République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université 8 Mai 1945 Guelma

Faculté de science et technologie

Département de génie mécanique

TRAVAUX PRATIQUES

D'OPTIMISATION

1ère ANNEE MASTER

Spécialité : Construction Mécanique

Dr. BOUCHERIT Septi

Année Universitaire : 2023/2024

Sommaire

Introduction

Ce polycopié présente une série de travaux pratiques visant à familiariser les étudiants avec les concepts et les outils de l'optimisation numérique à l'aide de MATLAB®. Ces TP couvrent un large éventail de sujets, allant de la présentation des fonctions de référence d'optimisation à la résolution de problèmes d'optimisation non linéaire avec contraintes.

Le premier TP introduit les fonctions de référence couramment utilisées dans le domaine de l'optimisation, permettant aux étudiants de comprendre leur utilisation et leurs caractéristiques. la définition et le traçage des courbes de quelques fonctions test utilisées en optimisation, ce qui permet aux étudiants de mieux comprendre le comportement des algorithmes d'optimisation.

Le deuxième TP se concentre l'optimisation linéaire avec contraintes ce qui permets aux étudiants de bien maitraiser les démarches à suivre pour résoudre un problème d'optimisation linéaire.

Les autres TP couvrent différents types de problèmes d'optimisation, tels que les problèmes d'optimisation non linéaire avec et sans contraintes, ainsi que la minimisation non linéaire avec divers types de contraintes.

Enfin, le dernier TP explore l'utilisation de l'outil optimtool pour résoudre des problèmes d'optimisation non linéaire avec contraintes, offrant aux étudiants une expérience pratique de la résolution de problèmes réels.

Dans l'ensemble, ces TP offrent aux étudiants une occasion précieuse d'acquérir des compétences pratiques en optimisation numérique et de comprendre l'application des concepts théoriques à des problèmes concrets.

TP N1

Objectifs :

1- Présentation des fonctions références d'optimisation en Matlab

2-Définition et traçage des courbes de quelques fonctions test en optimisation

Fonctions de références d'optimisation en Matlab

Les outils disponible dans l'Optimization Toolbox de Matlab fournissent une gamme variée de fonctionnalités pour la maximisation ou la minimisation de fonctions, qu'elles soient soumises à des contraintes ou non. Ces fonctionnalités incluent des solveurs pour différents types de programmation, tels que :

1- la programmation linéaire (PL) .

2- la programmation linéaire en nombres entiers mixtes (PLNIM).

3- la programmation quadratique (PQ).

4- la programmation conique du second ordre (PCSO).

5- la programmation non linéaire (PNL).

Il est possible de formuler un problème d'optimisation en utilisant des fonctions et des matrices, ou bien en spécifiant directement des expressions mathématiques de manière symbolique. En outre, l'utilisation de la différenciation automatique pour les fonctions objectif et les contraintes permet d'obtenir des solutions plus précises et rapides.

Selon la nature du problème d'optimisation à traiter, il est nécessaire de choisir la bonne fonction MATLAB apte à résoudre un tel problème. Dans le tableau suivant, sont regroupées les principales fonctions selon la nature des problèmes qu'elles peuvent résoudre.

Problème monodimensionnel sans contrainte	Méthode de la recherche dorée	**fminbnd**
Problème multidimensionnel sans contrainte	Méthode de Nedler et Mead	**fminsearch**
Problème multidimensionnel sans contrainte	Méthode de quasi-Newton	**fminunc**
Problème multidimensionnel sans contrainte - identification paramétrique	Méthode de Gauss-Newton	**lsqnonlin**
Identification paramétrique - cas d'un polynôme	Méthode des moindres carrés	**polyfit** / **polyval**
Problème linéaire multidimensionnel avec contraintes	Méthode du Simplex (cf. cours de Recherche Opérationnelle)	**linprog**

Problème non-linéaire multidimensionnel avec contraintes	Méthode Sequential Quadratic Programming	**fmincon**

1- `fminbnd` : c'est une fonction partiellement basée sur la méthode de la recherche dorée (optimisation monodimensionnelle sans contrainte). La syntaxe générale de cette fonction peut faire appel à une fonction externe (premier cas) ou à une fonction anonyme (second cas):

1er cas

```
[x,fval,exitflag,output] = fminbnd('fun',x1,x2,options)
```

2ème cas

```
[x,fval,exitflag,output] = fminbnd(@(x) fun(x),x1,x2,options)
```

Arguments d'entrée :

- `fun` : dans le premier cas, fun est une fonction externe où est exprimée la fonction objectif qu'on cherche à minimiser. Dans le second cas, fun est une fonction anonyme où est exprimée la fonction objectif qu'on cherche à minimiser. Entête de la fonction `fun` : `function f = fun(x)`
- `x1` et `x2` : bornes inférieure et supérieure du domaine dans lequel la solution est recherchée.
- `options` (facultatif) ; voir ci-dessous.

Arguments de sortie :

- `x` : valeur finale des variables, ce sont les solutions si la méthode a convergé ;
- `fval` : valeur finale de la fonction à minimiser ;
- `exitflag` (Facultatif) : indicateur de fin d'exécution : 1 arrêt car solution trouvée, 0 car nombre d'itérations maximal atteint ;
- `output` (Facultatif) : nombres d'itérations et d'évaluations de fonction réalisés.

2- `fminsearch` **(matlab)**

La fonction `fminsearch` code la méthode de Nedler et Mead et permet de résoudre un problème multidimensionnel sans contrainte. C'est un méthode d'ordre 0, qui n'utilise aucune information concernant les gradients de la fonction objectif. La syntaxe générale de cette fonction peut faire appel à une fonction externe (premier cas) ou à une fonction anonyme (second cas):

1er cas

```
[x,fval,exitflag,output] = fminsearch('fun',x0,options)
```

2ème cas

```
[x,fval,exitflag,output] = fminsearch(@(x) fun(x),x0,options)
```

Arguments d'entrée :

- `fun` : dans le premier cas, `fun` est une fonction externe où est exprimée la fonction objectif qu'on cherche à minimiser. Dans le second cas, `fun` est une fonction anonyme où est exprimée la fonction objectif qu'on cherche à minimiser.

  Entête de la fonction externe `fun` : `function f = fun(x) ;`

- `x0` : vecteur des valeurs initiales pour démarrer la méthode. La dimension de x0 est égale au nombre de variables du problème ;

- options (facultatif) ; voir ci-dessous.

Arguments de sortie : voir les arguments de sortie de fminbnd.

3- fminunc **(optimization toolbox)**

La fonction fminunc code une méthode de type quasi-Newton pour l'optimisation multidimensionnelle sans contrainte. C'est une méthode d'ordre 2, la matrice hessienne est évaluée. La syntaxe générale de cette fonction peut faire appel à une fonction externe (premier cas) ou à une fonction anonyme (second cas):

1er cas

```
[x,fval,exitflag,output] = fminunc('fun',x0,options)
```

2ème cas

```
[x,fval,exitflag,output] = fminunc(@(x) fun(x),x0,options)
```

Arguments d'entrée et de sortie : voir fminsearch

4- `lsqnonlin` **(optimization toolbox)**

La fonction `lsqnonlin` code la méthode de Gauss-Newton pour l'optimisation multidimensionnelle sans contrainte. lsqnonlin est dédiée aux problèmes d'identification paramétrique puisque la fonction objectif doit s'exprimer sous la forme de termes élevés au carré ; chaque terme représentant la différence entre une valeur donnée et une valeur calculée par un modèle. La syntaxe générale de cette fonction peut faire appel à une fonction externe (premier cas) ou à une fonction anonyme (second cas):

1er cas

```
[x,resnorm,residual,exitflag,output] = lsqnonlin('fun',x0,lb,ub,options)
```

2ème cas

```
 [x,resnorm,residual,exitflag,output] = lsqnonlin(@(x)
fun(x),x0,lb,ub,options)
```

Arguments d'entrée :

- `fun` : dans le premier cas, `fun` est une fonction externe où sont exprimés les termes intervenant dans la fonction objectif qu'on cherche à minimiser. Dans le second cas, fun est une fonction anonyme où sont exprimés les termes intervenant dans la fonction objectif qu'on cherche à minimiser. L'entête de la fonction `fun` est ainsi : `function F = fun(x)` ; F est un vecteur contenant l'ensemble des termes formant la fonction objectif (égale à la somme des termes de F élevés au carré) ;

- `x0` : vecteur des valeurs initiales pour démarrer la méthode. La dimension de x0 est égale au nombre de variables du problème ;

- `lb` et `ub` : bornes inférieure et supérieure des éléments de x : domaine dans lequel la solution est recherchée. lb et ub ont la même dimension que x ;

- options (pas obligatoire) ; voir ci-dessous.

Arguments de sortie :

- `x` : valeur finale des variables, ce sont les solutions si la méthode a convergé ;

- `resnorm` : valeur finale de la fonction à minimiser ;

- `residual` : vecteur des résidus, c'est à dire le vecteur sortie de fun pour la valeur finale de x

- `exitflag` (pas obligatoire) : indicateur de fin d'exécution : 1 arrêt car solution trouvée, 0 car nombre d'itérations maximal atteint ;

- `output` (pas obligatoire) : nombres d'itérations et d'évaluations de fonction réalisés.

5- `fmincon` (optimization toolbox)

La fonction `fmincon` code des plusieurs méthodes (dont SQP) de résolution de problème d'optimisation multidimensionnelle avec contraintes. L'utilisateur avisé peut choisir la méthode numérique ou sinon laisser la fonction `fmincon` choisir automatiquement la méthode la plus adaptée. La syntaxe générale de cette fonction peut faire appel à une fonction externe (premier cas) ou à une fonction anonyme (second cas) :

1er cas

```
 [x,fval,exitflag,output,lambda,grad,hessian] =
fmincon('fun',x0,A,B,Aeq,Beq,lb,ub,'nonlcon',option)
```

2ème cas

```
 [x,fval,exitflag,output,lambda,grad,hessian] = fmincon(@(x)
fun(x),x0,A,B,Aeq,Beq,lb,ub,'nonlcon',option)
```

Arguments d'entrée :

- `fun` : dans le premier cas, fun est une fonction externe où est exprimée la fonction objectif qu'on cherche à minimiser. Dans le second cas, fun est une fonction anonyme où est exprimée la fonction objectif qu'on cherche à minimiser. Entête de la fonction `fun` : `function f = fun(x) ;`

- `x0` : vecteur des valeurs initiales pour démarrer la méthode. La dimension de x0 est égale au nombre de variables du problème ;

- `A` et `B` : respectivement une matrice et un vecteur qui permettent d'exprimer les contraintes linéaires de type inégalité sous la forme : `A*x <= B`.

  A et B ont autant de lignes que de contraintes inégalité linéaires ; A a autant de colonnes que le nombre de variables de contrôle ;

- `Aeq` et `Beq` : respectivement une matrice et un vecteur qui permettent d'exprimer les contraintes linéaires de type égalité sous la forme : `Aeq*x = Beq`.

  `Aeq` et `Beq` ont autant de lignes que de contraintes égalité linéaires ; `Aeq` a autant de colonnes que le nombre de variables de contrôle ;

- `lb` et `ub` : bornes inférieure et supérieure des éléments de x. lb et ub ont la même dimension que x ;

- `'nonlcon'` : nom de la fonction externe MATLAB dans laquelle sont définies les éventuelles contraintes non-linéaires égalité et inégalité. L'entête de nonlcon.m est : `function [c,ceq] = nonlcon(x)`

  C : est un vecteur qui contient les résidus des contraintes inégalité (toutes les composantes de c doivent être négatives à la solution).

  `ceq` est un vecteur qui contient les résidus des contraintes égalité (toutes les composantes de ceq doivent être nulles à la solution).

  `nonlcon` peut être également définie comme une fonction anonyme ;

- `options` (pas obligatoire) ; voir ci-dessous.

Arguments de sortie :

- `x` : valeur finale des variables, ce sont les solutions si la méthode a convergé ;

- `fval` : valeur finale de la fonction à minimiser ;

- `exitflag` (pas obligatoire) : indicateur de fin d'exécution : 1 arrêt car solution trouvée, 0 car nombre d'itérations maximal atteint ; il existe d'autres valeurs possible.

- `output` (pas obligatoire) : nombres d'itérations et d'évaluations de fonction réalisés.
- `lambda` : structure qui contient la valeur des multiplicateurs de Lagrange à la valeur finale de x
- `grad` : vecteur gradient à la valeur finale de x
- `hessian` : matrice hessienne à la valeur finale de x

Options

Les options par défaut peuvent être modifiées par l'intermédiaire de la fonction `optimset` qui est utilisée ainsi (par exemple avec `fminbnd`) :

```
options = optimset('propriete1',valeur1, 'propriete2',valeur2, etc ...)
 [x,fval,exitflag,output] = fminbnd('fun',x1,x2,options)
```

Les principales options modifiables sont entre autres :

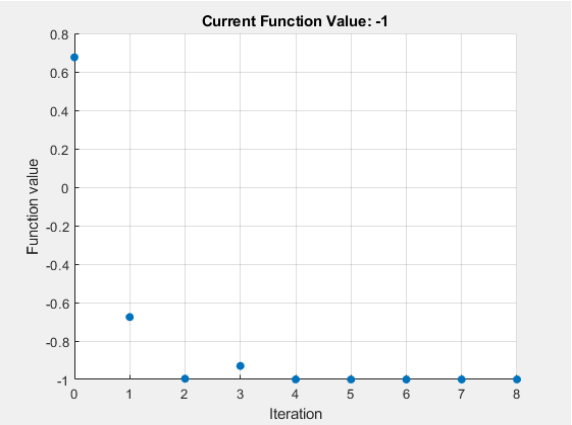
- `'display'` : on choisira la valeur `'iter'` pour la propriété `'display'` afin d'avoir des informations sur le processus de convergence affichées à l'écran à chaque itération. Par défaut, les affichages ne sont proposés qu'en fin d'exécution.
- `'MaxIter'` : si le nombre d'itérations n'est pas suffisant, il peut être modifié en choisissant une valeur entière plus grande.
- `'MaxFunEvals'` : si le nombre d'évaluations de la fonction objectif n'est pas suffisant, il peut être modifié en choisissant une valeur entière plus grande.
- `'tolX'` : si on veut changer le critère d'arrêt basé les variations des variables de contrôle, il peut être modifié en choisissant une valeur réelle (plus cette valeur est petite, plus les tolérances sont serrées).
- `'Diagnostics'` : on choisira la valeur `'on'` pour avoir un diagnostic du problème traité avant le début des calculs itératifs.

Exemple

1- Trouver le point où la fonction $sin(x)$ atteint son minimum dans l'intervalle $0 < x < 2\pi$.

Solution

```
fun = @sin;
x1 = 0;
x2 = 2*pi;
options = optimset('PlotFcns',@optimplotfval);
[x,fval,exitflag,output] = fminbnd(fun,x1,x2,options)
```




```
x = 4.7124
fval = -1.0000
exitflag = 1
output = struct with fields:
    iterations: 8
     funcCount: 9
     algorithm: 'golden section search, parabolic interpolation'
       message: 'Optimization terminated:...'
```

Execrces

1- Minimiser la fonction de Rosenbrock. :

$$f(x) = 100(x_2^2 - x_1)^2 + (1 - x_1)^2.$$

La fonction est minimisée au point $x = [1,1]$ avec une valeur minimale de 0.

Définissez le point de départ à $x_0 = [-1.2,1]$ et minimisez la fonction de Rosenbrock en utilisant fminsearch.

Solution

```
fun = @(x)100*(x(2) - x(1)^2)^2 + (1 - x(1))^2;
x0 = [-1.2,1];
options = optimset('PlotFcns',@optimplotfval);
x = fminsearch(fun,x0,options)
```

2- Minimiser la fonction $f(x) = 3{x_1}^2 + 2x_1x_2 + {x_2}^2 - 4x_1 + 5x_2$.

Solution

```
fun = @(x)3*x(1)^2 + 2*x(1)*x(2) + x(2)^2 - 4*x(1) + 5*x(2);
x0 = [1,1];
[x,fval] = fminunc(fun,x0)
```

3- Configurez le problème de minimisation de la fonction de Rosenbrock sur le disque unité, $||x||^2 = 1$

Solution

Tout d'abord, créez une fonction qui représente la contrainte non linéaire. Enregistrez cela sous le nom de fichier unitdisk.m sur votre chemin MATLAB

```
function [c,ceq] = unitdisk(x)
c = x(1)^2 + x(2)^2 - 1;
ceq = [];
```

Créez les spécifications restantes du problème.

```
fun = @(x)100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
nonlcon = @unitdisk;
A = [];
b = [];
Aeq = [];
beq = [];
lb = [];
ub = [];
x0 = [0,0];
[x,fval,exitflag,output] = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon)
```

TP N2

Programmation Linéaire

Programmation Linéaire :

Démarches à suivre pour résoudre un problème d'optimisation linéaire

1- Se faire une idée générale du problème

2- Identifier l'objectif (maximiser ou minimiser quelque chose)

3- Identifier (nommer) les variables

4- Identifier les contraintes

5- Déterminer les variables que vous pouvez contrôler

6- Spécifiez toutes les quantités en notation mathématique

7- Vérifiez que le modèle est complet et correct.

Le problème d'optimisation consiste à minimiser la fonction objectif, en tenant compte de toutes les autres expressions comme contraintes.

$$\min f^T x \quad de\ telle\ sorte\ que \begin{cases} A.x \leq b \\ Aeq.x = beq \\ lb \leq x \leq ub \end{cases}$$

$f^T x$ Désigne un vecteur ligne de constantes f multipliant un vecteur colonne de variables x. En d'autres termes.

$$f^T x = f(1)x(1) + f(2)x(2) + \ldots + f(n)x(n),$$

Où n est la longueur de f.

Les problèmes de cette partie sont à traité à l'aide du logiciel *Matlab*. La fonction à utiliser est la fonction *linprog*. La commande *help linprog* donne accès à l'aide sur cette fonction. Il est important de remarquer que *linprog* résout un problème de minimisation : Donc, le maximum d'une fonction Z sera cherché comme le minimum de la fonction –Z.

Syntaxe:

x = linprog(f,A,b) résout min `f'*x` sous contraintes `A*x ≤ b`.

x = linprog(f,A,b,Aeq,beq) résout le problème ci-dessus en satisfaisant également les contraintes d'égalité `Aeq*x = beq`. Mettre `A = []` et `b = []` s'il n'y a pas d'inégalités.

x = linprog(f,A,b,Aeq,beq,lb,ub) définit des bornes inférieures et supérieures sur les variables x, de sorte que la solution soit toujours dans la plage lb ≤ x ≤ ub. Mettre `Aeq = []` et `beq = []` s'il n'y a pas d'égalités.

La commande *linprog* met en oeuvre deux types d'algorithmes :

1- *Medium-Scale Algorithms* : Algorithme qui implémente la méthode de simplexe.

2- *Large-Scale Algorithms* : Il utilise la méthode de 'point intérieurs'. On désélectionne l'option implicite à l'aide de la commande :

```
Options=optimset ('LargeScale','off') ;
```

Pour les exercices de ce TP, seul le premier Algorithme est utilisé

Exercice 1

Un confiseur dispose de 4800kg de miel, 10800 kg de sucre et 33000 kg d'amandes. Il se propose de fabriquer du nougat mou et du nougat dur avec deux machines :
Avec 300g de miel, 900g de sucre et 300g d'amandes, la machine N1 produit 25 barres de nougat mou et 80 barres de nougat dur.
Avec 600g de miel, 1200g de sucre et 300 g d'amandes, la machine N2 produit 160 barres de nougat mou et 32 barres de nougat dur.
La vente d'une barre de nougat mou rapporte 0.40 euro, celle de nougat dur 0.50 euro.
Comment répartir la production entre les deux machines pour maximiser le bénéfice ? Quel est alors le bénéfice maximum ?

Formuler ce problème comme un problème de programmation linéaire. Le résoudre sous Matlab.

Voici les commandes utilisées pour résoudre ce problème :

```
Z=[-50 -80];              % le maximum cherché est le minimum de z
A=[1 1;3 4;1 2];          % coefficients de la matrice des contraintes
B=[11000 3600 1600] ;     % second membres des contraintes
LB=[11000 3600 1600];
Options=optimset ('LargeScale','off') ;
[x,zval,EXITFLAG]=linprog(Z,A,B,[],[],[0 0],[],[],Options)
```

Exercice 2

Soit la fonction économique: $z = x_1 + 3x_2 + 5x_3 + x_4 + 4x_5$ les variables sont positives et sont soumises aux contraintes suivantes :

$$\begin{cases} x_1 + x_2 - x_3 + x_4 = 1 \\ 2x_1 + 4x_3 + 2x_4 + x_5 = 7 \\ x_1 + 6x_2 + x_3 + 2x_5 = 19 \end{cases}$$

1- Les variables sont réelles, trouver le maximum de z.
2- Les variables sont réelles, trouver le minimum de z.

Exercice 3

Dans ce problème les variables sont réelles et positives

1- Trouver le minimum de la fonction : $z = 50x_1 + 40x_2 + 10x_3 + 6x_4$

Les contraintes sont : $\begin{cases} -5x_1 + 3x_2 + 3x_3 + 2x_4 \leq 50 \\ 16x_1 + 12x_2 \geq 152 \\ -3x_3 + 2x_4 \geq 6 \end{cases}$

$$0 \leq x_1 \leq 5 \qquad 0 \leq x_2 \leq 8 \qquad 0 \leq x_3 \leq 2 \qquad 0 \leq x_4 \leq 2$$

2- Trouver le maximum de la fonction : $z = 16x_1 + 12x_2$

Les contraintes sont : $\begin{cases} 5x_1 + 3x_2 + 3x_3 + 2x_4 \leq 50 \\ -50x_1 + 40x_2 + 10x_3 + 6x_4 \leq 150 \\ 16x_3 + 12x_2 - 30x_3 - 20x_4 \leq 0 \end{cases}$

$$0 \leq x_1 \leq 5 \qquad 0 \leq x_2 \leq 8 \qquad 0 \leq x_3 \leq 2 \qquad 0 \leq x_4 \leq 2$$

Essayer plusieurs points initiaux : (0,0,0,0) , (1,1,1,1),((2,2,2,2) par exemple. Pouvez vous expliquer les résultats ?

Exercice 4

Une usine fabrique quatre produits A,B,C,D au moyen de deux machines M1 et M2. Les temps machines requis par unité de volume de chacun de ces produits sont données ci-dessous ainsi que les bénéfices unitaires correspondants

	A	B	C	D
M1	7mn	10mn	4mn	9mn
M2	3mn	40mn	1mn	1mn
bénéfice	45	100	30	50

La disponibilité journalière de M1 est de 1200 mn et celle de M2 est de 800mn. On suppose qu'aucun problème d'ordonnancement ne vient compliquer les choses (utilisation simultanée des machines, etc…)

1- Quelle quantité de chaque produit faut-il fabriquer chaque jour de façon à rendre le bénéfice maximum.
2- Que devient la solution si les bénéfices unitaires associés à A et D passent de 45 à 40 et de 50 à 60 respectivement
3- Même question, le bénéfice unitaire associé à C baissant de 30 à 25, les autres restant les mêmes qu'au 1.

4- Que devient la solution si les disponibilités journalières des machines M1 et M2 sont portées de 1200mn à 1500mn et de 800mn à 100mn respectivement.

TP N° 03 :

Paramètres d'un algorithme d'optimisation

1.1 Approximation Initiale

Pour initialiser l'algorithme, il est nécessaire d'avoir une approximation initiale a la solution x0.(Point de départ). Le choix d'une bonne approximation initiale conditionne la convergence ou pas à la solution.

1.2. Nombre d'Itérations

Un algorithme d'optimisation utilise un processus récursif, calcule une nouvelle approximation (itération) à la solution réelle jusqu'à ce que les critères de convergence soient atteints. En programmation, c'est une boucle de répétition où la nouvelle approximation est construite à partir les approximations antérieures.

1.3. Vitesse de convergence

Quand on parle de convergence proche d'une solution, on parle de la vitesse à laquelle les termes de l'itération approchent sa limite.

$$lim_{n\to\infty}\frac{|x_{n+1}-\xi|}{|x_n-\xi|^q}=\mu, \qquad ou\ \mu>0\ et\ q\ est\ l'ordre de\ convergence$$

En général, les ordres de convergences son linéaires ($p=1$), quadratiques ($p=2$), cubiques ($p=3$), quartiques ($p=4$)… Une méthode d'optimisation avec un ordre de convergence supérieur arrive à la solution avec peu d'itérations. Le choix d'une méthode avec une haute convergence est important pour les problèmes d'une certaine taille ou avec de multiples paramètres. Par exemple, pour une convergence quadratique, on peut dire que le nombre de chiffres corrects est double (au minimum) à chaque pas de calcul. Ou dit sous une autre forme, l'erreur diminue quadratiquement à chaque itération.

Si un algorithme ne converge pas, ça ne veut pas dire qu'il n'existe pas de solution. Il n'existe aucun algorithme universel dont la convergence soit garantie, en général il dépend du choix de l'initialisation x0 et des propriétés de la fonction (continuité, dérivabilité)

1.4. Critère d'arrêt

Critères pour arrêter le processus de calcul. Il existe plusieurs critères d'arrêt. Les plus utilisées :

a) Nombre maximal d'itérations Nmax

b) $\|f(x_n)\| < \varepsilon_1$ Valeur de la fonction

c) $\|x_{n+1} - x_n\| < \varepsilon_2$ Différence entre deux approximations successives

Où $\varepsilon_1, \varepsilon_2 , \in \Re$ sont les tolérances et sont choisies en fonction du type de problème. En général, ce sont des valeurs négligeables ($\varepsilon_i \approx 10^{-4} - 10^{-6}$).

2. Rappel

2.1. Points critiques : maximums, minimums

Dans un ensemble ordonné, le plus grand élément (ou le plus petit) d'une partie de cet ensemble est un extremum maximum (ou minimum) s'il est supérieur (ou inferieur) à tous les autres éléments de la partie. Ce groupe d'éléments sont connus sous le nom de points critiques ou points extremum définis sur un domaine d'étude D (espace topologique).

$f(a)$est un **maximum global** si $\forall x \in D \quad f(x) \leq f(a)$

$f(a)$est un **minimum global** si $\forall x \in D \quad f(x) \geq f(a)$

$f(a)$est un **maximum local** (ou relatif) s'il existe un voisinage V de a tel que

$$\forall x \in V , \quad f(x) \leq f(a)$$

$f(a)$est un **minimum local** (ou relatif) s'il existe un voisinage V de a tel que

$$\forall x \in V , \ f(x) \geq f(a)$$

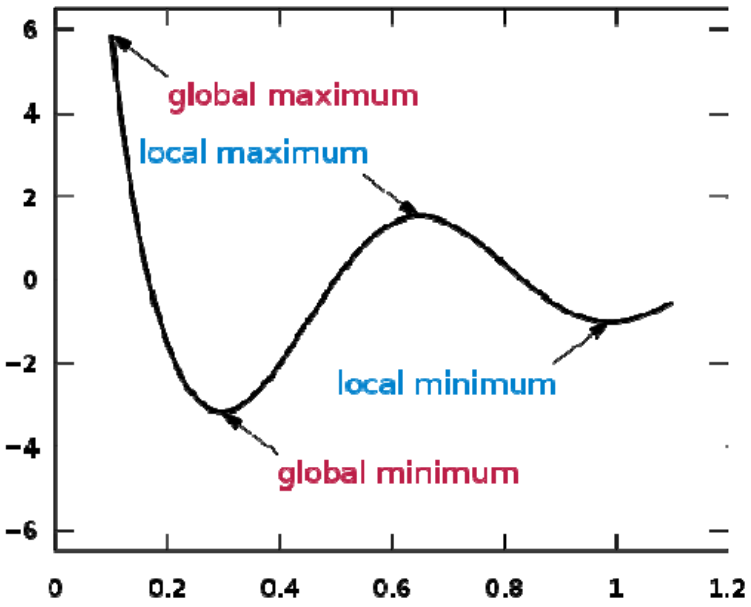


Figure 1- Points extrêmes (maximums et minimums locaux et globaux) sur une fonction
Pour trouver les maximums et les minimums d'une fonction, on utilise le calcul différentiel, et là ou la dérivée de la fonction s'annule, on trouve soit un maximum ou un minimum. Un minimum local est facile à trouver, mais il est difficile de trouver un minimum absolu.

En général, une fonction a plusieurs minimums. Pour arriver à la solution désirée (minimum global), il est très important d'analyser la fonction en détail avant de choisir un point dedépart x_0 pour l'algorithme.

Pour trouver le type de point critique, on utilise les deuxièmes dérivées évaluées dans le point d'étude.

Pour le cas d'une fonction à une variable $f(x)$

- si $f''(a) > 0$, on trouve un minimum local en ce point,
- si $f''(a) < 0$ on trouve un maximum local
- si $f''(a) > 0$, quand $f''(a) = 0$ on n'a pas d'information, mais ça peut être un point de selle.

Par exemple, pour les fonctions continues et dérivables deux fois, les points stationnaires identifiés (là ou la dérivée est 0) sont classés selon la matrice Hessienne (minimum local si positif, maximum local si négatif et indéfini s'il s'agit d'un point selle). Pour le cas d'une fonction à deux variables, on trouve $f_{xx}(x,y)$, $f_{yy}(x,y)$ et $f_{xy}(x,y)$ evalué au point (a,b). Le déterminant de la matrice Hessienne

$$H = f_{xx}\,(x,y) * f_{yy}\,(x,y) - {f^2}_{xy}\,(x,y)$$

$f_{xx}\,(a,b) * f_{yy}\,(a,b) - {f^2}_{xy}\,(a,b)$	$f_{xx}\,(a,b)$	Classification
> 0	> 0	Minimum local
> 0	< 0	Maximum local
< 0	-	Point selle

Manipulation

Pour trouver les points extrêmes (ou points critiques) d'une fonction de deux variables, par exemple : $f\,(x,y) = x^3 + y^3 + 3x^2 - 3y^2 - 8$, on doit trouver les points qui annulent les dérivés partiels de la fonction $\partial_x f\,(x,y) = 0\; et\; \partial_y f\,(x,y) = 0$.

```
%-------------------------------------------------------
syms x y ;

f=x^3+y^3+3*x^2-3*y^2-8;
fx=diff(f,x)
fy=diff(f,y)
S=solve(fx,fy)
%-------------------------------------------------------
```

La matrice Hessienne H(f) d'une fonction f est une matrice carrée de ses dérivées partielles secondes.

$$H_{ij}(f) = \frac{\partial^2 f}{\partial x_i \partial x_j} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots\cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots\cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \cdots\cdots & \cdots\cdots & \ddots & \cdots\cdots \\ \frac{\partial^2 f}{\partial x_n \partial y_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots\cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

La commande solve trouve les solutions qui sont égales à zéro simultanément pour les deux fonctions dérivées. S est une structure variable. Pour voir les valeurs de S :

```
[S.x,S.y]
```

Le résultat montre les points critiques pour la fonction analysée {(0,0),(0,2),(-2,0).(-2,2)}.

Pour visualiser les résultats on peut utiliser la fonction :

```
[x,y]= meshgrid (-3:0.1:3);
```

```
z= x.^3+y.^3+3*x.^2-3*y.^2-8;

mesh(x,y,z)

xlabel('x')

ylabel('y')

zlabel ('z=f(x,y)')
```

Ou aussi la fonction

```
surfc(x,y,z)
```

Parfois c'est aussi utile de visualiser les lignes de niveaux dans un graphique séparé

```
contour(x,y,z)
```

Ou dans le même graphique, on utilise pour dessiner les contours en dessous de la maille ou pour dessiner les courbes de niveau en dessus de la surface.

```
meshc(x,y,z)

surfc(x,y,z)
```

On peut changer le paramètre par défaut des courbes de niveau :

```
contour(x,y,z,20)
```

On observe mieux les deux points critiques (-2,0 et 2,0)

Matlab permet de dessiner les courbes a différentes hauteurs avec :

```
[c,h] = contour(x,y,z,-14 :-4) ;

clabel (c,h)
```

A partir de ces contours, on observe :

- Peu importe la direction d'approche du point (-2,0) les courbes de niveau augmentent. Par conséquent, on trouve un maximum local à (-2,0).
- Peu importe la direction d'approche du point (0,2) les courbes de niveau diminuent.

Par conséquent, on trouve un minimum local au (0,2).

- Pour les points (0,0) et (-2,2), on observe une croissance et décroissance des courbes de niveau selon des directions opposées. On peut dire qu'y a un point de selle à (0,0) et à (-2,2).

Travail demandé

Soit les deux fonctions

$$f\ (x_1, x_2) = \ {x_1}^2 - 3x_1x_2 + 4{x_2}^2 + x_1 - \ x_2$$

$$f(x, y) = x^2 + y^2 + xy - 3x - 6y$$

Déterminer les points critiques des deux fonctions

TP N4

Minimisation non linéaire sans contraintes avec gradient et Hessien

Ce TP montre comment minimiser la fonction de Rosenbrock sans dérivées et avec gradient et Hessien

La fonction de Rosenbrock

$$f(x) = 100(x(2) - x(1)^2)^2 + (1 - x(1))^2.$$

La fonction f(x) a un minimum unique au point x = [1,1] où f(x) = 0. Cet exemple montre plusieurs façons de minimiser f(x) en commençant au point $x_0 = [-1.9,2]$.

1- Optimisation sans dérivées

La fonction fminsearch trouve un minimum pour un problème sans contraintes. Elle utilise un algorithme qui n'estime aucune dérivée de la fonction objective. Au lieu de cela, elle utilise une méthode de recherche géométrique décrite dans l'algorithme fminsearch. Minimiser la fonction banane en utilisant fminsearch. Inclure une fonction de sortie pour rapporter la séquence des itérations.

```
fun = @(x)(100*(x(2) - x(1)^2)^2 + (1 - x(1))^2);
options = optimset('OutputFcn',@bananaout,'Display','off');
x0 = [-1.9,2];
[x,fval,eflag,output] = fminsearch(fun,x0,options);
title 'Rosenbrock solution via fminsearch'

Fcount = output.funcCount;
disp(['Number of function evaluations for fminsearch was ',num2str(Fcount)])

disp(['Number of solver iterations for fminsearch was ',num2str(output.iterations)])
```

2- Optimisation avec des dérivées estimées

La fonction fminunc trouve un minimum pour un problème sans contraintes. Elle utilise un algorithme basé sur les dérivées. L'algorithme tente d'estimer non seulement la première dérivée de la fonction objective, mais aussi la matrice des dérivées secondes (HESSIEN. fminunc est généralement plus efficace que fminsearch.

Minimiser la fonction Rosenbrock en utilisant fminunc.

```
options = optimoptions('fminunc','Display','off',...
    'OutputFcn',@bananaout,'Algorithm','quasi-newton');
[x,fval,eflag,output] = fminunc(fun,x0,options);
title 'Rosenbrock solution via fminunc'

Fcount = output.funcCount;

disp(['Number of function evaluations for fminunc was ',num2str(Fcount)])

disp(['Number of solver iterations for fminunc was ',num2str(output.iterations)])
```

3- Optimisation avec la méthode de descente

Si vous tentez de minimiser la fonction de Rosenbrock en utilisant un algorithme de descente, la courbure élevée du problème rend le processus de résolution très lent. Vous pouvez exécuter fminunc avec l'algorithme de descente la plus raide en définissant l'option HessUpdate cachée sur

la valeur 'steepdesc' pour l'algorithme 'quasi-newton'. Définissez un nombre maximal d'évaluations de fonction plus grand que par défaut, car le solveur ne trouve pas rapidement la solution. Dans ce cas, le solveur ne trouve pas la solution même après 600 évaluations de fonction.

```
options = optimoptions(options,'HessUpdate','steepdesc',...
    'MaxFunctionEvaluations',600);
[x,fval,eflag,output] = fminunc(fun,x0,options);
title 'Rosenbrock solution via steepest descent'

Fcount = output.funcCount;

disp(['Number of function evaluations for steepest descent was ',...
    num2str(Fcount)])

disp(['Number of solver iterations for steepest descent was ',...
    num2str(output.iterations)])
```

4- Optimisation avec un gradient analytique

Si vous fournissez un gradient, fminunc résout l'optimisation en utilisant moins d'évaluations de fonction. Lorsque vous fournissez un gradient, vous pouvez utiliser l'algorithme 'trust-region', qui est souvent plus rapide et utilise moins de mémoire que l'algorithme 'quasi-newton'. Réinitialisez les options HessUpdate et MaxFunctionEvaluations à leurs valeurs par défaut.

```
grad = @(x)[-400*(x(2) - x(1)^2)*x(1) - 2*(1 - x(1));
            200*(x(2) - x(1)^2)];
fungrad = @(x)deal(fun(x),grad(x));
options = resetoptions(options,{'HessUpdate','MaxFunctionEvaluations'});
options = optimoptions(options,'SpecifyObjectiveGradient',true,...
    'Algorithm','trust-region');
[x,fval,eflag,output] = fminunc(fungrad,x0,options);
Fcount = output.funcCount;
disp(['Number of function evaluations for fminunc with gradient was ',...
    num2str(Fcount)])

disp(['Number of solver iterations for fminunc with gradient was ',...
    num2str(output.iterations)])
```

5- Optimisation avec un Hessien analytique

Si vous fournissez un Hessien (matrice des dérivées secondes), fminunc peut résoudre l'optimisation en utilisant encore moins d'évaluations de fonction. Pour ce problème, les résultats sont les mêmes avec ou sans le Hessien.

```
Hess = @(x)[1200*x(1)^2 – 400*x(2) + 2, -400*x(1) ;
            -400*x(1), 200];
fungradhess = @(x)deal(fun(x),grad(x),hess(x));
options.HessianFcn = ‘objective’;
[x,fval,eflag,output] = fminunc(fungradhess,x0,options);
Fcount = output.funcCount;
disp([‘Number of function evaluations for fminunc with gradient and Hessian was ‘,…
    num2str(Fcount)])

disp([‘Number of solver iterations for fminunc with gradient and Hessian was
‘,num2str(output.iterations)])
```

Travail demandé

Copier les programmes précédents dans des fichiers séparés, exécuter chaque programme et remplir le tableau ci-dessous

Méthodes	Nombre d'évaluation de la fonction	Nombre d'évaluation du gradient	Nombre d'itérations du solveur
fminsearch			
fminunc			
la méthode de descente			
gradient analytique			
avec un Hessien analytique			

Interpréter les résultats

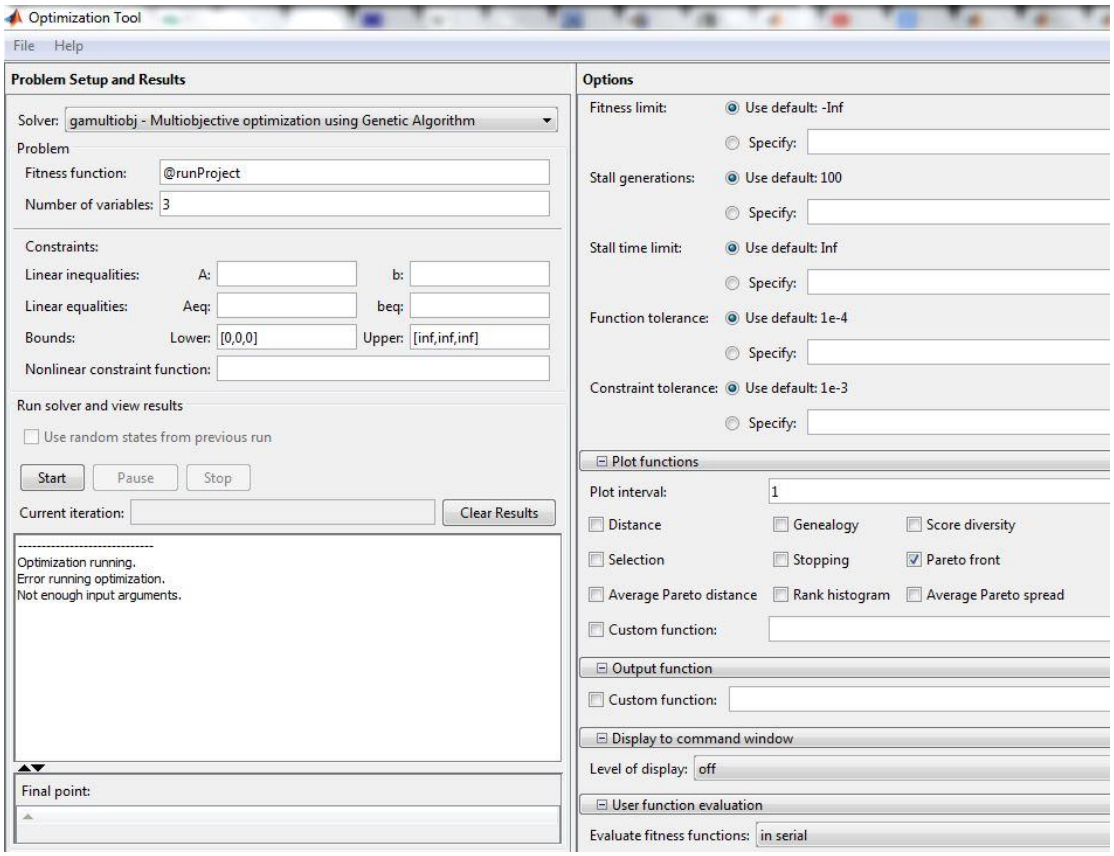
TP N° 05

Optimisation de fonctions continues non linéaires

(optimtool de Matlab)

1. Toolbox d'optimisation de Matlab (optimtool)

L'objectif de ces travaux pratiques est d'expérimenter les algorithmes vus en cours sur quelques problèmes non linéaires à l'aide de la toolbox d'optimisation de Matlab. La toolbox d'optimisation de Matlab dispose d'une interface graphique simplifiant l'utilisation et le paramétrage des algorithmes. Pour ouvrir cet outil, entrer la commande **optimtool** dans la fenêtre de commande (command window).



Le choix de l'algorithme d'optimisation est déterminé par la combinaison de plusieurs paramètres.

Pour commencer, nous allons utiliser l'algorithme BFGS, pour cela :

1. Dans le champ **Solver**, choisir **fminunc- Unconstrained nonlinear minimization**,
2. Dans le champ **Algorithm**, choisir **Medium scale** Il faut ensuite déterminer la fonction à minimiser. Nous allons nous exercer avec la fonction de **rosenbrock**:

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 1$$

Enregistré cette fonction dans le fichier rosenbrock.m.

```
%-------------------------------------------------------------
function [f,g]= rosenbrock(X)
f =100*(X(2)-X(1)^2)^2+(1-X(1))^2+1;
if nargout>1 % si le gradient est requis alors on le calcule
g =[100*(4* X(1)^3-4*X(1)*X(2))+2*X(1)-2
100*(2* X(2)-2*X(1)^2)];
end
end
%-------------------------------------------------------------
```

Il reste à indiquer que l'on souhaite minimiser cette fonction dans l'outil d'optimisation :
1. dans le champ Objective function, entrer @rosenbrock,
2. dans le champ Derivates, choisir Gradient supplied car le gradient est calculé par la fonction rosenbrock,
3. dans le champ Start point, entrer [-2 2], ce vecteur défini le point à partir duquel la minimisation va commencer.

On peut maintenant lancer la minimisation en cliquant sur le bouton Start. La fenêtre de résultat indique alors la valeur finale de la fonction ainsi que les raisons de l'arrêt de l'algorithme.

Ajout de tracés

Pour suivre, l´évolution de l'optimisation, il est utile de tracer certaines valeurs au cours des itérations.

Les graphiques proposés par Matlab n´étant pas très lisibles et incomplets, nous avons défini nos propres fonctions :

- La fonction plotrosenbrock permet de tracer la fonction de Rosenbrock avec le point courant et la direction de descente choisie.

```
% Rosenbrock's function
[x,y] = meshgrid(-1.5:.1:1.5, -1.5:.1:1.5);
f = (1 - x).^2 + 10 * (y - x.^2).^2;
v = [0.01,0.5,2,5,10,15,25,35,50,100,200];
[C, h] = contour(x, y, f, v);
clabel(C, h);
```

- La fonction plot Function Value permet de tracer la valeur de la fonction objectif au cours des itérations,
- la fonction plot Function Difference permet de tracer la valeur absolue de la différence entre deux évaluations successives de la fonction objectif (progression de la fonction objectif),
- la fonction plot Step Size permet de tracer la norme de la différence entre deux points successifs (progression de la variable de décision),

- la fonction plot Gradient Norm permet de tracer la valeur relative de la norme infinie du gradient.

Pour ajouter un tracé, il suffit d'ajouter le nom de la fonction précédé d'un @ dans le champ Custom function de l'onglet Plot functions. On peut utiliser plusieurs fonctions simultanément en les séparant par des virgules.

Critères d'arrêt

La toolbox d'optimisation de Matlab propose quatre critères d'arrêt :

– le critère Max iterations (MaxIter) limite le nombre d'itérations de l'algorithme. Si le nombre d'itérations atteint cette valeur, l'algorithme est stoppé.

– le critère Max function evaluations (MaxFunEvals) limite le nombre d'′evaluationsde la fonction objectif. Comme la fonction est évaluée plusieurs fois par itération, cette valeur doitêtre plus grande que MaxIter.

– le critère X tolerance (TolX) stoppe l'optimisation quand la norme de la différence entre deux points successifs (progression de la variable de décision) est inférieure à cette valeur. End'autres termes, l'algorithme s'arrête à l'itération k si :

$$\|x_k - x_{k-1}\| \leq TolX$$

– Le critère Function tolerance (TolFun) a un rôle double. Il stoppe l'optimisation quand la valeur absolue de la différence entre deux évaluations successives de la fonction objective (progression de la fonction objective) est inférieure à cette valeur, mais aussi quand la valeur relative de la norme infinie du gradient est inférieure à cette valeur. En d'autres termes, l'algorithme s'arrête à l'itération k si :

$$|f(x_k) - f(x_{k-1})| < TolFun \qquad ou \qquad \frac{\|\nabla f(x_k)\|_\infty}{1 + \|\nabla f(x_0)\|_\infty} < TolFun$$

A.1. Modifier les différents critères d'arrêt pour provoquer la sortie de l'algorithme pour chacun des critères.

Choix de l'algorithme

Pour utiliser l'algorithme de la plus forte pente, il suffit de choisir cette méthode dans le champ **Hessian update** de l'onglet **Approximated value**.

Pour utiliser l'algorithme de Nelder et Mead, choisir **fminsearch- Unconstrained nonlinear minimization** dans le champ **Solver**.

A.2. Observer le comportement des différents algorithmes dans l'espace de décision.
A.3. Compléter le tableau ci-dessous avec le nombre d'itérations minimal pour que la norme infinie du gradient soit inférieure à 1e-4.

Pour afficher le nombre d´évaluations de la fonction, choisir **iterativewithdetailed message** dans le champ **Level of display** de l'onglet **Display to command window**. Les résultats s'inscrivent dans la fenêtre de commande (**command window**) de Matlab.

Méthodes	Nombre d'itérations	Nombre d'évaluation de la fonction
Plus forte pente (steepest)		
BFGS		
BFGS avec estimation du gradiant		
Nelder et Mead		

TP N°6

Optimisation non-linéaires avec contraintes

- Programmation quadratique séquentielle

Fonction fmincon (optimization toolbox)

La fonction *fmincon* code de plusieurs méthodes (dont SQP) de résolution de problème d'optimisation multidimensionnelle avec contraintes. L'utilisateur avisé peut choisir la méthode numérique ou sinon laisser la fonction fmincon choisir automatiquement la méthode la plus adaptée. La syntaxe générale de cette fonction peut faire appel à une fonction externe (premier cas) ou à une fonction anonyme (second cas) :

[x,fval,exitflag,output,lambda,grad,hessian] = fmincon ('fun' ,x0, A, B, Aeq, Beq, lb, ub, 'nonlcon', option)

ou

[x,fval,exitflag,output,lambda,grad,hessian] = fmincon(@(x) fun(x), x0, A, B, Aeq, Beq, lb, ub, 'nonlcon',option)

Arguments d'entrée :

- *fun :* dans le premier cas, fun est une fonction externe où est exprimée la fonction objectif qu'on cherche à minimiser. Dans le second cas, fun est une fonction anonyme où est exprimée la fonction objectif qu'on cherche à minimiser. Entête de la fonction fun : function f = fun(x) ;

- *x0* : vecteur des valeurs initiales pour démarrer la méthode. La dimension de x0 est égale au nombre de variables du problème ;

- *A* et *B* : respectivement une matrice et un vecteur qui permettent d'exprimer les contraintes linéaires de type inégalité sous la forme : A*x <= B.

  A et *B* ont autant de lignes que de contraintes inégalité linéaires ; A a autant de colonnes que le nombre de variables de contrôle ;

- *Aeq* et *Beq* : respectivement une matrice et un vecteur qui permettent d'exprimer les contraintes linéaires de type égalité sous la forme : Aeq*x = Beq.

  Aeq et *Beq* ont autant de lignes que de contraintes égalité linéaires ; Aeq a autant de colonnes que le nombre de variables de contrôle ;

- *lb* et *ub* : bornes inférieure et supérieure des éléments de x. lb et ub ont la même dimension que x ;

- *'nonlcon'* : nom de la fonction externe MATLAB dans laquelle sont définies les éventuelles contraintes non-linéaires égalité et inégalité. L'entête de nonlcon.m est : function [c,ceq] = nonlcon(x)

c est un vecteur qui contient les résidus des contraintes inégalité (toutes les composantes de c doivent être négatives à la solution).

ceq est un vecteur qui contient les résidus des contraintes égalité (toutes les composantes de ceq doivent être nulles à la solution).

nonlcon peut être également définie comme une fonction anonyme ;

- options (pas obligatoire) ; voir ci-dessous.

Arguments de sortie :

- *x* : valeur finale des variables, ce sont les solutions si la méthode a convergé ;
- *fval* : valeur finale de la fonction à minimiser ;
- *exitflag* (pas obligatoire) : indicateur de fin d'exécution : 1 arrêt car solution trouvée, 0 car nombre d'itérations maximal atteint ; il existe d'autres valeurs possible.
- output (pas obligatoire) : nombres d'itérations et d'évaluations de fonction réalisés.
- *lambda* : structure qui contient la valeur des multiplicateurs de Lagrange à la valeur finale de x
- *grad* : vecteur gradient à la valeur finale de x
- *hessian* : matrice hessienne à la valeur finale de x

Options

Les options par défaut peuvent être modifiées par l'intermédiaire de la fonction optimset qui est utilisée ainsi (par exemple avec fminbnd) :

options = optimset('propriete1',valeur1, 'propriete2',valeur2, etc ...)

[x,fval,exitflag,output] = fminbnd('fun',x1,x2,options)

Les principales options modifiables sont entre autres :

- *'display'* : on choisira la valeur 'iter' pour la propriété 'display' afin d'avoir des informations sur le processus de convergence affichées à l'écran à chaque itération. Par défaut, les affichages ne sont proposés qu'en fin d'exécution.
- *'MaxIter'* : si le nombre d'itérations n'est pas suffisant, il peut être modifié en choisissant une valeur entière plus grande.
- *'MaxFunEvals'* : si le nombre d'évaluations de la fonction objectif n'est pas suffisant, il peut être modifié en choisissant une valeur entière plus grande.

- *'tolX'* : si on veut changer le critère d'arrêt basé les variations des variables de contrôle, il peut être modifié en choisissant une valeur réelle (plus cette valeur est petite, plus les tolérances sont serrées).

- *'Diagnostics'* : on choisira la valeur ‘on' pour avoir un diagnostic du problème traité avant le début des calculs itératifs.

Exemple d'utilisation : fmincon

On cherche à résoudre le problème suivant :

$$\min \quad -x1 \times x2$$

$$h(x1, x2) = 20x1 + 15x2 - 30 = 0$$

$$g(x1, x2) = (x1/2)2 + (x2)2 - 1 <= 0$$

$$\text{et } 0 <= x1 \; et \; 0 <= x2$$

C'est un problème avec une contrainte linéaire égalité, une contrainte non-linéaire inégalité et des bornes sur les variables de contrôle.

```
x0=[0.5 0.8] ; % valeurs initiales pour x1 et x2 prises respectivement égales
à 0.5 et 0.8
lb=[0 0] ; % definition des bornes inférieures
ub=[ ] ; % pas de bornes supérieures
Aeq=[20 15] ; Beq=[3 0] ; % définition des coefficients de la première
contrainte linéaire égalité
A=[ ] ; B=[ ] ; % pas contrainte linéaire inégalité
options = optimoptions(@fmincon,'Algorithm','sqp');
fun=@(x) -x(1)*x(2) ; % definition de la fonction objectif (intermédiaire
d'une fonction anonyme)
[x,fval,exitflag,output] = fmincon(@(x) fun(x),x0,A,B,Aeq,Beq,lb,ub,'nonlcon')
```

Dans une fonction externe nonlcon.m, va être définie la contrainte non linéaire inégalité :

```
function [c,ceq]=nonlcon(x)
ceq = [] ; % pas de contrainte non linéaire égalité
c(1)=(x(1)/2)^2+x(2)-1 ; % definition de la contrainte non lineaire égalité
sous la fome c(x) <=0
```

Travail à faire

Résoudre le problème suivant

1- Contrainte d'Inégalité Linéaire

Trouvez la valeur minimale de la fonction de Rosenbrock quand il y a une contrainte d'inégalité linéaire.

$f(x) = 100 * (x_2 - x_1^2)^2 \; + \; (1 - x_1)^2;$

Trouvez la valeur minimale en commençant du point [-1,2],

Contrainte $x_1 + 2x_2 \leq 1$. Exprimez cette contrainte sous forme Ax = b en prenant A = [1,2] et b = 1. Remarquez que cette contrainte signifie que la solution ne sera pas à la solution (1,1)

2- Inégalité Linéaire et Contrainte d'Égalité

Trouvez la valeur minimale de la même fonction en commençant du point [0.5,0],

Contrainte $x_1 + 2x_2 \leq 1$; $2x_1 + x_2 = 1$

Exprimez la contrainte d'inégalité linéaire sous la forme A*x = b en prenant A = [1,2] et b = 1.

Exprimez la contrainte d'égalité linéaire sous la forme Aeq*x = beq en prenant Aeq = [2,1] et beq = 1.

3- Contraintes bornées

Trouvez le minimum de la fonction objectif suivante

$$f(x) = x_1 + \frac{x_1}{(1 + x_2)} - 3x_1x_2 + x_2(1 + x_1)$$

En présence de contraintes bornées. $x_1 \leq 1$, et $x_2 \leq 2$.