Projet d'optimisation en Matlab

Équilibre d'une chaîne articulée

Séance 2 : Méthode de Newton pour problème d'optimisation avec contraintes d'égalité

On reprend le problème de trouver une position d'équilibre d'une chaîne au repos. Le problème consiste à minimiser l'énergie potentielle de la chaîne sous des contraintes d'égalité imposant que les distances entre les nœuds de la chaîne sont égales à des distances prescrites. Efforçons-nous de raisonner de manière abstraite, c'est-à-dire en ne considérant que la structure du problème à résoudre. Structurellement, le problème se simplifie en

$$\begin{cases} \min f(x) \\ c(x) = 0, \end{cases} \tag{2.1}$$

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 $c(x)$ donne la valeur des contraintes sur la longueur des barres. On a $c : \mathbb{R}^n \to \mathbb{R}^m$ avec $m = n_b$.

On peut facilement résoudre ce problème (localement) par une méthode newtonienne, laquelle ne requiert (pour ce problème) que la résolution d'un système linéaire par itération.

1 La méthode de Newton pour résoudre un système d'équations non linéaires

Dans une de ses formes les plus simples, la méthode de Newton peut être vue comme un algorithme conçut pour trouver un zéro d'une fonction $F : \mathbb{R}^N \to \mathbb{R}^N$, c'est-à-dire un point $z_* \in \mathbb{R}^N$ tel que

$$F(z_*) = 0.$$

Il s'agit donc de « résoudre » un système de N équations *non linéaires* à N inconnues. L'algorithme procède par approximations successives : étant donné une approximation $z_k \in \mathbb{R}^N$ de z_*, une nouvelle approximation z_{k+1} est calculée comme suit

$$z_{k+1} := z_k + d_k,$$

où d_k est solution du système linéaire

$$F'(z_k)d_k = -F(z_k).$$

Cet algorithme converge quadratiquement dans le voisinage d'un zéro *régulier* z_* de F : si F est de classe C^1 dans un voisinage de z_* (hypothèse de lissité) et si $F'(z_*)$ est inversible (hypothèse de régularité), alors il existe un voisinage V de z_* tel que, si le premier itéré $z_1 \in V$, l'algorithme de Newton génère une suite $\{z_k\}_{k\geqslant 1} \subseteq V$ qui converge quadratiquement vers z_*, ce qui veut dire que

$$\exists\, C > 0, \quad \forall\, k \geqslant 1 : \qquad \|z_{k+1} - z_*\| \leqslant C\|z_k - z_*\|^2.$$

2 Méthode de Newton pour résoudre un problème d'optimisation avec contraintes d'égalité

Trouver un *point stationnaire* (c'est moins bien qu'un minimum local ou global, mais c'est plus facile et déjà assez difficile) du problème (2.1) consiste à trouver une solution primale-duale $(x_*, \lambda_*) \in \mathbb{R}^n \times \mathbb{R}^m$ de son système d'optimalité du premier ordre (ses conditions de Lagrange) :

$$\begin{cases} \nabla f(x_*) + c'(x_*)^T \lambda_* = 0 \\ c(x_*) = 0. \end{cases} \tag{2.2}$$

On se ramène donc au problème de trouver un zéro $z_* := (x_*, \lambda_*) \in \mathbb{R}^N := \mathbb{R}^n \times \mathbb{R}^m$ d'un système de N équations à N inconnues, pour lequel l'algorithme de Newton est bien adapté. On retrouve le cadre de la section 1 en introduisant l'application $F : \mathbb{R}^N \to \mathbb{R}^N$ définie en $z := (x, \lambda) \in \mathbb{R}^N$ par

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

Sa dérivée s'écrit

$$F'(z) := \begin{pmatrix} \nabla^2_{xx} \ell(x, \lambda) & c'(x)^T \\ c'(x) & 0 \end{pmatrix}.$$

Une itération de Newton en $z_k := (x_k, \lambda_k)$ consiste donc à calculer la solution du système linéaire

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

où l'on a simplifié les notations en introduisant $L_k := \nabla^2_{xx} \ell(x_k, \lambda_k)$ et $A_k = c'(x_k)$, et ensuite à prendre comme nouvel itéré le couple $z_{k+1} := (x_{k+1}, \lambda_{k+1})$ défini par

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

Si l'on introduit $\lambda_k^{\text{PQ}} := \lambda_k + \mu_k$, on peut légèrement simplifier l'itération : le système linéaire à résoudre devient

$$\begin{pmatrix} L_k & A_k^T \\ A_k & 0 \end{pmatrix} \begin{pmatrix} d_k \\ \lambda_k^{\text{PQ}} \end{pmatrix} = - \begin{pmatrix} \nabla f(x_k) \\ c(x_k) \end{pmatrix}, \tag{2.3}$$

et le nouvel itéré est obtenu par

$$x_{k+1} := x_k + d_k \qquad \text{et} \qquad \lambda_{k+1} := \lambda_k^{\text{PQ}}. \tag{2.4}$$

Il s'agit donc d'un algorithme *primal-dual*, c'est-à-dire un algorithme dans lequel l'itération met à jour les variables primale x_k et duale λ_k *indépendamment*.

On utilisera cet algorithme (2.3)-(2.4) pour trouver la position d'équilibre de la chaîne. On voit qu'il est très simple à mettre en œuvre, dès que l'on dispose des dérivées premières et secondes des fonctions définissant le problème.

3 Implémentation

3.1 Structure du code de résolution

Classiquement, un code utilisant un solveur (de problèmes d'optimisation par exemple, mais pas seulement) peut être structuré de deux manières différentes : soit au moyen de

la *communication directe* (plus simple à mettre en œuvre et expliquée ci-dessous), soit au moyen de la *communication inverse* (plus complexe à mettre en œuvre, mais présentant quelques avantages, voir la section 5.6.1 du syllabus).

En *communication directe*, le code est structuré comme à la figure 2. Les parties en bleu

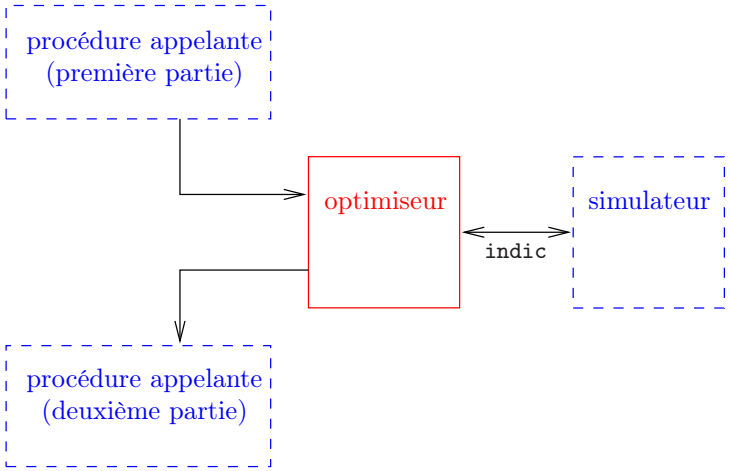

Figure 2: Shéma d'un code d'optimisation en communication directe

(pointillé) sont écrites par ceux qui connaissent l'application, tandis que la partie en rouge (continu), représentant le solveur d'optimisation, est généralement écrite par des numériciens généralistes, *indépendamment du problème considéré*. Dans la partie utilisateur, on s'exprime en termes du problème à résoudre (ici le problème de la chaîne articulée), tandis que dans le solveur, on s'exprime de manière abstraite, en termes mathématiques comme dans les sections 1 et 2 (parce que le solveur doit pouvoir être utilisé pour résoudre des problèmes très différents).

Dans la première séance on s'est attaché à écrire les parties en bleu. Dans cette séance, on écrira le solveur newtonien pour problèmes d'optimisation avec contraintes d'égalité expliqué à la section 2 (partie en rouge). On veillera à ce que l'optimiseur ne travaille que sur des variables « abstraites » (c'est-à-dire x, $f(x)$, $c(x)$, ...). On appellera

- `sqp` le code d'optimisation (pour une raison qui sera clarifiée ultérieurement),
- `chs` le simulateur.

3.2 Écriture de l'optimiseur

Comme l'optimiseur `sqp` 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 fait par l'intermédiaire du simulateur `chs` écrit lors de la première séance.

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

En entrée :

`simul` : spécification du simulateur, voir le point 5 de la section 3.3 ;
`x` : vecteur contenant la valeur initiale $x_1 \in \mathbb{R}^n$ des variables à optimiser ;
`lm` : vecteur contenant la valeur initiale $\lambda_1 \in \mathbb{R}^m$ des variables duales ;
`options` structure spéciafiant les paramètres de fonctionnement de l'algorithme.

`options.tol` : vecteur donnant deux seuils de tolérance pour l'optimalité (`options.tol(1)` et `options.tol(2)` doivent être dans l'intervalle ouvert $]0, 1[$) ; l'optimiseur considérera que l'optimum est atteint si l'on a en (x_k, λ_k) :

$$\|\nabla_x \ell(x_k, \lambda_k)\|_\infty \leqslant \texttt{options.tol(1)},$$
$$\|c(x_k)\|_\infty \leqslant \texttt{options.tol(2)},$$

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 la solution).

En sortie :

`x` : vecteur contenant la valeur finale x_k des variables à optimiser, lors de l'arrêt de l'optimiseur ;
`lm` : vecteur contenant la valeur finale λ_k des variables duales, 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 résussi 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.niter` donne le nombre d'itérations effectuées.

3.3 Conseils

Voici quelques conseils permettant de mener à bien l'écriture de l'optimiseur.

1. Dans Matlab, la solution $x = A^{-1}b$ d'un système linéaire régulier $Ax = b$ ne se calcule jamais par les instructions `x=inv(A)*b` ou `x=A^(-1)*b`. Elles sont trop coûteuses. On utilisera `x=A\b`.

2. Essayez d'aller au plus vite vers un optimiseur calculant la solution du problème, sans s'occuper des détails qui sont signes d'une implémentation soignée.
   - On pourra vérifier la valeur des variables au cours des itérations en enlevant le « ; » en fin d'instruction, mais dès que le solveur se stabilise, faire des sorties formatées sur écran avec `fprintf`.
   - L'arrêt de l'optimisation se fera sur le nombre d'itérations, que l'on augmentera progressivement jusqu'à ce que l'arrêt puisse se faire sur les tolérances données dans `option.tol`.

3. Notez que, dans `sqp`, les dimensions n et m du problème peuvent s'obtenir à partir des variables d'entrée de `sqp` ($n =$ `length(x)` et $m =$ `length(lm)`, à moins que `lm` soit vide...) ou après appel au simulateur.

4. Pour suivre le déroulement de l'optimisation, il est essentiel d'imprimer la valeur d'un certain nombre de variables au cours des itérations. Par exemple : le numéro k de l'itération, la valeur de $f(x_k)$, $\|c(x_k)\|_\infty$, $\|\nabla_x \ell(x_k, \lambda_k)\|_\infty$, ... Faire un tracé de la chaîne à chaque itération sera également utile (appel du simulateur avec `indic` $= 1$).

5. Il y a (au moins) deux manières de procéder pour informer `sqp` que le nom du simulateur est `chs`.

   – On appelle `sqp` par

   ```
   [...] = sqp('chs',...);
   ```

   et, dans `sqp`, l'appel au simulateur se fera par

   ```
   [...] = feval(simul,2,x);
   ```

   – On appelle `sqp` par

   ```
   [...] = sqp(@chs,...);
   ```

   et, dans `sqp`, l'appel au simulateur se fera directement par

   ```
   [...] = simul(2,x);
   ```

6. En cas de non convergence suspecte, on construira un cas-test dont on connaît la solution (par exemple une chaîne formée de deux barres de longueur 5 dont les extrémités sont distantes de 8 sur une horizontale). On analyse alors le comportement de l'algorithme lorsqu'il démarre en la solution ou dans le voisinage de celle-ci. On peut aussi modifier le système linéaire (2.3) en annulant $\nabla f(x_k)$ et en remplaçant L_k par l'identité : dans ce cas, on ne cherche qu'à annuler les contraintes et un bon comportement sera signe que le calcul des contraintes et de leur jacobienne est correct (il faudra alors vérifier si $\nabla f(x_k)$ et L_k sont calculés correctement dans le simulateur ou revoir les opérations dans l'optimiseur).

4 Cas-tests

Cas-test 2a : 5 barres de longueur 0.7, 0.5, 0.3, 0.2 et 0.5. Deuxième point de fixation de la chaîne : $(a, b) = (1, -1)$. Position initiale des nœuds : $(0.2, -1)$, $(0.4, -1.5)$, $(0.6, -1.5)$ et $(0.8, -1.3)$.

Cas-test 2b : Idem que le cas-test 2a, mais on prend comme position initiale des nœuds : $(0.2, 1)$, $(0.4, 1.5)$, $(0.6, 1.5)$ et $(0.8, 1.3)$.

Cas-test 2c : Idem que le cas-test 2a, mais on prend comme position initiale des nœuds : $(0.2, -1)$, $(0.4, -1.5)$, $(0.6, 1.5)$ et $(0.8, -1.3)$.

Cas-test 2d : Idem que le cas-test 2a, mais on prend comme position initiale des nœuds : $(0.2, 1)$, $(0.4, -1.2)$, $(0.6, 1.5)$ et $(0.8, -1.3)$.

5 Questions à se poser

1. L'algorithme converge t'il ?
2. Le comportement de l'optimiseur vous paraît-il compatible avec la théorie ?
3. En cas de convergence, celle-ci est-elle quadratique ?
   Dans l'affirmative, est-ce vers un minimum, un maximum, un point-selle ?
   Dans la négative, est-ce anormal ?

6 Questions

2.1. Comment peut-on évaluer la vitesse de convergence de z_k vers sa limite z_* en examinant le comportement de

$$\|F(z_k)\|_\infty = \max(\|\nabla_x \ell(x_k, \lambda_k)\|_\infty, \|c(x_k)\|_\infty)$$

au cours des itérations ? Quel est l'intérêt de $\|F(z_k)\|_\infty$ par rapport à $\|z_k - z_*\|$?

2.2. La première fois que le hessien du lagrangien est calculé dans `sqp`, il doit l'être avec un certain multiplicateur λ_1. Supposons que l'utilisateur appelle `sqp` avec `lm` valant `[]`, ce qui signifie qu'il n'a aucune idée de la valeur de celui-ci. Trouvez un moyen d'estimer λ_1 à partir de x_1, de telle sorte que si x_1 est un point stationnaire du problème, le solveur trouvera le bon λ_1 et n'itérera pas.

Il y a plusieurs possibilités.

7 Rapport

1. Pour chaque cas-test, on mentionnera :

   – s'il y a convergence ;
   – le nombre d'itérations requis ;
   – le type de « solution » trouvée : un minimum, un maximum, un point-selle.

   Justifiez vos réponses.

2. Pour le cas-test 2a, évaluez la vitesse de convergence de l'algorithme en utilisant $\|F(z_k)\|_\infty$ (ou tout autre norme, elles sont « équivalentes »).