e+5;
```

3.3 Résolution du problème quadratique osculateur

La difficulté principale de l'algorithme OQS/SQP local (i.e., sans technique de globalisation) provient de la résolution du problème quadratique osculateur (4.2). Développer un tel code peut prendre plusieurs mois à un ingénieur bien guidé; il n'est donc pas question ici de se lancer dans une telle aventure. Dans le cadre de ce projet, le problème quadratique osculateur (4.2) pourra être résolu, au choix, par les solveurs `Quadprog` ou `Qpalm`:

- `Quadprog` est le solveur de problème quadratique standard de `Matlab`; il fait partie de la boîte à outils d'optimisation et on peut l'appeler directement;
- `Qpalm` est un solveur de problèmes quadratiques convexes utilisant l'algorithme du lagrangien augmenté [2, 1] et permettant de traiter correctement les problèmes non réalisables et/ou non bornés; son niveau de maturité est moindre que `Quadprog`, mais il offre donc davantage de possibilités; pour y avoir accès, ajouter un répertoire au chemin de recherche

```
addpath ~jgilbert/qpalm
```

ou le télécharger à partir du site pédagogique du cours ou encore directement à l'adresse

<https://who.rocq.inria.fr/Jean-Charles.Gilbert/ensta/cours2a/notes/qpalm.zip>

Le fonctionnement de ces solveurs est donné par le `help` en ligne (entrer «`help quadprog`» ou «`help qpalm`» dans la fenêtre de commande de `Matlab`).

3.4 Pseudo-code de l'OQS/SQP

Nous donnons ci-dessous le pseudo-code de l'algorithme à implémenter. On donnera dans la section suivante quelques conseils de mise en œuvre.

Input:

L'itéré initial (x_1, λ_1) .

Les valeurs $f(x_1)$, $c_E(x_1)$, $c_I(x_1)$, $\nabla f(x_1)$, $c'_E(x_1)$, $c'_I(x_1)$ et $M_1 \succ 0$.

Une tolérance $\varepsilon > 0$.

Output: (x_*, λ_*) une solution primale-duale approchée.

Soit $k := 1$

loop

if test d'arrêt vérifié **then**

$(x_*, \lambda_*) := (x_k, \lambda_k)$.

stop

end if

 Calculer une solution primale-duale $(d_k, \lambda_k^{\text{PQ}})$ du problème (4.2).

$x_{k+1} := x_k + d_k = x_k + d_k$
 $\lambda_{k+1} := \lambda_k^{\text{PQ}}$

$\theta = 2$

c_I c_I g c_E q_i M

Calculer ~~$f(x_{k+1})$~~ , $c_E(x_{k+1})$, $c_I(x_{k+1})$, $\nabla f(x_{k+1})$, $c'_E(x_{k+1})$, $c'_I(x_{k+1})$ et M_{k+1} .
Approcher $\nabla_{xx}^2 \ell(x_{k+1}, \lambda_{k+1})$ par une matrice définie positive M_{k+1} .
 $k := k + 1$
end loop

h_l M

Le test d'arrêt utilisé dans l'algorithme pourra être

$$\max \{ \|\nabla_x \ell(x_k, \lambda_k)\|_\infty, \|c_E(x_k)\|_\infty, \|\min((\lambda_k)_I, -c_I(x_k))\|_\infty \} < \varepsilon. \quad (4.3)$$

3.5 Écriture de l'optimiseur

On rappelle que l'optimiseur doit être écrit indépendamment du problème à traiter (ici celui de la position d'équilibre d'une chaîne), les seules informations sur le problème communiquées à l'optimiseur se faisant via le simulateur `chs` (voir la section 2).

On écrira l'optimiseur sous la forme d'une fonction Matlab appelée `sqp` (à mettre dans un fichier `sqp.m`) et ayant la structure suivante :

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

En entrée :

`simul` : nom du simulateur (chaîne de caractères), dont la séquence d'appel a été décrite à la séance 1;

`x` : vecteur contenant la valeur initiale x_1 des variables à optimiser;

`lme` : vecteur contenant la valeur initiale $(\lambda_1)_E$ des multiplicateurs des contraintes d'égalité;

`lmi` : vecteur contenant la valeur initiale $(\lambda_1)_I$ des multiplicateurs;

`options` structure spécifiant les paramètres de fonctionnement de l'algorithme.

`options.tol` : vecteur donnant deux seuils de tolérance pour l'optimalité ; l'optimiseur considérera que l'optimum est atteint si l'on a en (x_k, λ_k) :

$$\begin{aligned} \|\nabla_x \ell(x_k, \lambda_k)\|_\infty &\leq \text{options.tol}(1), \\ \|c_E(x_k)\|_\infty &\leq \text{options.tol}(2), \\ \|\min((\lambda_k)_I, -c_I(x_k))\|_\infty &\leq \text{options.tol}(3), \end{aligned}$$

où $\|\cdot\|_\infty$ est la norme ℓ_∞ ;

`options.maxit` : entier donnant le maximum d'itérations autorisé (l'optimiseur peut donc s'arrêter, éventuellement sans avoir trouvé une solution).

En sortie :

`x` : vecteur contenant la valeur finale x_* des variables à optimiser, lors de l'arrêt de l'optimiseur;

`lme` : vecteur contenant la valeur finale $(\lambda_*)_E$ des multiplicateurs associés aux contraintes d'égalité, lors de l'arrêt de l'optimiseur;

`lmi` : vecteur contenant la valeur finale $(\lambda_*)_I$ des multiplicateurs associés aux contraintes d'inégalité, lors de l'arrêt de l'optimiseur;

`info` : structure donnant des informations sur le comportement du solveur ; on pourra envisager les informations suivantes :

`info.status` décrit ce que l'algorithme a réussi à faire :

$= 0$: terminaison normale (seuil d'optimalité atteint);
 $= 1$: inconsistance des arguments d'entrée;
 $= 2$: terminaison sur le maximum d'itérations autorisé (`options.maxit`);
`info.iter` donne le nombre d'itérations effectuées.

4 Questions

- 4.1. La section 3.2 propose une méthode pour obtenir une matrice M_k définie positive. Pourquoi ne prend-on pas simplement $M_k = I$ (l'identité)? Si vous ne voyez pas pourquoi, essayez de résoudre un problème (par exemple le cas-test 4.a) en prenant $M_k = I$ au lieu de $M_k = H_k + E_k$, comme à la section 3.2.
- 4.2. Trouvez un multiplicateur initial, plus efficace que $\lambda_0 = 0$, de telle sorte que si l'utilisateur donne au solveur un point initial primal x_0 qui est une « solution », l'algorithme trouve le multiplicateur optimal, sans faire d'itérations.
- 4.3. En utilisant les conditions d'optimalité du second ordre pour les problèmes avec contraintes d'égalité, trouvez un critère permettant de déterminer si le point limite obtenu ne peut pas être un minimum local.
- 4.4. Dans le cas-test 4a, d'où vient le comportement différent de votre solveur (qui, pour ce cas-test, devrait trouver le minimum global) par rapport au solveur sans contrainte d'inégalité (qui ne trouvait qu'un point stationnaire, qu'il soit utilisé avec ou sans recherche linéaire)?
- 4.5. Observez-vous toujours la convergence quadratique? Avez-vous une explication si ce n'est pas le cas?

5 Cas-tests

- **Cas-test 4.a:** on reprend le second cas-test sans plancher de la séance 2, à savoir

$L = [0.7 \ 0.5 \ 0.3 \ 0.2 \ \text{et} \ 0.5]'$;

deuxième point de fixation de la chaîne: $(a, b) = (1, -1)$; position initiale des nœuds:

$xy = [\begin{array}{cccc} 0.2 & 0.4 & 0.6 & 0.8 \dots \\ 1 & 1.5 & 1.5 & 1.3 \end{array}]'$;

- **Cas-test 4.b:** 10 barres de longueur

$L = [0.2 \ 0.2 \ 0.2 \ 0.3 \ 0.3 \ 0.5 \ 0.2 \ 0.2 \ 0.3 \ 0.1]'$;

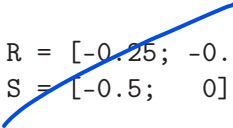
Deuxième point de fixation de la chaîne: $(a, b) = (1, 0)$. Position initiale des nœuds:

$xy = [\begin{array}{cccccccccc} 0.1 & 0.2 & 0.3 & 0.4 & 0.5 & 0.6 & 0.7 & 0.8 & 0.9 & \dots \\ -0.5 & -0.9 & -1.2 & -1.4 & -1.5 & -1.4 & -1.2 & -0.9 & -0.5 & \end{array}]'$;

Plancher:

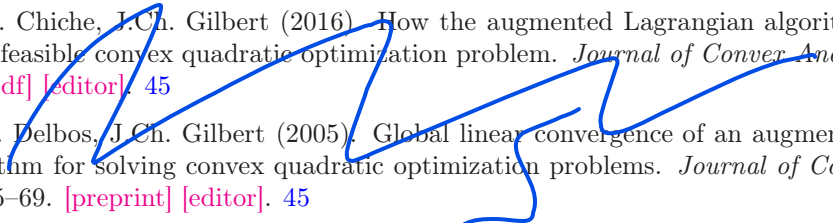
$R = -0.25;$
 $S = -0.5;$

- **Cas-test 4.c:** même cas-test que le 4.b, mais avec le plancher suivant:



$R = [-0.25; -0.5];$
 $S = [-0.5; 0];$

Références

- 
- [1] A. Chiche, J.Ch. Gilbert (2016). How the augmented Lagrangian algorithm can deal with an infeasible convex quadratic optimization problem. *Journal of Convex Analysis*, 23(2), 425–459. [\[pdf\]](#) [\[editor\]](#). 45
- [2] F. Delbos, J.Ch. Gilbert (2005). Global linear convergence of an augmented Lagrangian algorithm for solving convex quadratic optimization problems. *Journal of Convex Analysis*, 12(1), 45–69. [\[preprint\]](#) [\[editor\]](#). 45
- 