CentraleSupélec

Option commune Mathématiques Appliquées
Cursus Centrale & cursus Supélec
Année scolaire 2018-2019

TP « Optimisation »

Introduction

Ce texte concerne les 3 séances de Travaux Pratiques associés au cours d'optimisation. La première séance est consacrée à l'optimisation continue alors que les deux suivantes sont consacrées à l'optimisation discrète et l'optimisation multiobjectif.

Les fichiers nécessaires au TP (données, scripts) sont disponibles sur Moodle, sur Claroline et sur Edunao. Pour se connecter sur les PC fixes, l'identifiant à renseigner est soit l'identifiant personnel pour les élèves du cursus Supélec, soit l'identifiant personnel suivi de `@ECP` pour les élèves du cursus Centrale.

Langage de programmation. Vous avez le choix du langage de programmation. Nous recommandons l'usage de Matlab car les toolboxes « Optimization », « Global Optimization » et « Bioinformatics » (pour l'optimisation sur les graphes) permettent une mise en œuvre rapide des méthodes d'optimisation vues en cours.
Pour les langages de programmation autres que Matlab, il vous est demandé de vérifier que vous disposez sur votre ordinateur d'une version qui permette de faire de l'optimisation continue avec et sans contraintes, de l'optimisation approchée avec des algorithmes de type recuit simulé ou algorithmes génétiques, de l'optimisation sur les graphes et de l'optimisation discrète (en particulier, le solveur devra pouvoir considérer les problèmes de programmation linéaire mixte (variables continues et variables à valeurs entières)).

Un compte-rendu individuel ou par binôme (selon le mode de travail que vous choisirez) sera à envoyer à votre responsable de TP, copie vos responsables du cours, au plus tard une semaine après la dernière séance de TP (en précisant bien le ou les noms des élèves dans le monôme ou binôme). Le message devra être accompagné des scripts qui auront été développés. Ces scripts devront pouvoir être exécutés sans qu'il soit nécessaire de fournir des valeurs de paramètres. Ci-dessous les adresses des responsables de TP ou/et cours :

- julien.bect@centralesupelec.fr
- antonin.della-noce@centralesupelec.fr
- paul-henry.cournede@centralesupelec.fr

– laurent.lebrusquet@centralesupelec.fr

Pour les exercices qui n'auront pas été traités pendant les séances de TP, un « travail à la maison » sera nécessaire. L'évaluation du travail se fera selon les critères suivants :

– le nombre de méthodes proposées et la capacité à formuler un problème de différentes manières,
– la pertinence des méthodes proposées,
– la comparaison des méthodes entres elles,
– l'illustration des résultats par des figures pertinentes,
– l'interprétation des résultats.

1 Séance 1 : optimisation continue et optimisation approchée

1.1 Optimisation sans contraintes

Dans cette première partie, nous allons comparer diverses méthodes du gradient, et la méthode de la métrique variable (quasi-Newton) pour deux fonctions : f_1 (quadratique) et f_2 (non quadratique) définies $\forall U \in \mathbb{R}^5$, à valeurs dans $\mathbb{R}$

$$\begin{cases} f_1(U) = {}^tUSU - {}^tBU \\ \\ f_2(U) = {}^tUSU + {}^tU\exp(U) \end{cases} \tag{1}$$

où S et B sont des matrices fixées.

$\exp(U)$ est une notation pour signifier le vecteur dont les composantes sont les exponentielles des composantes de U.

Eléments de programmation. Pour Matlab : les valeurs numériques des matrices B et S sont données par la fonction Matlab `definition_constantes.m`. Les fonctions Matlab `f1.m` et `df1.m` contiennent les définitions de f_1 et de son gradient.
Pour Python, les fonctions à utiliser sont les fonctions `f1` et `df1` présentes dans le fichier `main.py` .
Pour Julia, les fonctions à utiliser sont les fonctions `f1` et `df1` présentes dans le fichier `functions.jl` .
Pour les langages autres que Matlab, Python ou Julia, il est nécessaire de retranscrire les scripts fournis dans le langage choisi.

1.1.1 Méthode du gradient.

a) Les fonctions Matlab, Python et Julia `gradient_rho_constant(f, df, U0, rho, tol)` permettent de minimiser une fonction par la méthode du gradient à pas fixe (ρ constant). Les fichiers `main.m`, `main.py` et `main.jl` illustrent l'usage de la fonction `gradient_rho_constant`.

Eléments de programmation avec Matlab. Ouvrir les fichiers `main.m` et `gradient_rho_constant.m` et observer en particulier comment la fonction est passée en argument d'entrée (passage par « handle »).

Essayer différentes valeurs de ρ (0.1, 0.01, ...) et voir que la convergence n'est pas toujours assurée.

b) S'inspirer de la fonction `gradient_rho_constant` pour coder un algorithme du gradient avec choix adaptatif du pas.

c) Tester les méthodes et comparer leurs performances : nombre d'itérations, nombre d'appels à la fonction coût, temps d'exécution.

Eléments de programmation avec Matlab. La fonction `tic/toc`, utilisée avant et après l'appel d'une fonction donne son temps d'exécution :

```
tic
GradResults=gradient_rho_constant(han_f1,han_df1,U0,0.1,1e-8);
toc
```

1.1.2 Utilisation des routines d'optimisation de la Toolbox Optimization. Méthode de Quasi-Newton (version BFGS).

Les toolboxes/librairies/packages des différents logiciels proposent différents algorithmes d'optimisation continue avec/sans contraintes. Les méthodes les plus classiques sont les méthodes de quasi-Newton. Comparer les performances des méthodes de quasi-Newton avec celles de la section 1.1.1.

Eléments de programmation avec Matlab. L'instruction `fminunc` de la toolbox Optimization permet l'accès à plusieurs méthodes d'optimisation continue sans contraintes (`doc fmincunc` pour voir les différentes possibilités).
L'appel aux différentes routines d'optimisation se fait par :

```
options =
optimoptions('fminunc','algorithm','quasi-newton','TolFun',tolerance);
[X,FVAL,EXITFLAG,OUTPUT] = fminunc(f,U0,options);
```

où `f` est le handle de la fonction à minimiser, `U0` le point initial de l'algorithme et `options` un objet défini par `optimoptions` et permettant de choisir l'algorithme à utiliser (par exemple quasi-Newton) ainsi que les paramètres de l'algorithme.
Une option intéressante de la fonction `fminunc` est la possibilité de laisser Matlab calculer un gradient approché (option par défaut) ou de lui fournir l'expression analytique du gradient :

```
options = optimoptions('fminunc','SpecifyObjectiveGradient',true);
[X,FVAL,EXITFLAG,OUTPUT] = fminunc(f_df,U0,options);
```

Dans ce cas, la fonction définie par le handle `f_df` doit fournir la valeur de la fonction à minimiser, mais aussi son gradient.

Aviez-vous un autre moyen d'obtenir les résultats trouvés pour f_1 ?

1.2 Optimisation sous contraintes

Dans cette partie, nous nous proposons de minimiser les fonctions f_1 et f_2 définies dans la section 1.1 pour $U \in \mathcal{U}_{ad} = [0;1]^5$.

1.2.1 Optimisation à l'aide de routines Matlab

Utiliser l'algorithme SQP (Sequential Quadratic Programming) pour minimiser f_1 et f_2.

Eléments de programmation avec Matlab. L'instruction `fmincon` de la toolbox `Optimization` permet l'accès à plusieurs méthodes d'optimisation continue avec contraintes.

1.2.2 Optimisation sous contraintes et pénalisation

1) Définir une fonction de pénalisation positive β, telle que $\beta(U) = 0$ si et seulement si $U \in \mathcal{U}_{ad}$.

Eléments de programmation avec Matlab. Il pourra être utile d'utiliser l'instruction `max(0,v)` calculant la partie positive du vecteur `v`.

2) Mettre en oeuvre la méthode de pénalisation en construisant une fonction : `f1penal(U)` (respectivement `f2penal(U)`) telle que $f_1^{\text{penal}}(U) = f_1(U) + \frac{1}{\epsilon}\beta(U)$, (respectivement avec f_2). Faire tendre ϵ vers 0 et obtenir le minimum du problème contraint. Vérifier que l'on trouve bien le même résultat qu'en section 1.2.1.

1.2.3 Méthodes duales pour l'optimisation sous contraintes

Il s'agit dans cette section de programmer l'algorithme d'Uzawa.

1) Ecrire $L_1(x, \lambda)$, le Lagrangien associé à f_1 et aux contraintes définissant $\mathcal{U}_{ad}$ et l'implémenter sous la forme d'une fonction.

2) Mettre en oeuvre l'algorithme d'Uzawa en résolvant à chaque itération la minimisation du problème perturbé et en mettant à jour λ par la méthode du gradient avec projection. Vérifier que l'on trouve bien le même résultat qu'en section 1.2.1.

1.3 Optimisation non convexe - Recuit simulé

Dans cette partie, nous allons étudier la minimisation d'une fonction non convexe f_3, avec $f_3(U) = f_1(U) + 10\ \sin(2 * f_1(U))$.

Le recuit simulé est une méthode d'optimisation dite heuristique. Elle peut permettre de résoudre des problèmes très variés d'optimisation non convexe (avec un grand nombre de minima locaux), problèmes discrets ou continus. Le principe de base consiste à explorer le domaine admissible en se déplaçant de point en point, selon le principe dit de Metropolis (et directement inspiré de la thermodynamique statistique de Boltzmann) :
Soit le point courant x_i et y un point dans le voisinage de x_i (notion dépendante du problème, à définir).
Si $J(y) \leq J(x_i)$, (où J est le critère à minimiser) alors $x_{i+1} = y$, mais si $J(y) > J(x_i)$), $x_{i+1} = y$ avec une probabilité $p = \exp(-(J(y)-J(x_i))/T)$ et $x_{i+1} = x_i$ avec une probabilité $(1 - p)$.
T est appelée température de descente (toujours par analogie thermodynamique) et on définit un processus de diminution de celle-ci au-cours de l'algorithme. Plus T diminue, plus la probabilité de choisir un point « moins bon » diminue. Ceci assure de finalement converger (par stabilisation) vers un minimum, au moins local. En réglant convenablement

les paramètres de descente, on doit pouvoir assurer la convergence vers le minimum global. Classiquement, le processus de descente en température choisi est le suivant : on se donne un T_0, température initiale, et on choisit $T_{i+1} = \alpha T_i$. On reste au cours de l'algorithme pendant L itérations à la même température (ce qui correspond à une descente par paliers successifs).

a) Définir la fonction f_3.

b) Utiliser une méthode d'optimisation classique (gradient ou BFGS). Tester différents points d'initialisation et vérifier que l'on ne converge pas toujours vers le même point, et en général vers des minima locaux.

c) Tester la la méthode du recuit simulé et vérifier qu'en réglant convenablement les paramètres de l'algorithme (paramètre de décroissance de la température, nombre de transformations acceptées par palier de température), on converge toujours vers le même point, quel que soit le point d'initialisation de l'algorithme.

Vérifier également que le minimum obtenu par la méthode du gradient ou de quasi-Newton n'est (en général) pas le minimum global.

Eléments de programmation avec Matlab. La fonction `simulannealbnd` de la toolbox `Global Optimization` permet l'optimisation par la méthode du recuit simulé. Cette fonction renvoie le point auquel le minimum est atteint et la valeur du minimum de la fonction.
Les principaux paramètres de réglage, accessibles et modifiables par l'instruction `saoptimset`, sont la température initiale (paramètre `'InitialTemperature'`) et le nombre de transformations acceptées par palier de température (paramètre `'ReannealInterval'`).

Remarque : Il existe bien d'autres algorithmes heuristiques pour la détermination de minima locaux : les algorithmes évolutionnaires (comme les algorithmes génétiques) longtemps considérés comme moins performants que le recuit simulé sont à nouveau très utilisés pour leur adaptation au calcul parallèle et par leur capacité à prendre en compte des critètes plus ou moins flous.

1.4 Application Synthèse d'un filtre à réponse impulsionnelle finie

On souhaite synthétiser un filtre à réponse impulsionnelle finie à 30 coefficients, tel que sa réponse fréquentielle soit proche de la réponse fréquentielle idéale définie par :

$$H_0(\nu) = 1 \text{ pour } \nu \in [0, 0.1]$$
$$H_0(\nu) = 0 \text{ pour } \nu \in [0.15, 0.5]$$

On choisit de ne considérer que les réponses impulsionnelles paires à valeurs réelles. La

réponse fréquentielle est donc à valeurs réelles et paire :

$$H(\nu) = \sum_{i=0}^{n-1} h[i] \cos(2\pi\nu i)$$

Soit $\{\nu_j\}_{1\leq j\leq p}$ une discrétisation de l'axe fréquentiel dans les 2 zones d'intérêt ($[0, 0.1]$ et $[0.15, 0.5]$).

On choisit la réponse impulsionnelle $h[i]$ tel que l'écart maximal entre les 2 réponses $H_0(\nu)$ et $H(\nu)$ soit le plus faible possible aux points de discrétisation (approche de type minimax). Le critère à minimiser est donc la fonction :

$$J(h) = \max_j |H_0(\nu_j) - H(\nu_j)|$$

1. Minimiser ce critère avec un algorithme d'optimisation sans contrainte (soit de type simplexe de Nelder et Mead, soit de type Newton). Commenter les résultats obtenus.
2. Montrer que le problème d'optimisation précédent (de type minimax) peut être reformulé comme un problème d'optimisation continue sous contraintes, et plus particulièrement comme un problème linéaire.

2 Séances 2 et 3 : optimisation discrète et optimisation multi-objectif

La deuxième séance est consacrée à la résolution de problèmes d'optimisation pour lesquels plusieurs formalisations mathématiques sont possibles et pour lesquels plusieurs algorithmes s'appliquent. Pour chaque exercice, il s'agit donc de faire le choix, à la fois du critère et de la méthode d'optimisation puis de faire une mise en œuvre.

2.1 Rangement d'objets (optimisation combinatoire)

On souhaite ranger N objets éparpillés sur le sol dans N boîtes alignées sur le sol (voir Figure 1).

Les boîtes sont alignées selon l'axe des x et sont ordonnées de gauche à droite (si $i < i'$, la boîte i est à gauche de la boîte i'). La boîte i est centrée sur un point B_i du plan (x, y). De même l'objet n°j est repéré par le point O_j du plan (x, y).

Les objets doivent être rangés de manière à :

- N'avoir qu'un seul objet par boîte,
- Ce que tous les objets soient rangés,
- Minimiser la distance totale de déplacement. Elle se calcule en sommant les distances nécessaires pour ranger chaque objet. On considère que pour ranger l'objet n°j dans la boîte n°i, la distance nécessaire est $\|O_j - B_i\|$ (distance euclidienne).

On représente une solution du problème par un vecteur à valeurs binaires : soit $\boldsymbol{x} \in \mathbb{R}^{n^2\times 1}$ tel que $x_{i,j} = 1$ si l'objet j est dans la boîte i, 0 sinon. L'objectif de l'exercice est de formuler le problème d'optimisation à résoudre comme un problème de programmation linéaire en nombres entiers, puis de le résoudre.

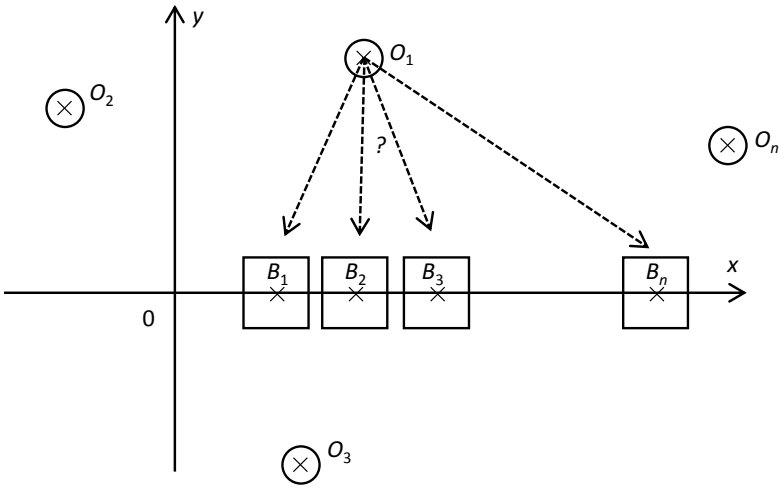

FIGURE 1 – Repérage dans le plan (x, y) des positions des boîtes et des objets à ranger.

1. Question préliminaire : Comment traduire mathématiquement que la boîte i contient un objet et un seul et que l'objet j se trouve dans une boîte et une seule ?
2. Formuler le problème d'optimisation à résoudre comme un problème de programmation linéaire en nombres entiers.
   Proposer une mise en œuvre et résoudre le problème pour l'exemple donné dans les fichiers `DonneesBoites.txt` et `DonneesObjets.txt` donnant, les coordonnées dans le plan (x, y) des points B_i et O_j.
3. Modifier le programme afin d'intégrer la contrainte suivante : l'objet n°1 doit se situer dans la boîte située juste à gauche de la boîte contenant l'objet n°2.
4. Montrer que les contraintes $x_{i,3} + x_{i+k,4} \leq 1 \forall i, \forall k > 0$ traduisent la contrainte que la boîte contenant l'objet n°3 se situe à droite de la boîte contenant l'objet n°4 (sans être forcément juste à côté). Modifier le programme pour intégrer cette contrainte supplémentaire.
5. Proposer une modélisation pour traduire la contrainte que la boîte contenant l'objet n°7 se situe à côté de la boîte contenant l'objet n°9. Modifier le programme pour intégrer cette contrainte supplémentaire.
6. On aimerait vérifier que la solution obtenue est la seule solution optimale. Proposer une stratégie pour le vérifier et conclure quant à l'unicité de la solution.

Pour chaque résolution, indiquer le vecteur des numéros de boîtes, la distance optimale et éventuellement illustrer par une figure la solution obtenue.

2.2 Communication entre espions (optimisation combinatoire)

Un service d'espionnage a n agents dans un pays ennemi. Chaque agent connait certains des autres agents avec qui il peut communiquer directement : soit $S_i \subset 1, 2, \ldots, n$ l'ensemble des agents connus de l'agent i (à noter que si $j \in S_i$, alors $i \in S_j$). Pour toute

communication directe entre l'agent i et l'agent j, le message transmis peut être intercepté avec une probabilité p_{ij}. L'agent n° 1 détient un message qu'il souhaite transmettre à tous les autres agents. On souhaite que cette transmission soit réalisée en minimisant la probabilité d'interception.

On suppose que :

- Si les agents i et j peuvent communiquer directement (et que donc p_{ij} existe), alors $p_{ji} = p_{ij}$.
- Si on considère 2 agents i et j quelconques, l'agent i, même s'il ne peut communiquer directement avec l'agent j (c'est-à-dire $i \notin S_j$), a la possibilité de faire parvenir un message à l'agent j via les autres agents.

1. Proposer une modélisation du problème puis une manière de résoudre le problème de manière exacte.

   Le fichier `ProbaInterception.txt` contient des valeurs de p_{ij} pour n=15. Les `NaN` (Not A Number) indiquent l'absence de canal communication directe entre les agents.

2. Résoudre le problème pour ces valeurs. Vous indiquerez la probabilité d'interception obtenue ainsi que la liste des communications directes entre agents, et éventuellement une représentation graphique du résultat.

Remarque : Exercice inspiré de R.K. Ahuja, T.I. Magnanti, and J.B. Orlin, Network flows, Prentice Hall, 1993.

2.3 Dimensionnement d'une poutre (optimisation multiobjectif)

On souhaite dimensionner une poutre creuse, de section carrée de côté a, de longueur unité et de densité unité, la partie creuse à l'intérieur de la poutre étant également de section carrée, de côté $b < a$ (voir figure 2). On veut minimiser à la fois son poids et sa déflexion :

$$\begin{aligned} p(a,b) &= a^2 - b^2, \\ d(a,b) &= \frac{10^{-3}}{10^{-2} + a^4 - b^4}, \end{aligned}$$

sous les contraintes géométriques :

$$\begin{cases} 0.02 \ \leq \ a \ \leq \ 1, \\ 0 \ \leq \ b \ \leq \ a - 0.01. \end{cases}$$

L'objectif est d'obtenir une estimation de la surface de Pareto (ici une courbe).

Méthode gloutonne

La méthode ici proposée est simple mais coûteuse en nombre d'évaluations des deux objectifs. Elle consiste à générer N couples (a, b) réalisables et d'approximer le front de Pareto par la courbe obtenue à partir des solutions de rang 1. Exposer la stratégie adoptée et justifier-la (à discuter en fonction du nombre N).

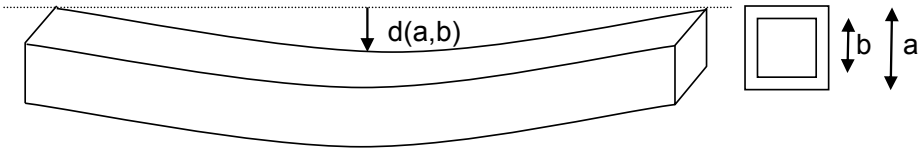

FIGURE 2 – Schéma de la poutre considérée : les paramètres a et b sont les 2 paramètres de conception ; $d(a, b)$ est la déformation obtenue (déflexion) lorsque on applique en son milieu une force de valeur fixe.

Méthode plus sophistiquée

Formuler le problème de recherche du Front de Pareto comme un problème d'optimisation monoobjectif paramétré. Le mettre en œuvre et comparer les résultats (qualité de l'estimation, nombre d'évaluations des deux objectifs) avec ceux de la méthode précédente.

Question bonus

Mettre en œuvre une métaheuristique inspirée des algorithmes génétiques.

Remarque : exemple inspiré de l'ouvrage Y. Collette et P. Siarry, Optimisation multiobjectif, Eyrolles, 2002.

2.4 Approvisionnement d'un chantier (optimisation combinatoire)

Pour cet exercice, aucune information n'est fournie quant à la méthode à mettre en œuvre car plusieurs solutions permettent de résoudre de manière exacte le problème posé.

Une entreprise de construction a un chantier à accomplir sur N semaines. L'entreprise a établi, semaine par semaine, ses besoins en engins (type tractopelle) pour ce chantier : soit d_t, le nombre d'engins nécessaires pour la semaine t ($t = 1 \dots N$). L'entreprise fait appel à une société de location qui lui fournit les tarifs du tableau 1.

acheminent sur chantier p^{init}	800
tarif location / semaine p^{loc}	200
récupération sur chantier + remise en état p^{fin}	1200

Tableau 1 – coûts par engin (en unités arbitraires) proposés par la société de location.

Proposer un algorithme permettant d'optimiser la stratégie à adopter par l'entreprise de construction si elle souhaite minimiser ses dépenses. Appliquer l'algorithme proposé aux données d_t du fichier `DonneesEnginsChantier.txt`.

2.5 Livraisons avec fenêtres de temps (optimisation combinatoire)

Cet exercice est facultatif. Il permet aux plus motivés de se confronter à un problème dont la formulation mathématique n'est pas évidente si l'on veut utiliser un algorithme d'optimisation efficace (en l'occurrence ici un algorithme de programmation dynamique).

Supposons qu'un camion ait à livrer des points $x_1, x_2, \ldots, x_n$, tous situés sur une droite. Pour chaque point x_i, on connait l'instant le plus tard d_i où la livraison peut se faire. Le camion se situe au départ en un point x_0, et veut terminer sa tournée au plus tôt. Le camion n'a pas à retourner à son point de départ. De plus, on suppose que la livraison en chaque point se fait de manière instantanée, et que le temps pour aller de x_i à x_j est proportionnel à la distance les séparant.

1. Montrer que l'on peut trouver la tournée optimale en $O(n^2)$ par un algorithme de type programmation dynamique (c'est un résultat de J. N. Tsitsiklis (voir articles dans le répertoire `Livraisons`).
2. Proposer une mise en œuvre et l'appliquer aux données des fichiers `DonneesLivraisons.txt` et `i0.txt` (numéro du point considéré comme point de départ) du répertoire `Livraisons`.

Exercice issu de F. Meunier, Introduction à la recherche opérationnelle.