Méthodes Numériques : Optimisation

Cours de L3, 2020-2021
Université Paris-Dauphine

David Gontier

(version du 7 mai 2021).

TABLE DES MATIÈRES

INTRODUCTION

Dans ce cours [1], nous étudions des algorithmes pour résoudre deux types de problèmes : des problèmes *d'optimisation* et des problèmes *de résolution*. Un problème d'optimisation est un problème de la forme

$$\mathbf{x}^* = \operatorname{argmin} \left\{ F(\mathbf{x}),\ \mathbf{x} \in \mathbb{R}^d \right\},$$

où F est une fonction de $\mathbb{R}^d$ dans $\mathbb{R}$, et les problèmes *de résolution* sont de la forme

$$\text{Trouver} \quad \mathbf{x}^* \in \mathbb{R}^d \quad \text{solution de} \quad f(\mathbf{x}) = \mathbf{0},$$

où f est une fonction de $\mathbb{R}^d$ dans $\mathbb{R}^d$ (c'est à dire d équations à d inconnues).

Notre but n'est pas de trouver des *solutions analytiques* à ces problèmes (de type $x^* = \frac{\sqrt{2}}{3} + \ln(2)$), mais de trouver des solutions *numériques* (de type $x^* = 1.16455170135...$). Dans la plupart des applications, c'est la connaissance d'une solution numérique qui est intéressante [2].

Nous présentons différents *algorithmes de résolution itératifs*. Un tel algorithme génère une suite $(\mathbf{x}_n)$ qui converge vers $\mathbf{x}^*$, ce qui permet de trouver une solution numérique avec une précision arbitraire. Nous prouverons la convergence de ces algorithmes, et étudierons leur *vitesse de convergence*.

Ce cours sera illustré par de nombreux exemple en PYTHON3.

La première et principale partie du cours concerne les problèmes d'optimisation *sans contraintes*. Nous abordons les algorithmes de type descente de gradient, la méthode du gradient conjugué, et les méthodes de type Newton ou BFGS.

Dans la seconde partie du cours, nous exposons certains algorithmes pour des problèmes d'optimisation *avec contraintes*. Nous introduisons les méthodes de pénalisation et de gradient projeté.

1. Page web du cours ici.
2. Il existe des équations sans solution analytique. Par exemple, comme le montre la théorie d'Abel-Galois, l'équation $x^5 - 3x - 1 = 0$ n'admet pas de solutions par radicales (s'écrivant avec les opérations élémentaires).

Jupyter

Installation

Les exemples et les TPs de ce cours se feront sur l'interface[3] *Jupyter*, qui est un outil qui permet d'écrire du PYTHON sur un navigateur Web. Pour installer Jupyter, il faut d'abord installer PYTHON3, puis lancer dans un Terminal la commande

```
> pip3 install jupyter
```

Cette commande permet d'installer l'environnement *Jupyter*. Il faut aussi charger la librairie de calculs numériques et la librairie graphique, avec

```
> pip3 install numpy, matplotlib
```

Une fois l'installation terminée, on peut lancer directement un *notebook* avec la commande[4]

```
> cd DossierOuEstLeNotebook/
> jupyter notebook
```

Plusieurs lignes s'affichent dans le Terminal, dont une ligne de la forme

```
Copy/paste this URL into your browser when you connect for the first time,
to login with a token:
    http://localhost:8890/?token=f432a30bc62d1af678101656a61dc1e69d38090df31e4674
```

Si le navigateur Web ne se lance pas directement, lancez en un (Firefox, Safari, Chrome,...), et copier l'adresse précédente comme url.

Premier fichier

Pour créer un nouveau fichier, cliquer sur "New" (en haut à droite), et sélectionner PYTHON3. Une page s'ouvre. Cette page, appelée *notebook*, sera structurée en *cellules*. Dans la première cellule, on écrira **toujours**

```
1 %pylab inline
```

Cette ligne permet de charger les librairies `numpy` (fonctions mathématiques) et `matplotlib` (affichage graphique). On exécute la cellule avec *shift+enter*. Pour vérifier que tout marche bien, on peut écrire dans la deuxième cellule

```
1 tt = linspace(0,1,100)
2 def f(x) : return sin(2*pi*x)
3 plot(tt, f(tt))
```

puis valider avec *shift+enter*. Une sinusoïde devrait apparaître.

Loin de mon ordinateur ? Projet à plusieurs ?

Il existe maintenant des serveurs Jupyter en ligne. On peut citer colab.research.google.com si vous avez un compte Google, et notebooks.azure.com si vous avez un compte Microsoft. Il y a aussi cocalc.com, qui permet de faire du LaTeX(voir documentation ici).

3. Le nom «Jupyter» est une concaténation de Julia-Python-R.
4. En lançant la commande `jupyter notebook`, on créée un *serveur* sur l'ordinateur (localhost). Vous pouvez ensuite dialoguez avec ce serveur grâce à votre navigateur Web. Si vous fermez le Terminal, vous fermez le serveur, et votre notebook ne marche plus !

CHAPITRE 1

GÉNÉRALITÉS

1.1 Conventions

Dans ce manuscrit, nous emploierons systématiquement les lettres $x, y, z, \ldots \in \mathbb{R}$ lorsqu'il s'agit de variables réelles, et $\mathbf{x}, \mathbf{y}, \mathbf{z}, \ldots \in \mathbb{R}^d$ lorsqu'il s'agit de vecteurs de $\mathbb{R}^d$. Les composantes de $\mathbf{x} \in \mathbb{R}^d$ sont notées $\mathbf{x} = (x_1, \ldots, x_d)$.

Nous étudions dans ce cours deux types de problèmes : des **problèmes d'optimisation** de type

$$\text{argmin}\{F(\mathbf{x}),\ \mathbf{x} \in \mathbb{R}^d\}, \tag{1.1}$$

auquel cas nous utiliserons la lettre majuscule F, et F sera une fonction de $\mathbb{R}^d$ dans $\mathbb{R}$, toujours à valeurs réelles (que signifie *optimiser* F sinon ?), et des **problèmes de résolutions** de type

$$f(\mathbf{x}) = \mathbf{0}, \tag{1.2}$$

auquel cas nous utiliserons la lettre minuscule f, qui sera une fonction de $\mathbb{R}^d$ dans $\mathbb{R}^d$ (autant de variables que d'équations). Dans les deux cas, nous noterons $\mathbf{x}^* \in \mathbb{R}^d$ la solution, ou solution locale, de (1.1)-(1.2).

1.2 Calcul de dérivées par différences finies

Dans tout ce cours, il sera important de calculer des dérivées de fonctions, comme le montre l'exemple suivant.

Supposons que nous avons déjà construit un algorithme efficace pour résoudre des problèmes de type (1.2). On pourrait alors utiliser cet algorithme pour résoudre des problèmes de type (1.1) avec [1] $f = \nabla F$. Autrement dit, si on sait résoudre $f = \mathbf{0}$, on sait optimiser. Cela nécessite cependant de savoir calculer le gradient (donc les dérivées partielle) de F.

Lorsque ∇F est donnée par des formules explicites, on peut directement entrer cette formule dans l'ordinateur. Malheureusement, il arrive souvent que les dérivées de F ne soit pas explicites. Cela arrive par exemple lorsque F est déjà le résultat d'un calcul complexe. On peut cependant approcher les dérivées numériquement par des *différences finies*.

1. On rappelle que si $F : \mathbb{R}^d \to \mathbb{R}$ est de classe $\mathbb{C}^1$, alors, pour tout $\mathbf{x} \in \mathbb{R}^d$, $F'(\mathbf{x})$ est une application linéaire de $\mathbb{R}^d$ dans $\mathbb{R}$. Le gradient de F est l'unique élément $(\nabla F)(\mathbf{x})$ de $\mathbb{R}^d$ tel que, pour tout $\mathbf{h} \in \mathbb{R}^d$, $F'(\mathbf{x}) \cdot \mathbf{h} = \langle \nabla F(\mathbf{x}), \mathbf{h} \rangle$ (produit scalaire de $\mathbb{R}^d$). L'application $\mathbf{x} \mapsto \nabla F$ est continue de $\mathbb{R}^d$ dans $\mathbb{R}^d$, et on a $\nabla F(\mathbf{x}) = (\partial_{x_1} F, \cdots, \partial_{x_d} F)^T$.

Définition 1.1. *Soit $F : \mathbb{R}^n \to \mathbb{R}$ de classe C^1. L'approximation de $f := \nabla F$ par* **différence finie** *(resp.* **différence finie centrée***) d'ordre $\varepsilon > 0$ est la fonction $\widetilde{f}_\varepsilon$ (resp. f_ε) de $\mathbb{R}^n$ dans $\mathbb{R}^n$ dont la i-ème composante est définie par (on note $\mathbf{e}_i$ le i-ème vecteur canonique de $\mathbb{R}^n$)*

$$\left(\widetilde{f}_\varepsilon\right)_i(\mathbf{x}) := \frac{F(\mathbf{x}+\varepsilon\mathbf{e}_i) - F(\mathbf{x})}{\varepsilon} \quad \textit{resp.} \quad (f_\varepsilon)_i(\mathbf{x}) := \frac{F(\mathbf{x}+\varepsilon\mathbf{e}_i) - F(\mathbf{x}-\varepsilon\mathbf{e}_i)}{2\varepsilon}.$$

Exercice 1.2

Montrer que si $F : [a,b] \to \mathbb{R}$ est de classe C^∞, alors pour tout $\varepsilon > 0$, et pour tout $x \in (a+\varepsilon, b-\varepsilon)$,

$$\left|\frac{F(x+\varepsilon)-F(x)}{\varepsilon} - F'(x)\right| \le \varepsilon\left(\frac{1}{2}\sup_{[a,b]}|F''|\right) \quad \text{et} \quad \left|\frac{F(x+\varepsilon)-F(x-\varepsilon)}{2\varepsilon} - F'(x)\right| \le \varepsilon^2\left(\frac{1}{6}\sup_{[a,b]}|F'''|\right).$$

En pratique, les calculs numériques sont effectués avec une certaine précision η : un ordinateur ne peut retenir qu'un nombre fini de chiffres significatifs. Dans la norme IEEE 754 par exemple (celle qu'utilise Python3), un réel en double précision est codé sous la forme

$$x = (-1)^s \cdot J \cdot 2^{(-1)^p N},$$

où $s \in \{0,1\}$ est le signe de x, J est un entier codé en base binaire avec 52 bits (donc $0 \le J \le 2^{52}-1$), $p \in \{0,1\}$, et N est un entier codé en base binaire avec 10 bits (donc $0 \le N \le 2^{10}-1$). Au total, x est codé sur $1+52+1+10 = 64$ bits.

Exercice 1.3

Dans la norme IEEE 754,
a/ Quelles sont les valeurs de (s, J, p, N) pour les nombres 0 et 1 ?
b/ Quelle la valeur numérique strictement plus grande que 0 ? et que 1 ?

Ainsi, pour η suffisamment petit [2], un ordinateur ne fait pas la différence entre $1+\eta$ et 1, mais fait la différence entre 0 et η. Une première conséquence de ce phénomène est que l'addition $+_\eta$ numérique est commutative (on a toujours $a +_\eta b = b +_\eta a$), mais n'est pas associative. On a par exemple :

$$-1 +_\eta (1 +_\eta \eta) = -1 +_\eta 1 = 0, \quad \text{mais} \quad ((-1) +_\eta 1) +_\eta \eta = 0 +_\eta \eta = \eta.$$

Une autre conséquence est que les nombres réels ne sont calculés qu'approximativement. Ainsi, la fonction mathématique F est approchée numériquement par une fonction F_η telle que

$$\forall \mathbf{x} \in \mathbb{R}^n,\ |F_\eta(\mathbf{x}) - F(\mathbf{x})| \le \eta.$$

En particulier, la différence $F(x+\varepsilon) - F(x)$ est d'ordre η si $\varepsilon \ll \eta$.

Exercice 1.4

a/ Avec les mêmes hypothèses que l'Exercice 1.2, montrer qu'on a

$$\left|\frac{F_\eta(x+\varepsilon)-F_\eta(x)}{\varepsilon} - F'(x)\right| \le \varepsilon\left(\frac{1}{2}\sup_{[a,b]}|F''|\right) + 2\frac{\eta}{\varepsilon}, \quad \text{(différence finie numérique)},$$

$$\left|\frac{F_\eta(x+\varepsilon)-F_\eta(x-\varepsilon)}{2\varepsilon} - F'(x)\right| \le \varepsilon^2\left(\frac{1}{6}\sup_{[a,b]}|F'''|\right) + \frac{\eta}{\varepsilon} \quad \text{(différence finie centrée numérique)}.$$

b/ Soit $c_1, c_2 > 0$. Calculer en fonction de η le minimiseur et le minimum des fonctions

$$e_1(\varepsilon) := c_1\varepsilon + c_2\frac{\eta}{\varepsilon} \quad \text{et} \quad e_1(\varepsilon) := c_1\varepsilon^2 + c_2\frac{\eta}{\varepsilon}.$$

c/ En déduire qu'il faut choisir $\varepsilon = O(\sqrt{\eta})$ pour les différences finies, et $\varepsilon = O(\eta^{1/3})$ pour les différences finis centrées.

2. Un rapide calcul montre que $\eta \approx 2^{-52} \approx 2.2 \times 10^{-16}$. Un ordinateur ne peut retenir que 16 chiffres significatifs au maximum.

En pratique, on a $\eta \approx 10^{-16}$. On retiendra qu'on pourra approcher ∇F par des différences finies centrées en choisissant $\varepsilon = 10^{-5}$, et qu'on aura une erreur numérique de l'ordre de 10^{-10}.

1.3 Algorithmes itératifs

Nous étudions dans ce cours exclusivement des algorithmes *itératifs*. Ces algorithmes construisent une suite $(\mathbf{x}_n)_{n\in\mathbb{N}}$ qui converge vers $\mathbf{x}^*$. Ils sont construits selon le modèle suivant :

Code 1.1 – Structure d'un algorithme itératif

```
1  def algo(F, x0, ..., tol=1e-6, Niter=1000):
2      # Initialisation
3      xn = x0      # premier élément de la suite
4      L = []       # la liste [x_0, ... x_{n-1}], pour le moment vide
5
6      # Boucle principale
7      for n in range(Niter):  # pas de while dans ce cours !
8          if ... < tol:       # Si le critère de convergence est atteint,
9              return xn, L    # on renvoie xn et la liste [x_0, ... x_{n-1}].
10         L.append(xn)        # Sinon, on ajoute xn à la liste,
11         xn = ...            # puis on met à jour xn avec la formule d'itération
12     print("Erreur, l'algorithme n'a pas convergé après ", Niter, " itérations")
```

Faisons quelques commentaires sur ce code.

- l.1 : le code `algo` prend en entrée la fonction F, un point d'initialisation x_0, éventuellement d'autres paramètres, et enfin deux paramètres d'arrêt : une tolérance `tol` (dont la valeur par défaut est ici 10^{-6}), et un nombre d'itérations maximal `Niter` (dont la valeur par défaut est ici 1000).
- l.2 : le code initialise le point x_n avec $x_{n=0} = x_0$, et la liste $L = [x_0, \dots x_{n-1}]$ avec $L_{n=0} = []$ (pour le moment vide).
- l.7 : dans la boucle principale, n varie entre dans `range(Niter)` $= [0, 1, \dots, N_{\text{iter}} - 1]$. À l'itération n, on calcule le n-ème élément de la suite (x_n).
  Attention : On n'utilisera jamais de boucles `while` dans ce cours.
- l.8-9 : on commence par vérifier si le *critère d'arrêt* est atteint, auquel cas on peut sortir de la boucle en renvoyant x_n (le dernier élément calculé), et $L = [x_0, \dots x_{n-1}]$. Nous verrons plusieurs types de critère d'arrêt. Dans ce cours, nous renvoyons systématiquement les itérations précédentes (la liste L). Cela nous permet d'étudier la qualité de l'algorithme (= sa vitesse de convergence, cf. chapitre suivant).
- l.10-11 : si le critère n'est pas vérifié, on met à jour la liste L, et on calcule x_{n+1} avec une *formule d'itération*, qui dépend de l'algorithme. On nomme toujours ce nombre x_n (au lieu de x_{n+1}) : implicitement, on passe au rang $n+1$ entre la ligne 10 et 11.
- l.12 : si l'algorithme sort de la boucle, cela signifie que $n = N_{\text{iter}}$: on a atteint le nombre maximal d'itérations autorisées. On dit dans ce cas que l'algorithme *n'a pas convergé*.

1.4 Vitesse de convergence

La *vitesse de convergence* d'un algorithme itératif est la vitesse de convergence de la suite $(\mathbf{x}_n)$ vers $\mathbf{x}^*$. Cette vitesse peut-être très différente de la vitesse réelle de l'algorithme, qui prend aussi en compte le temps que met l'ordinateur à évaluer la fonction F par exemple. Cependant, le temps physique de l'algorithme est évidemment proportionnel à son nombre d'itérations. Nous parlerons de l'efficacité réelle d'un algorithme dans la section suivante.

1.4.1 Convergence linéaire

Définition 1.5 : Convergence linéaire

On dit qu'une suite $(\mathbf{x}_n)$ converge **linéairement à taux au plus** $0 < \alpha < 1$ vers $\mathbf{x}^*$ s'il existe $C \in \mathbb{R}^+$ tel que

$$\forall n \in \mathbb{N}, \quad |\mathbf{x}_n - \mathbf{x}^*| \leq C\alpha^n. \tag{1.3}$$

La borne inférieure des $0 < \alpha < 1$ vérifiant cette propriété est appelée **taux de convergence** de la suite.
Si (1.3) est vérifiée pour tout $\alpha > 0$, on dit que la convergence est **super-linéaire**.

Par exemple, si $C = 1$ et $\alpha = \frac{1}{10}$, on a $|\mathbf{x}_n - \mathbf{x}^*| \leq 10^{-n}$. Cela signifie qu'on gagne une bonne décimale par itérations. On retiendra qu'un algorithme qui converge linéairement gagne un nombre fixe de bonnes décimales par itération.

Si $(\mathbf{x}_n)$ converge linéairement vers $\mathbf{x}^*$ à taux α, alors on a

$$\ln(|\mathbf{x}_n - \mathbf{x}^*|) \approx n \cdot \ln(\alpha).$$

Ainsi, pour mettre en évidence une convergence linéaire d'un algorithme, on tracera $\ln(\mathbf{x}_n - \mathbf{x}^*)$ en fonction de n. On observe une droite, dont le coefficient directeur est $\ln(\alpha)$ (donc négatif, car $0 < \alpha < 1$). En pratique, on pourra utiliser la fonction `semilogy` de Python, par exemple avec le code suivant :

Code 1.2 – Illustrer une convergence linéaire

```
1 xstar, L = algo(F, ...)
2 N = len(L) # nombre d'itérations
3 nn = range(N) # la liste [0, 1, ... N-1]
4 errors = [abs(xn - xstar) for xn in L] # la liste | xn - x* |
5 semilogy(nn, errors)
6
7 # On trouve le taux avec la fonction polyfit
8 p = polyfit(nn, log(errors), 1)
9 print("Convergence linéaire à taux %.3f"%exp(p[0]))
```

Un exemple de figure donné par ce code est donné plus loin, en Figure 2.1. Quelques remarques sur le code précédent.

- l.5. La fonction `semilogy(x,y)` trace la même courbe que `plot(x,log(y))` (graphe de $(x, \ln(y))$), mais a l'avantage de changer l'échelle des y pour qu'elle soit plus lisible (voir Figure 2.1).
- l.8. La fonction `polyfit(x,y,k)` cherche la *meilleure approximation polynomiale de degré k* qui interpole le graphe (x, y). Ici, on veut une approcher la courbe par une droite, d'où l'approximation polynomiale de degré 1. Le coefficient dominant est $p[0]$.

Pour démontrer une convergence linéaire, on utilise souvent le critère suivant.

Lemme 1.6 : Condition pour la convergence linéaire

Soit $(\mathbf{x}_n)$ une suite de $\mathbb{R}^d$. On suppose qu'il existe $0 < \alpha < 1$ tel que, pour tout $n \in \mathbb{N}^*$, on a $\|\mathbf{x}_{n+1} - \mathbf{x}_n\| \leq \alpha \|\mathbf{x}_n - \mathbf{x}_{n-1}\|$. Alors il existe $\mathbf{x}^* \in \mathbb{R}^d$ tel que la suite $(\mathbf{x}_n)$ converge vers $\mathbf{x}^*$, et on a

$$\forall n \in \mathbb{N}^*, \quad \|\mathbf{x}_n - \mathbf{x}^*\| \leq \frac{\alpha^n}{1-\alpha} \|\mathbf{x}_1 - \mathbf{x}_0\|. \tag{1.4}$$

En particulier, la suite $(\mathbf{x}_n)$ converge linéairement vers $\mathbf{x}^*$ à un taux inférieur ou égale à α.

Démonstration. On commence par remarquer qu'on a, par une récurrence immédiate, l'inégalité

$$\forall n \in \mathbb{N}, \quad \|\mathbf{x}_{n+1} - \mathbf{x}_n\| \leq \alpha \|\mathbf{x}_n - \mathbf{x}_{n-1}\| \leq \cdots \leq \alpha^n \|\mathbf{x}_1 - \mathbf{x}_0\|.$$

De plus, pour tout $n, p \in \mathbb{N}$, on a

$$\begin{aligned}\|\mathbf{x}_{n+p} - \mathbf{x}_n\| &\leq \|\mathbf{x}_{n+p} - \mathbf{x}_{n+p-1}\| + \|\mathbf{x}_{n+p-1} - \mathbf{x}_{n+p-2}\| + \cdots + \|\mathbf{x}_{n+1} - \mathbf{x}_n\| \\ &\leq \left(\alpha^{n+p} + \alpha^{n+p-1} + \cdots + \alpha^n\right) \|\mathbf{x}_1 - \mathbf{x}_0\| = \alpha^n \left(\alpha^p + \alpha^{p-1} + \cdots + 1\right) \|\mathbf{x}_1 - \mathbf{x}_0\| \\ &\leq \alpha^n \left(\sum_{k=0}^{\infty} \alpha^k\right) \|\mathbf{x}_1 - \mathbf{x}_0\| = \frac{\alpha^n}{1-\alpha} \|\mathbf{x}_1 - \mathbf{x}_0\|. \end{aligned} \tag{1.5}$$

On en déduit que $\|\mathbf{x}_{n+p} - \mathbf{x}_n\|$ converge vers 0 lorsque $n \to \infty$, uniformément en $p \in \mathbb{N}$. Ainsi, la suite $(\mathbf{x}_n)$ est de Cauchy. L'espace $\mathbb{R}^n$ étant complet, elle converge vers une certaine limite $\mathbf{x}^* \in \mathbb{R}^n$. En faisant tendre p vers l'infini dans (1.5), on obtient (1.4). □

1.4.2 Convergence quadratique

Définition 1.7 : Convergence quadratique

On dit qu'une suite $(\mathbf{x}_n)$ converge **à l'ordre au moins** $\beta > 1$ vers $\mathbf{x}^*$ s'il existe $C \in \mathbb{R}^+$ et $0 < \alpha < 1$ tel que

$$\forall n \in \mathbb{N}, \quad |\mathbf{x}_n - \mathbf{x}^*| \leq C\alpha^{\beta^n}.$$

Le plus grand β vérifiant cette propriété est appelé **ordre de convergence** de la suite. Si $\beta = 2$, on parle de **convergence quadratique**.

Par exemple, si (x_n) converge quadratiquement vers x^* avec $C = 1$ et $\alpha = \frac{1}{10}$, on a $|\mathbf{x}_n - \mathbf{x}^*| \leq 10^{-2^n}$. Autrement dit, on double le nombre de bonnes décimales à chaque itération. Plus généralement, on retiendra qu'une convergence à l'ordre β multiplie par β le nombre de bonnes décimales à chaque itération.

Si $(\mathbf{x}_n)$ converge à l'ordre β vers $\mathbf{x}^*$, on a cette fois

$$\ln(|\ln(|\mathbf{x}_n - \mathbf{x}^*|)|) \approx n \cdot \ln \beta.$$

Contrairement au cas de la convergence linéaire, on ne trace pas cette fois la fonction $\ln(|\ln(|\mathbf{x}_n - \mathbf{x}^*|)|)$ (on ne verra presque jamais une belle droite). Ainsi, en pratique, on montrera numériquement que les convergences sont super-linéaires en montrant qu'elle décroissent plus vite que des droites en échelle semilog. On peut aussi vérifier que le nombre de bonnes décimales est multiplié par un certain facteur à chaque itération (voir Figure 2.3).

Le critère suivant permet de démontrer une convergence à l'ordre β.

Lemme 1.8 : Condition pour la convergence quadratique

Soit $(\mathbf{x}_n)$ une suite de $\mathbb{R}^d$. On suppose qu'il existe $C \geq 0$ et $\beta > 1$ tel que

$$\forall n \in \mathbb{N}^*, \quad \|\mathbf{x}_{n+1} - \mathbf{x}_n\| \leq C \|\mathbf{x}_n - \mathbf{x}_{n-1}\|^\beta, \quad \text{et} \quad \alpha := C^{\frac{1}{\beta-1}} \|\mathbf{x}_1 - \mathbf{x}_0\| < 1.$$

Alors il existe $\mathbf{x}^* \in \mathbb{R}^d$ tel que $(\mathbf{x}_n)$ converge à l'ordre β vers $\mathbf{x}^*$. Plus spécifiquement,

$$\|\mathbf{x}_n - \mathbf{x}^*\| \leq \widetilde{C}\alpha^{\beta^n}, \quad \text{avec} \quad \widetilde{C} := \frac{1}{1-\alpha^{\beta-1}} \frac{1}{C^{\frac{1}{\beta-1}}}.$$

Démonstration. Par récurrence, on obtient que pour tout $n \in \mathbb{N}$,

$$\begin{aligned}\|\mathbf{x}_{n+1} - \mathbf{x}_n\| &\leq C\|\mathbf{x}_n - \mathbf{x}_{n-1}\|^\beta \leq C\left(C\|\mathbf{x}_{n-1} - \mathbf{x}_{n-2}\|^\beta\right)^\beta = C^{1+\beta}\|\mathbf{x}_{n-1} - \mathbf{x}_{n-2}\|^{\beta^2} \\ &\leq \cdots \leq C^{1+\beta+\cdots+\beta^{n-1}} \|\mathbf{x}_1 - \mathbf{x}_0\|^{\beta^n} = C^{\frac{\beta^n-1}{\beta-1}} \|\mathbf{x}_1 - \mathbf{x}_0\|^{\beta^n} = C^{\frac{-1}{\beta-1}} \alpha^{\beta^n}.\end{aligned}$$

Montrons que la suite $(\mathbf{x}_n)$ est de Cauchy. On a

$$\begin{aligned}\|\mathbf{x}_{n+p} - \mathbf{x}_n\| &\leq \|\mathbf{x}_{n+p} - \mathbf{x}_{n+p-1}\| + \|\mathbf{x}_{n+p-1} - \mathbf{x}_{n+p-2}\| + \cdots + \|\mathbf{x}_{n+1} - \mathbf{x}_n\| \\ &\leq C^{\frac{-1}{\beta-1}} \left(\alpha^{\beta^{n+p-1}} + \alpha^{\beta^{n+p-2}} + \cdots + \alpha^{\beta^n}\right)\end{aligned}$$

Ainsi, pour montrer que $(\mathbf{x}_n)$ est de Cauchy, il suffit de montrer que la série $(\alpha^{\beta^{n+k}})$ est convergente. Pour tout $m \in \mathbb{N}$, la fonction $x \mapsto (1+x)^m - 1 - mx$ est positive sur $\mathbb{R}_+$ (elle s'annule en $x = 0$, et sa dérivée est positive sur $\mathbb{R}_+$). En particulier, avec $x = \beta - 1$, on a $\beta^k \geq 1 + k(\beta - 1)$, et donc

$$\alpha^{\beta^{n+k}} = \alpha^{\beta^n \beta^k} \leq \alpha^{\beta^n(1+k(\beta-1))} = \alpha^{\beta^n + \beta^n(\beta-1)k} \leq \alpha^{\beta^n + (\beta-1)k} = \alpha^{\beta^n} \alpha^{(\beta-1)k}.$$

En remplaçant dans l'inégalité précédente, on obtient

$$\|\mathbf{x}_{n+p} - \mathbf{x}_n\| \leq C^{\frac{-1}{\beta-1}} \alpha^{\beta^n} \left(\sum_{k=0}^{\infty} \alpha^{(\beta-1)k}\right) = C^{\frac{-1}{\beta-1}} \alpha^{\beta^n} \frac{1}{1 - \alpha^{(\beta-1)}}.$$

Comme α^{β^n} converge vers 0, ceci montre que $(\mathbf{x}_n)$ est une suite de Cauchy, et converge vers une limite $\mathbf{x}^* \in \mathbb{R}^d$. De plus, en faisant tendre p vers l'infini, on obtient l'inégalité souhaitée. □

Exercice 1.9

Soit $\Phi : \mathbb{R} \to \mathbb{R}$ de classe C^2, et soit x^* une solution du problème de point fixe $\Phi(x^*) = x^*$. On suppose qu'il existe $0 < \alpha < 1$ tel que $|\Phi'(x^*)| < \alpha$.
a/ Montrer qu'il existe $\varepsilon > 0$ tel que pour tout $x \in (x^* - \varepsilon, x^* + \varepsilon)$, on a $|\Phi'(x)| \leq \alpha$.
b/ Soit $x_0 \in (x^* - \varepsilon, x^* + \varepsilon)$, et soit (x_n) la suite définie par récurrence par $x_{n+1} = \Phi(x_n)$. Montrer que la suite (x_n) converge linéairement à taux au plus α vers x^*.
c/ On suppose que $\Phi'(x^*) = 0$. Montrer que si x_0 est suffisamment proche de x^*, alors la suite (x_n) converge vers x^* à l'ordre au moins 2.

1.5 Efficacité d'un algorithme itératif

En pratique, l'opération qui prend le plus de temps à calculer numériquement est l'évaluation de F (ou f) en un point $\mathbf{x} \in \mathbb{R}^d$. Cette opération peut prendre quelques microsecondes (si F a une formule explicite par exemple), ou bien plusieurs minutes/heures (si $F(\mathbf{x})$ est elle même la solution d'un problème complexe). Ainsi, pour comparer l'efficacité de deux algorithmes, on préfère comparer leur *taux effectif* (ici définit pour des convergences linéaires).

Définition 1.10 : Taux effectif de convergence linéaire

Soit $\mathcal{A}$ un algorithme itératif ayant une structure similaire au Code 1.1. On suppose que $\mathcal{A}$ génère une suite $(\mathbf{x}_n)$ telle que $(\mathbf{x}_n)$ converge linéairement vers $\mathbf{x}^*$ à taux $\alpha \in (0, 1)$. On suppose de plus que l'algorithme appelle k fois la fonction F à chaque itération de sa boucle principale. La *taux effectif* de l'algorithme est $\tau(\mathcal{A}) := \alpha^{1/k}$.

Par exemple, si $\mathcal{A}$ est un algorithme qui génère une suite $(\mathbf{x}_n)$ de type $\mathbf{x}_{n+1} = \Phi(\mathbf{x}_n)$, où Φ est une fonction qui appelle k fois la fonction F, et si $(\mathbf{x}_n)$ converge linéairement vers $\mathbf{x}^*$ à taux $\alpha \in (0, 1)$, alors la vitesse effective de $\mathcal{A}$ est $\tau(\mathcal{A}) = \alpha^{1/k}$.

Soit maintenant $\mathcal{A}'$ l'algorithme qui génère la suite $(\mathbf{y}_n)$ avec $\mathbf{y}_{n+1} = \Phi(\Phi(\mathbf{y}_n))$, alors, par une récurrence immédiate, on a $\mathbf{y}_n = \mathbf{x}_{2n}$, et donc

$$\|\mathbf{y}_n - \mathbf{x}^*\| = \|\mathbf{x}_{2n} - \mathbf{x}^*\| \leq C\alpha^{2n} \leq C(\alpha^2)^n,$$

et on en déduit que la suite $(\mathbf{y}_n)$ converge vers $\mathbf{x}^*$ à taux α^2. Comme $\alpha^2 < \alpha$, l'algorithme $\mathcal{A}'$ génère une suite qui converge plus rapidement (au sens des suites) vers $\mathbf{x}^*$ que la suite $(\mathbf{x}_n)$ générée par l'algorithme $\mathcal{A}$. Cependant, le taux effectif des deux algorithmes est la même, car $\mathcal{A}'$ évalue $2k$ fois la fonction F par itération, et la vitesse effective est $\tau(\mathcal{A}') = (\alpha^2)^{1/(2k)} = \alpha^{1/k} = \tau(\mathcal{A})$.

Première partie

Optimisation sans contraintes

CHAPITRE 2

MÉTHODES EN UNE DIMENSION

Dans ce chapitre, nous étudions les méthodes existantes en une dimension. On veut résoudre des problèmes de type

$$\text{argmin}\{F(x),\ x \in \mathbb{R}\} \quad \text{ou} \quad f(x) = 0,$$

où F et f sont des fonctions de $\mathbb{R}$ dans $\mathbb{R}$. La variable x vit dans $\mathbb{R}$, qui est un espace facile à explorer : il est plus facile de trouver une gare sur des rails ($\approx \mathbb{R}$), qu'un point d'eau dans le désert ($\approx \mathbb{R}^2$) ou qu'une planète dans l'espace ($\approx \mathbb{R}^3$). Cela permet d'utiliser des méthodes à la fois naïves et efficaces.

2.1 Méthodes par dichotomie

2.1.1 Résoudre $f(x) = 0$ par dichotomie

Commençons par la méthode la plus naturelle, dite de *dichotomie*[1]. Soit $f : \mathbb{R} \to \mathbb{R}$ une fonction continue telle que $f(a) < 0$ et $f(b) > 0$. D'après le théorème des valeurs intermédiaires, il existe $x^* \in (a, b)$ tel que $f(x^*) = 0$. Si de plus f est strictement croissante, alors x^* est unique, f est strictement négative sur (a, x^*), et strictement positive sur (x^*, b). Cela suggère de définir les suites suivantes :

$$\textbf{(Initialisation)} \qquad (x_0^-, x_0^+) = (a, b), \quad x_0 := \frac{x_0^+ + x_0^-}{2}, \tag{2.1}$$

puis

$$(x_{n+1}^-, x_{n+1}^+) = \begin{cases} (x_n^-, x_n) & \text{si } f(x_n) \geq 0, \\ (x_n, x_n^+) & \text{si } f(x_n) < 0. \end{cases} \qquad \text{et} \quad x_{n+1} := \frac{x_{n+1}^+ + x_{n+1}^-}{2}. \tag{2.2}$$

Lemme 2.1 : Convergence de la dichotomie

Soit f continue et strictement croissante sur (a, b) avec $f(a) < 0 < f(b)$, soit x^* l'unique solution de l'équation $f(x) = 0$, et soit (x_n) la suite définie par (2.1)-(2.2). Alors (x_n) converge linéairement vers x^* à taux $\alpha = \frac{1}{2}$.

Démonstration. Soit $I_n := [x_n^-, x_n^+]$. On a $I_{n+1} \subset I_n$, et (on note $l(I)$ désigne la longueur de l'intervalle I)

$$l(I_{n+1}) = \frac{1}{2} l(I_n) = \cdots = \frac{1}{2^{n+1}} l(I_0) = \frac{1}{2^{n+1}} |b - a|.$$

1. De $\delta\iota\chi\alpha$ (en deux) et $\tau\epsilon\mu\nu\omega$ (couper)

D'après le théorème des segments emboités, on en déduit que les suites (x_n^-), (x_n^+), et (x_n) convergent vers $x^* := \bigcap_{n=1}^{\infty} I_n$. De plus, comme on a toujours $f(x_n^-) < 0$ et $f(x_n^+) \geq 0$, on obtient à la limite, en utilisant la continuité de f, $f(x^*) \leq 0$ et $f(x^*) \geq 0$, donc $f(x^*) = 0$.

Pour tout $n \in \mathbb{N}$, on a $x^* \in I_n$ et $x_n \in I_n$. On en déduit que

$$|x^* - x_n| \leq l(I_n) \leq \frac{1}{2^n}|b-a|,$$

donc la suite (x_n) converge linéairement vers x^* à taux au plus $\frac{1}{2}$. Montrons que ce taux est optimal, et supposons par l'absurde qu'il existe $\alpha < \frac{1}{2}$ et $C \in \mathbb{R}^+$ tel que pour tout $n \in \mathbb{N}$, on a $|x^* - x_n| \leq 2C\alpha^n$. En particulier, on a,

$$|x_{n+1} - x_n| \leq |x_{n+1} - x^*| + |x^* - x_n| \leq 2C\alpha^n.$$

Par ailleurs, on a aussi

$$|x_{n+1} - x_n| = \frac{1}{2}|x_n - x_{n-1}| = \cdots = \frac{1}{2^n}|x_1 - x_0|.$$

En combinant les deux, on obtient que $2^{-n}|x_1 - x_0| \leq C\alpha^n$, ou encore $0 < (2C)^{-1}|x_1 - x_0| \leq (2\alpha)^n$. Par hypothèse, on a $2\alpha < 1$, et le membre de droite tend vers 0, lorsque n tend vers l'infini, ce qui est impossible (le terme de gauche est strictement positif et ne dépend pas de n). □

Ceci suggère d'utiliser l'algorithme de dichotomie suivant.

Code 2.1 – Algorithme de dichotomie

```
def dichotomie(f, a, b, tol=1e-6, Niter=1000):
    assert(f(a) < 0 and f(b) > 0), "Erreur, on doit avoir f(a) < 0 et f(b) > 0"
    assert(a < b), "Erreur, on doit avoir a < b."

    xmoins, xplus, xn = a, b, (a+b)/2 #initialisation
    L = []

    for n in range(Niter):      #boucle principale
        if xplus-xmoins < 2*tol: #critere d'arret
            return xn,L
        L.append(xn)
        if f(xn) < 0:         # si f(x) est negatif,
            xmoins = xn       # remplacer xmoins par xn
        else:                 # sinon,
            xplus = xn        # remplacer xplus par xn
        xn = (xmoins + xplus)/2
    print("Erreur, l'algorithme n'a pas convergé après ", Niter, " itérations.")
```

Exercice 2.2

a/ Comment vérifier numériquement que l'algorithme renvoie une solution numérique de $f(x) = 0$?
b/ Calculer la solution dans l'intervalle $[1, 2]$ de $x^5 - 3x + 1 = 0$ à 10^{-8} près.

Pour vérifier la convergence linéaire, on utilise le Code 1.2 (voir Figure 2.1). On observe une courbe proche d'une droite (ce qui illustre *a posteriori* que la convergence est linéaire), et on trouve un taux de convergence de $0.504 \approx \frac{1}{2}$, ce qui est en accord avec le Lemme 2.1.

2.1.2 Trouver $\operatorname{argmin} F$ par la méthode de la "section dorée"

On s'intéresse maintenant au cas où on veut résoudre un problème de type

$$x^* := \operatorname{argmin} \{F(x),\ x \in [a, b]\},$$

où F est une fonction continue sur le compact $[a, b]$ (donc le minimum existe).

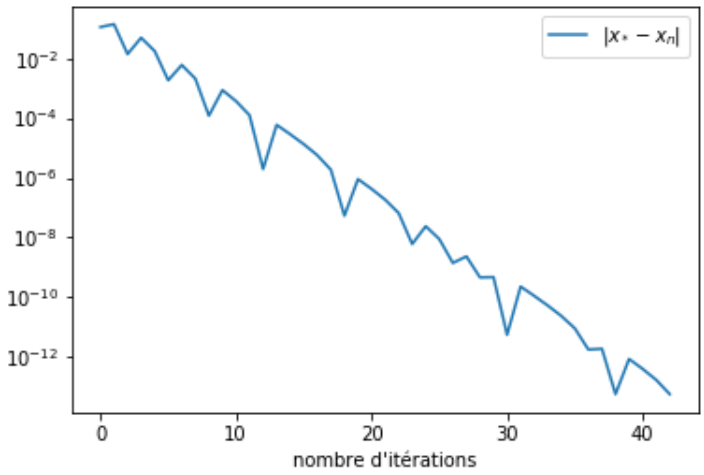

Figure 2.1 – La convergence linéaire de l'algorithme de dichotomie.

Définition 2.3 : Fonction unimodale

Soit F continue sur $[a,b]$. On dit que F est **unimodale** sur $[a,b]$ s'il existe $x^* \in (a,b)$ tel que F est strictement décroissante sur $[a,x^*]$ et strictement croissante sur $[x^*,b]$.

Si F est unimodale et dérivable, sa dérivée $f = F'$ est négative sur $[a,x^*]$, et positive sur $[x^*,b]$. Ainsi, *si on a accès à* F', on pourrait utiliser l'algorithme de dichotomie de la Section 2.1.1 et résoudre $F' = 0$. Malheureusement, la dérivée peut ne pas exister, ou n'est pas forcément une quantité calculable (cf. Section 1.2). Nous présentons ici la méthode dite de la *section dorée*, qui permet de contourner ce problème.

On dit qu'un triplet $a^* < c^* < b^* \in [a,b]$ est **admissible** si $F(c^*) \leq \min\{F(a^*), F(b^*)\}$.

Exercice 2.4

Soit F unimodale sur $[a,b]$ et x^* son minimiseur.
a/ Montrer que si (a^*,c^*,b^*) est admissible, alors $x^* \in [a^*,b^*]$.
b/ Montrer que pour tout $d^* \in (c^*,b^*)$ alors soit (a^*,c^*,d^*), soit (c^*,d^*,b^*) est admissible.

L'idée de l'algorithme est de construire une suite (a_n,c_n,b_n) de triplets admissibles, tel que les intervalles $[a_n,b_n]$ soient emboités avec $(b_n - a_n) \to 0$. On en déduira que les suites (a_n) et (b_n) convergent vers x^* avec le théorème des segments emboités. Pour construire une telle suite de triplets admissibles, on utilise l'Exercice 2.4. Plus exactement, notre algorithme travaillera avec des quadruplets (a_n,c_n,d_n,b_n) tel que l'un des deux triplets (a_n,c_n,b_n) ou (a_n,d_n,b_n) soit admissible, et on posera

$$(a_{n+1},c_{n+1},d_{n+1},b_{n+1}) = \begin{cases} (a_n,X,c_n,d_n) & \text{si} \quad (a_n,c_n,d_n) \text{ est admissible,} \\ (c_n,d_n,X,b_n) & \text{si} \quad (c_n,d_n,b_n) \text{ est admissible,} \end{cases}$$

où X est un nombre à choisir. Dans la méthode de la section dorée, on choisit X de telle sorte qu'il existe $\alpha \in (1/2,1)$ tel que, pour tout $n \in \mathbb{N}$, on a la relation de barycentres

$$c_n = \alpha a_n + (1-\alpha) b_n \quad \text{et} \quad d_n = (1-\alpha) a_n + \alpha b_n. \tag{2.3}$$

Cela implique en particulier qu'on doit poser

$$X = \begin{cases} \alpha a_n + (1-\alpha) d_n & \text{si} \quad (a_n,c_n,d_n) \text{ est admissible,} \\ (1-\alpha) c_n + \alpha b_n & \text{si} \quad (c_n,d_n,b_n) \text{ est admissible.} \end{cases} \tag{2.4}$$

Il n'est pas évident que cette définition suffise pour vérifier (2.3). Par exemple, si (a_n, c_n, d_n) est admissible, alors la relation précédente assure que $c_{n+1} = X$ vérifie la bonne relation de barycentre $c_{n+1} = \alpha a_{n+1} + (1-\alpha) b_{n+1}$, mais on ne sait *a priori* que d_{n+1} vérifie l'autre relation. Il se trouve que lorsque α est l'inverse du nombre d'or [2], c'est le cas, comme le montre le résultat suivant.

Lemme 2.5 : Convergence de la méthode de section dorée

Si

$$\alpha = \frac{\sqrt{5}-1}{2} \approx 0.618 \quad \left(= \frac{1}{\varphi} \quad \text{où} \quad \varphi := \frac{1+\sqrt{5}}{2} \quad \text{est le nombre d'or} \right), \tag{2.5}$$

alors la relation (2.3) est satisfaite pour tout $n \in \mathbb{N}$.

Démonstration. Supposons par récurrence que (2.3) soit satisfaite à l'étape n. On suppose que le triplet (a_n, c_n, d_n) est admissible (la démonstration est la même dans l'autre cas). Dans ce cas, on a $c_{n+1} = X = \alpha a_n + (1-\alpha) d_n = \alpha a_{n+1} + (1-\alpha) b_{n+1}$, et la première relation de (2.3) est vérifiée au rang $n+1$, par construction. Pour l'autre relation, on a

$$\begin{aligned} d_{n+1} = c_n = \alpha a_n + (1-\alpha) b_n = \alpha a_n + (1-\alpha) \left[\frac{1}{\alpha} (d_n - (1-\alpha) a_n) \right] \quad \text{(avec la 2ème relation de (2.3))} \\ = \frac{2\alpha - 1}{\alpha} a_n + \frac{1-\alpha}{\alpha} d_n = \frac{2\alpha-1}{\alpha} a_{n+1} + \frac{1-\alpha}{\alpha} b_{n+1}. \end{aligned}$$

C'est de la forme $d_{n+1} = (1-\alpha) a_{n+1} + \alpha b_{n+1}$ si et seulement si $(1-\alpha)/\alpha = \alpha$, ou encore si $\alpha^2 = 1-\alpha$, donc la seule solution positive est (2.5). □

Exercice 2.6

a/ Montrer qu'on a $b_{n+1} - a_{n+1} = \alpha(b_n - a_n)$.
b/ En déduire que la suite (x_n) avec $x_n := \frac{1}{2}(b_n + a_n)$ converge vers un certain x^*, et que la convergence est linéaire à taux au plus α.
c*/ Montrer que le taux de convergence est exactement α.

Cela donne l'algorithme suivant pour la section dorée.

Code 2.2 – Algorithme de la section dorée

```
def sectionDoree(F, a, b, tol=1e-6, Niter=1000):
    assert(a < b), "Erreur, on doit avoir a < b"

    #Initialisation
    alpha = (sqrt(5)-1)/2
    an, bn, cn ,dn = a, b, alpha*a + (1 - alpha)*b, (1-alpha)*a + alpha*b
    Fan, Fbn, Fcn, Fdn = F(an), F(bn), F(cn), F(dn)
    L = []

    #On suppose que le premier quadruplet contient un triplet admissible
    assert (Fcn < Fan and Fcn < Fbn) or (Fdn < Fan and Fdn < Fbn), ...
    "Erreur, les premiers triplets ne sont pas admissibles"

    #Boucle de la section dorée
    for n in range(Niter):
        if bn-an < 2*tol: #la précision est atteinte
            return (bn+an)/2, L
        L.append((bn+an)/2) #on ajoute le point milieu
        if (Fcn < Fan and Fcn < Fdn): #si c'est (a, c, d) qui est admissible
            an, bn, cn, dn = an, dn, alpha*an + (1-alpha)*dn, cn
```

2. d'où le nom de la méthode *section dorée*.

```
            Fan, Fbn, Fcn, Fdn = Fan, Fdn, F(cn), Fcn
        else: #sinon c'est (c, d, b) qui est admissible
            an, bn, cn ,dn = cn, bn, dn, (1-alpha)*cn + alpha*bn
            Fan, Fbn, Fcn, Fdn = Fcn, Fbn, Fdn, F(dn)
    print("l'algorithme n'a pas convergé après ", Niter, "itérations")
```

Exercice 2.7

a/ Combien d'appels à la fonction F utilise cette implémentation de la section dorée ?
b/ Vérifier graphiquement que la fonction $F(x) := -\exp(\arctan(x) - \cos(5x))$ est unimodale sur l'intervalle $[0, 1]$, et calculer numériquement son minimum sur cet intervalle.

Exercice 2.8

Écrire un algorithme `initialisationSectionDoree(F, a, b)` qui renvoie $[a', b']$ tel que $[a', b'] \subset [a, b]$ et tel que la deuxième assertion de `sectionDoree` soit satisfaite sur $[a', b']$. On pourra utiliser une méthode de dichotomie.

De nouveau, on peut tracer la vitesse de convergence avec le Code 1.2. On trouve un taux de convergence numérique de 0.637, qui est proche de $\alpha \approx 0.618$, comme attendu.

La convergence est en $\alpha^n \approx 0.618^n$. C'est donc une convergence plus lente que la méthode de dichotomie appliquée à $f = F'$, qui converge en 0.5^n. Cependant, elle ne nécessite pas de calculer la dérivée de F. On utilisera donc la méthode de la section dorée lorsqu'on veut éviter de calculer la dérivée F'.

Exercice 2.9

a/ Calculer la vitesse effective de l'algorithme de dichotomie décrit par le Code 2.1.
b/ *Idem* pour l'algorithme de la section dorée décrit par le Code 2.2.
c/ *Idem* pour l'algorithme de dichotomie, lorsqu'il est appliqué à

$$f(x) := \frac{F(x+\varepsilon) - F(x-\varepsilon)}{2\varepsilon} \quad \text{(dichotomie avec différences finies centrées).}$$

d/ *Idem* pour l'algorithme de *trichotomie*, où les intervalles sont coupés en 3 à chaque étape ?

2.2 Résoudre $f(x) = 0$ avec des algorithmes de type Newton

L'algorithme de Newton a déjà été vu en L2, et nous y reviendrons plus en détails pour les fonctions à plusieurs variables. Nous rappelons ici les points principaux de cette méthode dans le cas unidimensionnel, et présentons quelques variantes.

Les algorithmes de type Newton résolvent des problèmes de type $f(x) = 0$ (problème de résolution, pas d'optimisation). En particulier, nous verrons dans le cas où $f = \nabla F$ que l'algorithme de Newton peut trouver des minima/maxima, mais aussi des points selles.

2.2.1 L'algorithme de Newton

L'idée de cet algorithme est qu'on peut approcher $f(x)$ pour $x \sim x_0$ par la fonction *linéaire*

$$f(x) \approx \widetilde{f_{x_0}}(x) := f(x_0) + f'(x_0) \cdot (x - x_0).$$

De plus, la solution de $\widetilde{f_{x_0}}(x) = 0$ a pour solution (évidente) $x_0^* := x_0 - [f'(x_0)]^{-1} f(x_0)$. On pose donc naturellement (voir Figure 2.2).

$$x_{n+1} := x_n - \frac{f(x_n)}{f'(x_n)}. \tag{2.6}$$

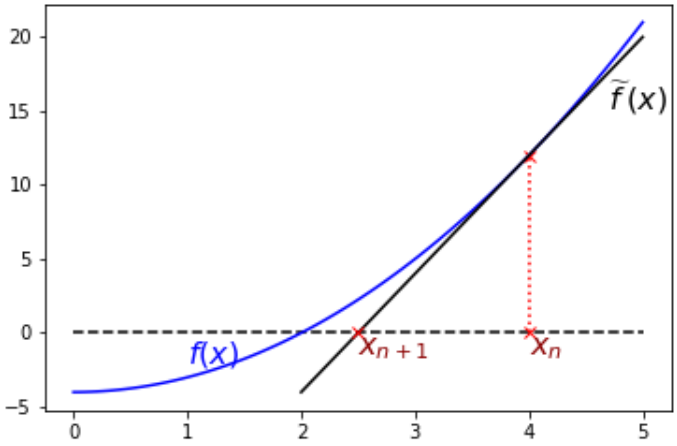

Figure 2.2 – Une itération de l'algorithme de Newton

L'algorithme de Newton nécessite la connaissance de la fonction f'. En particulier, si on veut utiliser l'algorithme de Newton pour résoudre un problème d'optimisation ($f = F'$), on doit connaître la dérivée seconde F'' de F. En pratique, on utilisera soit des formules explicites (lorsque celles-ci sont calculables), soit des différences finies pour approcher ces quantités (cf. Section 1.2).

Enfin, puisqu'on veut résoudre l'équation $f(x) = 0$, il est naturel d'utiliser le critère d'arrêt $|f(x_n)| < \text{tol}$. Cela donne l'algorithme suivant.

Code 2.3 – Algorithme de Newton en 1d

```
def newton(f, df, x0, tol=1e-6, Niter=1000):
    xn, L = x0, [] # Initialisation
    for n in range(Niter):
        fxn = f(xn)
        if abs(fxn) < tol:
            return xn, L
        L.append(xn)
        xn = xn - fxn/df(xn)
    print("Erreur, l'algorithme n'a pas convergé après ", Niter, "itérations")
```

Exercice 2.10

a/ En utilisant $f = \sin$, calculer π. Combien d'itérations faut-il si on commence avec $x_0 = 3$?
b/ Ecrire une fonction `puissanceInverse(X,n)` qui renvoie $X^{1/n}$ en résolvant l'équation $x^n = X$.

On observe numériquement une convergence superlinéaire (voir Figure 2.3). De plus, en calculant par exemple `puissanceInverse(10000, 4)` (dont la solution est $x^* = 10$) avec $x_0 = 11$, on obtient la suite suivante

$$L = 11, 10.128287002253945, 10.002416848739019, 10.000000875820872, 10.000000000000115],$$

dont le nombre de bonnes décimales semble doubler ou tripler à chaque itération.

Lemme 2.11 : Vitesse de convergence de la méthode de Newton

Soit $f : \mathbb{R} \to \mathbb{R}$ une fonction de classe C^3, et $x^* \in \mathbb{R}$ telle que $f(x^*) = 0$ et $f'(x^*) > 0$. Alors il existe $\varepsilon > 0$ tel que pour tout $x_0 \in \mathcal{B}(x^*, \varepsilon)$, la suite (x_n) définie par (2.6) converge quadratiquement vers x^*.

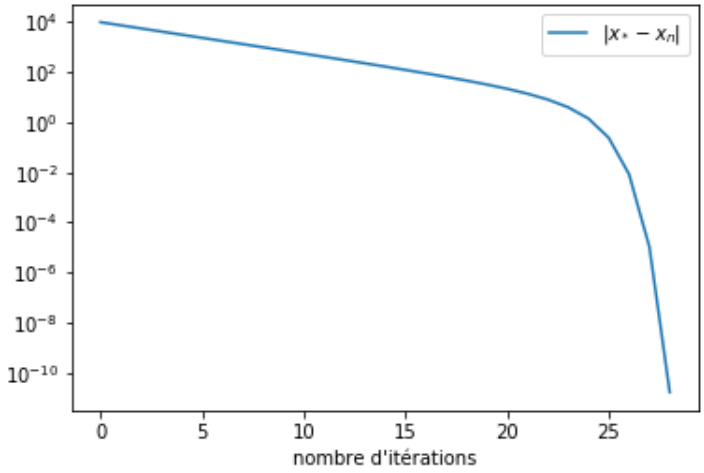

Figure 2.3 – La convergence super-linéaire de l'algorithme de Newton

La preuve, qui est le sujet de l'exercice suivant, est laissée au lecteur.

Exercice 2.12

Soit $f : \mathbb{R} \to \mathbb{R}$ de classe C^3 avec $f(x^*) = 0$ et $f'(x^*) > 0$.
a/ Montrer qu'il existe $\varepsilon > 0$ tel que pour tout $x \in (x^* - \varepsilon, x^* + \varepsilon)$, on a $f'(x) > 0$.
b/ Soit $\Phi : x \mapsto x - [f'(x)]^{-1} f(x)$. Montrer que pour tout $x \in (x^* - \varepsilon, x^* + \varepsilon)$, $\Phi(x) = x$ ssi $x = x^*$.
c/ Montrer que la suite (x_n) définie par (2.6) vérifie $x_{n+1} = \Phi(x_n)$. En utilisant l'Exercice 1.9, montrer que la suite (x_n) converge quadratiquement vers x^*.

2.2.2 Méthode de la sécante

Le principal défaut de la méthode de Newton est d'utiliser la dérivée de f. Pour éviter de calculer f', on peut utiliser la méthode de la sécante (voir Figure 2.4).

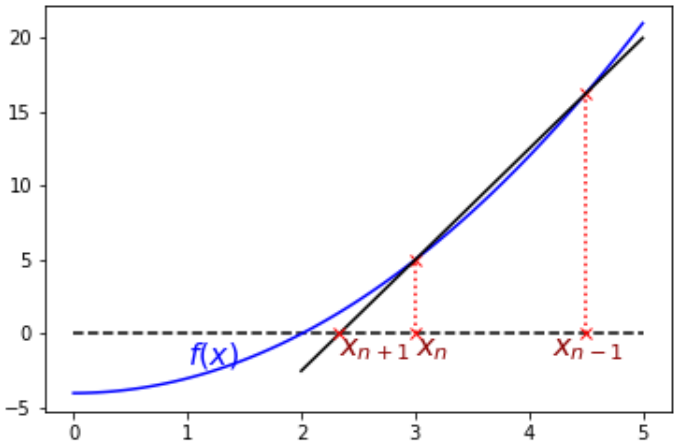

Figure 2.4 – Une itération de l'algorithme de la sécante

Exercice 2.13

Montrer d'après la Figure 2.4 qu'on a

$$x_{n+1} := x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} f(x_n). \tag{2.7}$$

a/ Écrire un programme `secante(f, x0, x1)` qui implémente la méthode de la sécante.
b/ En utilisant $f = \sin$, $x_0 = 3$ et $x_1 = 4$, calculer π.

Un code qui n'utilise qu'un seul appel à la fonction f par itération est le suivant :

Code 2.4 – Algorithme de la sécante en 1d

```
def secante(f, x0, x1, tol=1e-6, Niter=1000):
    xn, xnm1, L = x1, x0, [x0] # Initialisation
    fxn, fxnm1 = f(xn), f(xnm1)
    for n in range(Niter):
        if abs(fxn) < tol:
            return xn, L
        L.append(xn)
        xn, xnm1 = xn - (xn - xnm1)/(fxn - fxnm1)*fxn, xn # double affectation ! ...
        fxn, fxnm1 = f(xn), fxn # ... pour éviter des variables temporaires
    print("Erreur, l'algorithme n'a pas convergé après ", Niter, "itérations")
```

En calculant 10 comme solution de $x^4 = 10000$ avec $x_0 = 12$ et $x_1 = 11$, on obtient la suite

$$L = [12, 11, 10.23855619360, 10.03237133166, 10.00113262871, 10.00000548437, 10.00000000093].$$

De nouveau, on observe une convergence super-linéaire. En fait, on peut calculer l'ordre de convergence de cette méthode.

Lemme 2.14 : Vitesse de convergence de la méthode de la sécante

Si $f : \mathbb{R} \to \mathbb{R}$ est une fonction de classe C^3 avec $f(x^*) = 0$ et $f'(x^*) \neq 0$, et si $x_0, x_1 \in \mathbb{R}$ sont suffisamment proches de x^*, alors la suite (x_n) définie par (2.7) converge vers x^* à l'ordre au moins $\varphi = \frac{1}{2}(1 + \sqrt{5}) \approx 1.618$ (le nombre d'or).

Démonstration. Nous ne donnons la preuve que dans le cas où f est une fonction quadratique. Cela évite des complications qui sont inutiles à la compréhension de la preuve. On pose $e_n := x_n - x^*$. Le développement de Taylor d'une fonction f quadratique donne

$$f(x_n) = f'(x^*)e_n + \frac{1}{2} f''(x^*)e_n^2.$$

(dans le cas général, il faudrait rajouter le reste intégrale, ce qui alourdit la preuve). En remplaçant dans (2.7), et en remarquant que $x_{n+1} - x_n = e_{n+1} - e_n$, on obtient

$$e_{n+1} - e_n = -\frac{(e_n - e_{n-1})\left(f'(x^*)e_n + \frac{1}{2}f''(x^*)e_n^2\right)}{f'(x^*)(e_n - e_{n-1}) + \frac{1}{2}f''(x^*)(e_n^2 - e_{n-1}^2)} = -e_n \left(\frac{f'(x^*) + \frac{1}{2}f''(x^*)e_n}{f'(x^*) + \frac{1}{2}f''(x^*)(e_n + e_{n-1})} \right),$$

ou encore

$$e_{n+1} = e_n \tau_n \quad \text{avec} \quad \tau_n := 1 - \frac{f'(x^*) + \frac{1}{2}f''(x^*)e_n}{f'(x^*) + \frac{1}{2}f''(x^*)(e_n + e_{n-1})} = \frac{\frac{1}{2}f''(x^*)e_{n-1}}{f'(x^*) + \frac{1}{2}f''(x^*)(e_n + e_{n-1})}.$$

On a $f'(x^*) \neq 0$, donc la fonction $T(x, y) := \dfrac{\frac{1}{2}f''(x^*)y}{f'(x^*) + \frac{1}{2}f''(x^*)(x + y)}$ est continue en $(0, 0)$ avec $T(0, 0) = 0$. Il existe $\varepsilon > 0$ tel que pour tout $x, y \in \mathcal{B}(0, \varepsilon)$, on a $|T(x, y)| < \frac{1}{2}$. Ainsi, si e_0 et e_1 sont dans

$\mathcal{B}(x^*,\varepsilon)$, on a $|\tau_1| = |T(e_1,e_0)| < \frac{1}{2}$, et on en déduit que $e_2 \in \mathcal{B}(0,\varepsilon)$. Par une récurrence immédiate, on a toujours $e_n \in \mathcal{B}(0,\varepsilon)$ et $|\tau_n| < \frac{1}{2}$. Par ailleurs, comme $e_{n+1} = e_n\tau_n$, on a $|e_{n+1}| = |e_n|\,|\tau_n| \le \frac{1}{2}|e_n|$, et la suite $|e_n|$ est décroissante, et converge vers 0. En particulier, on a

$$\frac{|e_{n+1}|}{|e_n|\,|e_{n-1}|} = \left|\frac{\frac{1}{2}f''(x^*)}{f'(x^*)+\frac{1}{2}f''(x^*)(e_n+e_{n-1})}\right| \xrightarrow[n\to\infty]{} C := \left|\frac{f''(x^*)}{2f'(x^*)}\right|.$$

On note $u_n = -\log(|e_n|)$. Puisque $(e_n)\to 0$, la suite (u_n) tend vers $+\infty$, et plus (u_n) diverge vite vers l'infini, plus la suite (e_n) converge vite vers 0. En passant au log dans la ligne précédente, on obtient

$$u_{n+1} - u_n - u_{n-1} \xrightarrow[n\to\infty]{} \log(C).$$

La suite $(u_{n+1}-u_n-u_{n-1})$ converge, donc est bornée. Ainsi, si $A \ge \sup_n(u_{n+1}-u_n-u_{n-1})$, la suite $v_n := u_n - A$ vérifie $v_{n+1}-v_n-v_{n-1} = u_{n+1}-u_n-u_{n-1}+A \ge 0$, et donc

$$v_{n+1} \ge v_n + v_{n-1}.$$

Soit $\varphi := \frac{1}{2}(1+\sqrt{5})$ le nombre d'or. Le nombre φ vérifie $\varphi > 1$ et $\varphi^2 = 1+\varphi$. La suite (v_n) diverge vers $+\infty$, donc pour n_0 assez grand, on a $v_{n_0} \ge 1$ et $v_{n_0+1} \ge \varphi$. Par une récurrence immédiate, on en déduit que $v_{n_0+n} \ge \varphi^n$. Cela montre que $u_{n_0+n} \ge \varphi^n + A$, et enfin $|e_{n_0+n}| \le \mathrm{e}^{-A}\alpha^{\varphi^{n_0+n}}$ avec $\alpha := \mathrm{e}^{-\varphi^{n_0}} < 1$, ce qui conclut la preuve. □

La méthode de la sécante est **la meilleure méthode unidimensionelle** ! Avec 1 appel à la fonction par itération et une convergence à l'ordre $\Phi \approx 1.618$, elle est meilleure que la méthode de Newton, qui converge à l'ordre 2, mais doit faire 2 appels à la fonction (1 pour f, et 1 pour f'). La méthode de Newton a donc un ordre effectif de convergence de $\sqrt{2} \approx 1.414$, qui est plus petit que Φ.

Exercice 2.15

Soit $f(x) := ax^2$, et soit (x_n) la suite définie par (2.7) avec $x_0 = 1$ et $x_1 = \frac{1}{2}$.
a/ Montrer que la suite x_n vérifie $x_{n+1}^{-1} = x_n^{-1} + x_{n-1}^{-1}$.
b/ Montrer que $x_n = 1/F_{n+1}$, où F_n est la suite de Fibonacci définie par $F_0 = F_1 = 1$ et $F_{n+1} = F_n + F_{n-1}$.
c/ En déduire que (x_n) converge linéairement vers 0 à taux $\alpha := \frac{1}{\varphi}$, où $\varphi = \frac{1}{2}(1+\sqrt{5})$.
d/ Comparer cette vitesse avec celui du Lemme 2.14. Que se passe-t-il ?

Les Figures 2.2-2.4 montrent à quel point les problèmes d'optimisation sont «faciles» en une dimension. On voit par exemple directement sur la Figure 2.2 que la suite (x_n) va être *décroissante*, ce qui prouve directement la convergence. La notion de *croissance/décroissance* n'a malheureusement pas d'équivalent sur $\mathbb{R}^d$.

2.3 Exercices supplémentaires

Exercice 2.16

Soit $F(x) := \sqrt{1+x^2}$ et $f = F'$. On veut résoudre l'équation $f(x) = 0$ avec l'algorithme de Newton.
a/ Montrer que la suite définie par l'algorithme de Newton (2.6) vérifie $x_{n+1} = -x_n^3$.
b/ On suppose que $|x_0| < 1$. Montrer que x_n converge vers $x^* = 0$ à l'ordre 3.
c/ Que se passe-t-il si $x_0 = \pm 1$ et si $|x_0| > 1$?

Exercice 2.17

Soit $\Phi : \mathbb{R} \to \mathbb{R}$ la fonction définie par $\Phi(x) = x + \sin(x)$. On pose $x_0 \in \mathbb{R}$ et $x_{n+1} = \Phi(x_n)$.
a/ Montrer que Φ est une fonction strictement croissante, et calculer ses points fixes.
b/ Soit $k \in \mathbb{N}$, et $x_0 \in (k\pi, (k+1)\pi)$. Montrer que pour tout $n \in \mathbb{N}$, on a $x_n \in (k\pi, (k+1)\pi)$, puis montrer que la suite (x_n) est croissante si k est pair, et décroissante si k est impair.
c/ Montrer que si $x_0 = 3$, alors la suite (x_n) converge vers π.
d/ Montrer que pour tout $x \in \mathbb{R}$, on a

$$|\Phi(x) - \Phi(\pi)| \leq \frac{1}{6} \max |\Phi'''| \cdot |x - \pi|^3.$$

e/ En déduire que si $x_0 = 3$, alors la suite (x_n) converge vers π à l'ordre 3.

CHAPITRE 3

MÉTHODES DE TYPE DESCENTE DE GRADIENT

3.1 Introduction.

Les *méthodes de descente* permettent de trouver des minimiseurs locaux de fonctions à plusieurs variables (problème d'optimisation uniquement). L'idée est de partir d'un point $\mathbf{x}_0$ suffisamment proche [1] du minimiseur $\mathbf{x}^*$ qu'on cherche, et de construire une suite $(\mathbf{x}_n)$ tel que $F(\mathbf{x}_{n+1}) < F(\mathbf{x}_n)$. La suite des $(F(\mathbf{x}_n))$ formera alors une suite décroissante bornée inférieurement, et convergera vers une valeur, qu'on espère être $F(\mathbf{x}^*)$.

On rappelle le développement de Taylor à l'ordre de 2 de F en $\mathbf{x} \in \mathbb{R}^d$, qui affirme que

$$F(\mathbf{x}+\mathbf{h}) = F(\mathbf{x}) + \langle \nabla F(\mathbf{x}), \mathbf{h} \rangle + \frac{1}{2}\mathbf{h}^T \left[H_F(\mathbf{x})\right] \mathbf{h} + o(\mathbf{h}^2), \tag{3.1}$$

où $\nabla F(\mathbf{x}) \in \mathbb{R}^d$ est le gradient de F, et $H_F(\mathbf{x}) \in \mathcal{M}_d(\mathbb{R})$ est la hessienne de F.

Exercice 3.1

Soit $F : \mathbb{R}^d \to \mathbb{R}$ de classe C^2.
a/ Soit $\mathbf{x}^* \in \mathbb{R}^d$ un minimiseur local de F. Montrer que $\nabla F(\mathbf{x}^*) = \mathbf{0}$ et que $H_F(\mathbf{x}^*)$ est une matrice positive.
b/ Réciproquement, soit $\mathbf{x}^* \in \mathbb{R}^d$ tel que $\nabla F(\mathbf{x}^*) = \mathbf{0}$ et $H_F(\mathbf{x}^*)$ est une matrice *définie* positive, alors $\mathbf{x}^*$ est un minimiseur *strict* local.

Dans la suite, comme F est une fonction à plusieurs variables, on prendra l'habitude en PYTHON d'utiliser un `array` pour les éléments $\mathbf{x} \in \mathbb{R}^d$. On aura `x = array([x_1, x_2, ..., x_d])`.

3.1.1 Nombre de variables

Avant de commencer, discutons du nombre de variables d que les fonctions F peuvent avoir, et donnons quelques exemples pratiques de type de problèmes rencontrés.

- $d = 1$. C'est le cas étudié précédemment. Il permet de trouver des solutions numériques de fonctions réelles.
- $d = 2$. Le cas le plus utilisé est celui de la *régression linéaire*, où on approxime un nuage de points par une droite(cf Exercice 3.29). Tous nos exemples se feront avec des fonctions à deux variables, car elles ont l'avantage d'être facilement représentées grâce à leurs courbes de niveau.
- $d \approx 10^2/10^4$. Ce cas arrive lorsqu'on veut optimiser des trajectoires. On peut par exemple optimiser la trajectoire d'un avion pour consommer le moins de carburant. Une trajectoire peut être représentée par une suite de points de l'espace par lequel l'avion passe. On peut aussi penser à l'optimisation de forme : trouver la structure d'un pont la plus solide.

1. Toutes les méthodes sont *locales*. On supposera toujours que $\mathbf{x}_0$ est «proche» de $\mathbf{x}^*$.

— $d \approx 10^4/10^6$. Ce cas apparaît en imagerie, où on veut reconstruire une image à partir de données incomplète (par exemple une échographie). Dans ce cas, le nombre de variables est le nombre de pixels, qui peut facilement atteindre le million.
— $d \approx 10^6/10^8$. Ce cas peut apparaître lors de l'optimisation sur des réseaux (trouver les branchements internet optimaux en énergie à l'échelle d'un pays). Les réseaux de neurones modernes (intelligence artificielle) sont créés en optimisant plusieurs millions de paramètres.

Exercice 3.2

Soit $F : \mathbb{R}^d \to \mathbb{R}$. On veut calculer $\nabla F(\mathbf{x})$ par différence finie centrée. Combien d'appels à F doit-on faire ? Même question pour la Hessienne $H_F(\mathbf{x})$.

Exercice 3.3

Écrire une fonction Python `gradientDFC(F, x, eps)` qui calcule le gradient de F en $\mathbf{x}$ par différences finies centrées dans chaque direction. Si `x` est un `array` de taille d, alors `gradientDFC` doit aussi renvoyer un `array` de taille d.

3.1.2 Surfaces de niveau

Soit $F : \mathbb{R}^d \to \mathbb{R}$ une fonction à plusieurs variables. La **surface** [2] **de niveau** $\mathcal{C}_\lambda(F)$ de F de niveau λ est l'ensemble $F^{-1}(\{\lambda\})$, c'est à dire l'ensemble des points $\mathbf{x} \in \mathbb{R}^d$ tel que $F(\mathbf{x}) = \lambda$.

Exercice 3.4

Soit $F : \mathbb{R}^d \to \mathbb{R}$. Montrer que les surfaces de niveau de F forment une partition de $\mathbb{R}^d$, c'est à dire que $\mathcal{C}_F(\lambda) \cap \mathcal{C}_F(\mu) = \emptyset$ si $\lambda \neq \mu$, et que pour tout $\mathbf{x} \in \mathbb{R}^d$, il existe $\lambda \in \mathbb{R}$ tel que $\mathbf{x} \in \mathcal{C}_F(\lambda)$.

Lorsque F est une fonction à deux variables, on peut représenter les courbes de niveau avec la fonction `contour` de Python. Dans l'exemple suivant, on trace les courbes de

$$F((x_1, x_2)) = x_2^2 - \cos(x_1 - x_2^2 - x_2).$$

Code 3.1 – Pour afficher des courbes de niveau de F.

```
1  def F(x):  # x est un array
2      x1, x2 = x[0], x[1] # Les indices de Python commencent à 0.
3      return x2**2 - cos(x1 - x2**2 - x2)
4
5  Nx1, Nx2 = 100, 50 #100 points en x1, 50 en x2
6  xx1, xx2 = linspace(-5, 5, Nx1), linspace(-2, 2, Nx2)
7  #On construit toutes les valeurs sous forme de liste de listes
8  Z = [[F(array([x1,x2])) for x1 in xx1] for x2 in xx2]
9
10 contour(xx1, xx2, Z, 20) # On affiche 20 courbes de niveau de F
11 axis('equal') # Pour avoir la même échelle des deux axes.
```

On obtient la Figure 3.1.

Plus la fonction F atteint des valeurs petites (resp. grandes), plus les courbes de niveau correspondantes sont bleues/violettes (resp. jaunes/rouges). Ainsi, on voit graphiquement sur la Figure 3.1 que le point $(0, 0)$ est un minimum local de F. L'idée de la méthode du gradient est de «voyager» vers les zones où F atteint des petites valeurs.

3.1.3 Algorithme de descente

Dans la suite, pour $F : \mathbb{R}^d \to \mathbb{R}$ une fonction à d variables, $\mathbf{x} \in \mathbb{R}^d$ un point et $\mathbf{h} \in \mathbb{R}^d$ une direction, on pose

$$F_{\mathbf{x},\mathbf{h}}(t) := F(\mathbf{x} + t\mathbf{h}). \tag{3.2}$$

2. On parle de **courbes de niveau** si F est une fonction à deux variables

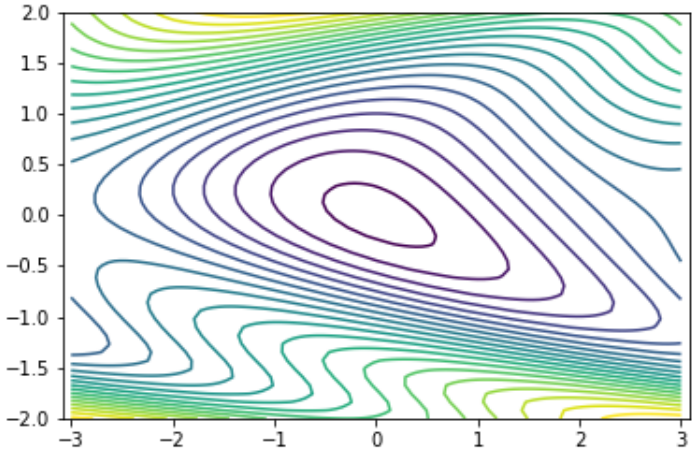

Figure 3.1 – Exemple de courbes de niveau de F.

La fonction $F_{\mathbf{x},\mathbf{h}}$ est une fonction à une seule variable $t \in \mathbb{R}$, qui décrit le comportement de F sur la droite $\mathbf{x} + t\mathbf{h}$ qui passe par $\mathbf{x}$ avec la direction $\mathbf{h}$.

Définition 3.5 : Direction de descente

Soit $F : \mathbb{R}^d \to \mathbb{R}$ une fonction de classe C^2, soit $\mathbf{x} \in \mathbb{R}^d$ et soit $\mathbf{h} \in \mathbb{R}^d \setminus \{\mathbf{0}\}$ vu comme un vecteur. On dit que $\mathbf{h}$ est une *direction de descente* de F en $\mathbf{x}$ si $F'_{\mathbf{x},\mathbf{h}}(0) < 0$.

Cela signifie qu'en étant en $\mathbf{x} \in \mathbb{R}^d$, et en prenant la direction $\mathbf{h}$, la fonction F va commencer par décroitre. Un algorithme de descente fonctionne alors de la façon suivante : on part de $\mathbf{x}_0 \in \mathbb{R}^d$ proche de $\mathbf{x}^*$, puis pour tout $n \in \mathbb{N}^*$,

— on choisit une direction de descente $\mathbf{h}_n \in \mathbb{R}^d$ de F en $\mathbf{x}_n$.
— on choisit un *pas* $t_n > 0$.
— on pose $\mathbf{x}_{n+1} = \mathbf{x}_n + t_n\mathbf{h}_n$.

Si $\mathbf{h}_n$ une direction de descente, et si le pas t_n est suffisamment petit, on aura bien $F(\mathbf{x}_{n+1}) < F(\mathbf{x}_n)$. En revanche, si le pas est trop petit, alors on aura aussi $F(\mathbf{x}_{n+1}) \approx F(\mathbf{x}_n)$, et l'algorithme devra faire beaucoup d'itérations pour trouver $\mathbf{x}^*$. Il faut donc trouver un compromis pour choisir le bon pas. Nous verrons différentes méthodes pour choisir le pas.

En ce qui concerne le choix de la direction de descente, nous utiliserons pour commencer le résultat suivant.

Lemme 3.6 : Meilleure direction de descente locale

Soit $\mathbf{x} \in \mathbb{R}^d$ tel que $\nabla F(\mathbf{x}) \neq \mathbf{0}$. La direction $\mathbf{h} \in \mathbb{R}^d$ est de descente si et seulement si $\langle \nabla F(\mathbf{x}), \mathbf{h} \rangle < 0$. En particulier, $\mathbf{h}_\nabla := -\nabla F(\mathbf{x})/\|\nabla F(\mathbf{x})\|$ est une direction de descente. C'est la *meilleure direction locale* de descente dans le sens où elle est la solution de

$$\operatorname{argmin} \left\{ F'_{\mathbf{x},\mathbf{h}}(0), \quad \mathbf{h} \in \mathbb{R}^d, \ \|\mathbf{h}\| = 1 \right\}. \tag{3.3}$$

On retiendra que l'opposée du gradient (normalisée) est la meilleure direction *locale* de descente.

Démonstration. D'après (3.2) et (3.1) à l'ordre 1, on a

$$F_{\mathbf{x},\mathbf{h}}(t) = F_{\mathbf{x},\mathbf{h}}(0) + t\langle \nabla F(\mathbf{x}), \mathbf{h} \rangle + o(t).$$

On en déduit que $F'_{\mathbf{x},\mathbf{h}}(0) = \langle \nabla F(\mathbf{x}), \mathbf{h} \rangle$, ce qui montre la première partie. Par ailleurs, le minimum de (3.3) existe, car c'est le minimum d'une fonction continue sur la sphère unité de $\mathbb{R}^d$, qui est compacte. Avec l'inégalité de Cauchy-Schwarz, on a $\left|F'_{\mathbf{x},\mathbf{h}}(0)\right| \leq \|\nabla F(\mathbf{x})\| \cdot \|\mathbf{h}\|$, avec égalité si et seulement si $\mathbf{h}$ et $\nabla F(\mathbf{x})$ sont colinéaires. En particulier, les deux candidats possibles pour le minimum de (3.3) sont $\pm \nabla F(\mathbf{x})/\|\nabla F(\mathbf{x})\|$, et on a

$$F'_{\mathbf{x},-\nabla F(\mathbf{x})/\|\nabla F(\mathbf{x})\|}(0) = -\|\nabla F(\mathbf{x})\| < 0 \quad \text{et} \quad F'_{\mathbf{x},\nabla F(\mathbf{x})/\|\nabla F(\mathbf{x})\|}(0) = \|\nabla F(\mathbf{x})\| > 0.$$

Ainsi, le minimum de (3.3) est $-\nabla F(\mathbf{x})/\|\nabla F(\mathbf{x})\| = \mathbf{h}_\nabla$, comme voulu. □

Enfin, on rappelle que le gradient $\nabla F(\mathbf{x})$ est toujours perpendiculaire à la surface de niveau qui passe par $\mathbf{x}$. En effet, soit $\lambda := F(\mathbf{x})$, et $\gamma : (\mathbb{R}, 0) \to (\mathbb{R}^d, \mathbf{x})$ une courbe à valeurs dans $\mathcal{C}_\lambda$, c'est à dire telle que pour tout $t \in \mathbb{R}$, $\gamma(t) \in \mathcal{C}_F(\lambda)$. Alors $F(\gamma(t)) = \lambda$, et en dérivant, on obtient

$$\langle \nabla F(\gamma(t)), \gamma'(t) \rangle = 0, \quad \text{et en } t = 0, \quad \langle \nabla F(\mathbf{x}), \gamma'(0) \rangle = 0.$$

Ceci étant vrai pour toute courbe γ incluse dans $\mathcal{C}_F(\lambda)$, $\nabla F(\mathbf{x})$ est perpendiculaire à $\mathcal{C}_F(\lambda)$.

3.2 Méthode de gradient à pas constant

3.2.1 Algorithme

La méthode de *descente de gradient à pas constant*, consiste, comme son nom indique, à choisir comme direction de descente $\mathbf{h}_n := -\nabla F(\mathbf{x}_n)$, et comme pas de descente $t_n = \tau > 0$ une constante fixe. Autrement dit, on a

$$\boxed{\mathbf{x}_{n+1} = \mathbf{x}_n - \tau \nabla F(\mathbf{x}_n).} \qquad \text{(Itération du gradient à pas constant).} \tag{3.4}$$

Cela donne l'algorithme suivant :

Code 3.2 – Algorithme du gradient à pas constant

```
1 def gradientPasConstant(gradF, x0, tau, tol=1e-6, Niter=1000):
2     xn, L = x0, []
3     for n in range(Niter):
4         gradFxn = gradF(xn)
5         if norm(gradFxn) < tol:
6             return xn, L
7         L.append(xn)
8         xn = xn - tau*gradFxn
9     print("Erreur, l'algorithme n'a pas convergé après ", Niter, "itérations")
```

On remarque que l'algorithme du gradient ne nécessite pas la fonction F, mais uniquement son gradient. Ici, `x0` est implicitement un `array` de taille d, et `gradF` est une fonction qui prend un `array` de taille d, et qui renvoie un `array` de taille d. En particulier, l'opération l.8 est bien définie.

En changeant le paramètre du pas $\tau > 0$, on obtient les Figures 3.2 pour la fonction $F(x, y) := y^2 - \cos(x)$.

On voit que lorsque τ est trop petit (figure de gauche), l'algorithme nécessite beaucoup d'itérations : la suite reste trop longtemps sur place. Lorsque τ est trop grand (figure de droite), la suite $(\mathbf{x}_n)$ oscille à côté de $\mathbf{x}^*$ sans le trouver. Il est donc important de choisir le pas τ adéquatement.

Exercice 3.7

a/ Écrire une fonction `nombreIterations(tau)` qui renvoie le nombre d'itérations que prend l'algorithme du gradient à pas constant pour la fonction $F = y^2 - \cos(x)$ en partant du point $(2, 3/2)$.
b/ Tracer le nombre d'itérations en fonction de $\tau \in [0, 1]$. Qu'observe-t-on ?

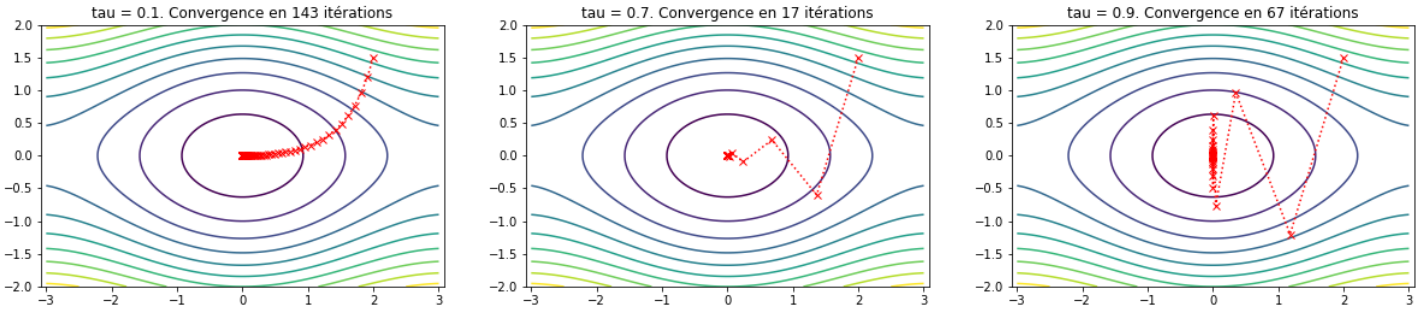

Figure 3.2 – Le gradient à pas constant avec $\tau = 0.1$ (gauche), $\tau = 0.7$ (droite) et $\tau = 0.9$ (gauche).

Exercice 3.8

Soit $F : x \mapsto \frac{1}{2}\alpha x^2 - bx$ avec $\alpha > 0$ et $b \in \mathbb{R}$, et soit (x_n) la suite définie par (3.4) avec $x_0 = 1$.
a/ Calculer x^*, le minimum global de F.
b/ Montrer que (x_n) converge vers x^* si et seulement si $\tau\alpha < 2$. Quelle est la vitesse de convergence dans ce cas ?
c/ Que se passe-t-il dans le cas $\tau = \alpha^{-1}$?

3.2.2 Étude de la convergence pour une fonction quadratique

Dans cette section, nous étudions en détails les propriétés de l'algorithme de gradient à pas constant, dans le cas où F est une fonction quadratique. Il existe deux raisons principales pour lesquelles cette étude est importante.

Pour commencer, d'après (3.1), une fonction F peut toujours être approchée par une fonction quadratique. De plus, si la hessienne $H_F(\mathbf{x}^*)$ est symétrique définie positive (minimum non dégénérée), alors pour tout $\mathbf{x}$ proche de $\mathbf{x}^*$, $H_F(\mathbf{x})$ est aussi symétrique définie positive. Il est donc naturel d'étudier dans un premier temps le cas des fonctions quadratiques de la forme

$$Q(\mathbf{x}) := \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x},$$

où $A \in \mathcal{M}_d(\mathbb{R})$ est une matrice symétrique définie positive, et $\mathbf{b} \in \mathbb{R}^d$ est un vecteur de $\mathbb{R}^d$.

Exercice 3.9

a/ Calculer le gradient de Q.
b/ Montrer que le minimum de Q existe, est unique, et est donnée par $\mathbf{x}^* := A^{-1}\mathbf{b}$.
c/ En déduire qu'on peut résoudre $A\mathbf{x} = \mathbf{b}$ en minimisant Q.

Par ailleurs, d'après l'exercice précédent, on peut résoudre le problème d'algèbre linéaire $A\mathbf{x} = \mathbf{b}$ avec des techniques d'optimisation. Il a été vu en L2 qu'il était possible d'inverser la matrice A avec un pivot de Gauss. Cependant, si $A \in \mathcal{M}_d(\mathbb{R})$, calculer A^{-1} avec le pivot de Gauss demande $O(d^3)$ opérations, et devient rapidement inutilisable si le nombre de variables d devient trop grand (cf Section 3.1.1). L'idée est de calculer directement $\mathbf{x}^*$ comme étant le minimiseur de Q, et d'utiliser des algorithmes itératifs.

Dans la suite, on note $\mathcal{S}_d(\mathbb{R})$ l'ensemble des matrices de $\mathcal{M}_d(\mathbb{R})$ qui sont symétriques réelles, $\mathcal{S}_d^+(\mathbb{R})$ (resp. $\mathcal{S}_d^{++}(\mathbb{R})$) celles qui sont symétriques positives (resp. définies positives). Pour $A, B \in \mathcal{S}_d(\mathbb{R})$, on note $A \geq 0$ si $A \in \mathcal{S}_d^+(\mathbb{R})$, et $A \geq B$ si $A - B \geq 0$, et on note $A > 0$ si $A \in \mathcal{S}_d^{++}(\mathbb{R})$, et $A > B$ si $A - B > 0$. Enfin, pour $\lambda \in \mathbb{R}$, on note $A \geq \lambda$ pour $A \geq \lambda \mathbb{I}_d$. On note $\mathbb{S}^{d-1} = \{\mathbf{x} \in \mathbb{R}^d,\ \|\mathbf{x}\| = 1\}$ la sphère en dimension d, et pour $A \in \mathcal{M}_d(\mathbb{R})$, on note $\|A\|_{\text{op}} := \max\{\|A\mathbf{x}\|, \mathbf{x} \in \mathbb{S}^{d-1}\}$ la norme d'opérateur de A.

Quelques rappels d'algèbre linéaire sont donnés en Appendix A.1. Dans cette section, nous aurons besoin des résultats suivants (voir l'Appendix pour la preuve).

Lemme 3.10 : Rappel d'algèbre linéaire

Soit $A \in \mathcal{S}_d(\mathbb{R})$ et soit $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_d$ ses valeurs propres dans l'ordre croissant. On a

$$\lambda_1 = \min\{\langle \mathbf{x}, A\mathbf{x}\rangle,\ \mathbf{x} \in \mathbb{S}^{d-1}\} \quad \text{et} \quad \lambda_d = \max\{\langle \mathbf{x}, A\mathbf{x}\rangle,\ \mathbf{x} \in \mathbb{S}^{d-1}\} \quad \text{(principe du min/max)}.$$

De plus, on a $\|A\|_{\text{op}} = \max\{|\lambda_1|, |\lambda_d|\}$.

Exercice 3.11

Soit $A \in \mathcal{S}_d^{++}$, et soit $0 < \lambda_1 \leq \lambda_d$ plus petite et plus grande valeur propre de A respectivement.
a/ Montrer que pour tout $\mathbf{x} \in \mathbb{R}^d$, on a $\lambda_1\|\mathbf{x}\|^2 \leq \mathbf{x}^T A\mathbf{x} \leq \lambda_d\|\mathbf{x}\|^2$.
b/ En déduire que $\|\cdot\|_A : \mathbf{x} \mapsto \mathbf{x}^T A\mathbf{x}$ est une norme équivalente à $\|\cdot\|$.

Le résultat principal de cette section est donné dans le lemme suivant.

Lemme 3.12 : Vitesse de convergence du gradient à pas constant

Soit $A \in \mathcal{S}_d^{++}(\mathbb{R})$ et $\mathbf{b} \in \mathbb{R}^d$, et soit $Q(\mathbf{x}) := \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x}$. Soit $0 < \lambda_1 \leq \cdots \leq \lambda_d$ les valeurs propres de A rangées en ordre croissant. Soit enfin la suite $(\mathbf{x}_n)$ définie par $\mathbf{x}_0 \in \mathbb{R}^d$ et

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \tau\nabla Q(\mathbf{x}_n) = \mathbf{x}_n - \tau(A\mathbf{x}_n - \mathbf{b}). \tag{3.5}$$

Alors la suite $(\mathbf{x}_n)$ converge vers $\mathbf{x}^* := A^{-1}\mathbf{b}$ si (et seulement si) $\tau\lambda_d < 2$. La convergence est linéaire, à taux $\max\{|1-\tau\lambda_1|, |1-\tau\lambda_d|\}$. Le pas optimal est $\tau^* = \frac{2}{\lambda_1+\lambda_d}$, et le taux vaut dans ce cas $\frac{\lambda_d-\lambda_1}{\lambda_d+\lambda_1}$.

Démonstration. On pose $\mathbf{r}_n := \mathbf{x}_n - \mathbf{x}^*$ (l'erreur à l'itération n). Comme $\mathbf{x}^* = A^{-1}\mathbf{b}$, on a

$$\mathbf{r}_{n+1} = \mathbf{x}_{n+1} - \mathbf{x}^* = \mathbf{x}_n - \mathbf{x}^* - \tau A(\mathbf{x}_n - \mathbf{x}^*) = (\mathbb{I} - \tau A)(\mathbf{x}_n - \mathbf{x}^*) = (\mathbb{I} - \tau A)\mathbf{r}_n.$$

On a donc $\|\mathbf{r}_{n+1}\| \leq \|\mathbb{I} - \tau A\|_{\text{op}}\|\mathbf{r}_n\|$. La matrice $\mathbb{I} - \tau A$ est symétrique, et ses valeurs propres sont $1 - \tau\lambda_d \leq \cdots \leq 1 - \tau\lambda_1$. Ainsi, d'après le Lemme 3.10, on a

$$\|\mathbf{r}_{n+1}\| \leq (\max\{|1-\tau\lambda_1|, |1-\tau\lambda_d|\})\,\|\mathbf{r}_n\|.$$

On en déduit que $(\mathbf{r}_n)$ converge vers $\mathbf{0}$, et donc $(\mathbf{x}_n)$ converge vers $\mathbf{x}^*$, si et seulement si $-1 < 1 - \tau\lambda_d$ (condition dans le lemme) et si $1 - \tau\lambda_1 < 1$ (ce qui est toujours vrai, car $\tau > 0$ et $\lambda_1 > 0$ par hypothèse).

Le taux optimal est atteint lorsque τ minimise $\tau \mapsto \max\{|1-\tau\lambda_1|, |1-\tau\lambda_d|\}$. Un calcul élémentaire montre que le minimum est atteint lorsque $|1 - \tau\lambda_1| = |1 - \tau\lambda_d|$, ou encore $1 - \tau\lambda_1 = \tau\lambda_d - 1$, soit $\tau = \frac{2}{\lambda_1+\lambda_d}$. □

Le lemme 3.12 montre qu'on peut espérer une convergence rapide lorsque $\lambda_d \approx \lambda_1$, i.e. lorsque les valeurs propres de A sont du même ordre de grandeur. On dit dans ce cas que A est **bien conditionnée**. En affichant les premiers pas de `gradientPasConstant` de la fonction $Q(x_1, x_2) = \frac{1}{2}(\lambda_1 x_1^2 + \lambda_2 x_2^2)$ pour différentes valeurs de (λ_1, λ_2), et en choisissant le pas optimal $\tau = 2/(\lambda_1 + \lambda_2)$, on obtient les Figures 3.3.

On observe comme prévu que la convergence est plus rapide lorsque $\lambda_1 \approx \lambda_2$.

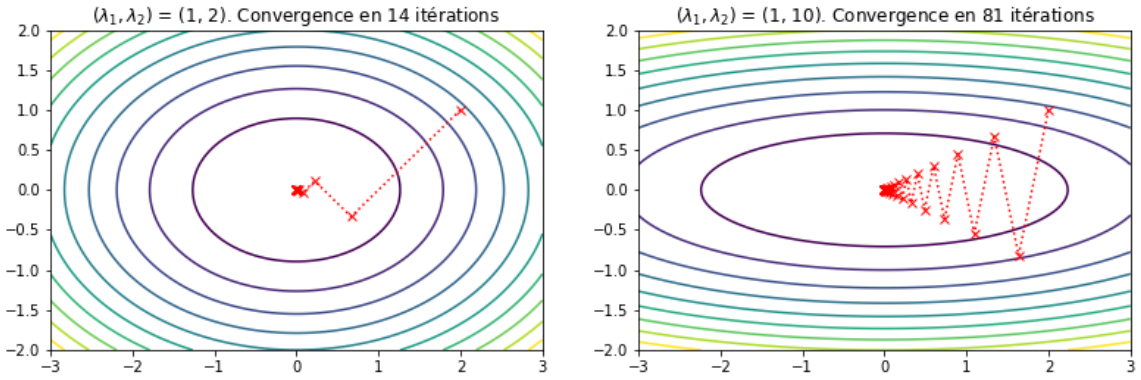

Figure 3.3 – Forme quadratique bien conditionnée (gauche), et mal conditionnée (droite).

3.2.3 Multiplication matrice/vecteur

D'après la formule d'itération (3.5), la matrice A n'intervient que dans le produit $A\mathbf{x}_n$. Il n'est donc pas nécessaire de *garder en mémoire* la matrice A, mais seulement d'avoir une fonction qui prend un vecteur $\mathbf{x} \in \mathbb{R}^d$, et qui renvoie le produit $A\mathbf{x} \in \mathbb{R}^d$.

Exercice 3.13

Comparer les codes Python suivants en terme de mémoire et de temps de calcul (on rappelle qu'en Python, `dot` désigne la multiplication des matrices, alors que `*` est un produit terme à terme -ou produit de Hadamard).

```
1    # Ici, L = [lambda_1, ..., lambda_d]
2    A = diag(L)
3    def multA_1(x): return dot(A,x)
4    def multA_2(x): return array(L)*x
```

Les matrices A provenant de problèmes réels sont généralement *creuses*, c'est à dire que la plupart des coefficients de A sont égaux à 0. Dans ce cas, il est possible à la fois de stocker la matrice A et d'effectuer la multiplication $A\mathbf{x}$ très rapidement, et les algorithmes itératifs constituent un outil puissant pour calculer la solution de $A\mathbf{x} = \mathbf{b}$.

3.2.4 Étude de la convergence dans le cas général

On s'intéresse maintenant aux propriétés de convergence dans le cas général. Dans ce cours, on ne s'intéressera qu'au cas où F est une fonction de classe C^2, et $\mathbf{x}^*$ est un minimiseur local de F *non dégénéré*, c'est à dire que $H_F(\mathbf{x}^*)$ est symétrique *définie* positive. On commence par un lemme utile d'algèbre linéaire.

Lemme 3.14

Pour tout $A, B \in \mathcal{S}_d(\mathbb{R})$, on a

$$|\lambda_1(A) - \lambda_1(B)| \leq \|A - B\|_{\text{op}} \quad \text{et} \quad |\lambda_d(A) - \lambda_d(B)| \leq \|A - B\|_{\text{op}}. \tag{3.6}$$

Démonstration. Soit $\mathbf{x}_B \in \mathbb{S}^{d-1}$ tel que $\lambda_1(B) = \langle \mathbf{x}_B, B\mathbf{x}_B \rangle$. D'après le principe du min/max (Lemme 3.10), on a

$$\lambda_1(A) \leq \langle \mathbf{x}_B, A\mathbf{x}_B \rangle = \langle \mathbf{x}_B, (A - B)\mathbf{x}_B \rangle + \langle \mathbf{x}_B, B\mathbf{x}_B \rangle \leq \|A - B\|_{\text{op}} + \lambda_1(B).$$

En intervertissant le rôle de A et B, on obtient (3.6). □

Autrement dit, les applications $A \mapsto \lambda_1(A)$ et $A \mapsto \lambda_d(A)$ sont 1-Lipschitz, donc continues. On en déduit le lemme suivant.

Lemme 3.15 : Continuité de la Hessienne

Soit $F : \mathbb{R}^d \to \mathbb{R}$ une fonction de classe C^2, et soit $\mathbf{x}^*$ un minimiseur non dégénéré de F. Soit $0 < \lambda_1 \leq \cdots \leq \lambda_d$ les valeurs propres de $H_F(\mathbf{x}^*)$. Alors, pour tout $0 < l < \lambda_1$ et tout $L > \lambda_d$, il existe $\varepsilon > 0$ tel que

$$\forall \mathbf{x} \in \mathcal{B}(\mathbf{x}^*, \varepsilon), \quad l \leq H_F(\mathbf{x}) \leq L.$$

On peut maintenant énoncer le résultat de convergence de la méthode du gradient à pas constant.

Théorème 3.16 : Vitesse de convergence du gradient à pas constant, cas non linéaire

Avec les mêmes notations que le Lemme 3.15, pour tout $0 < \tau < 2/L$ et pour tout $\mathbf{x}_0 \in \mathcal{B}(\mathbf{x}^*, \varepsilon)$, la suite définie par les itérations du gradient à pas constant (3.4) converge linéairement vers $\mathbf{x}^*$, à taux au plus $\max\{|1 - \tau l|, |1 - \tau L|\}$.

Démonstration. D'après la formule de Taylor à l'ordre 1 avec reste intégrale appliquée pour ∇F, et comme $\nabla F(\mathbf{x}^*) = \mathbf{0}$, on a, pour tout $\mathbf{x} \in \mathcal{B}(\mathbf{x}^*, \varepsilon)$,

$$\nabla F(\mathbf{x}) = \nabla F(\mathbf{x}^*) + \int_0^1 \left[H_F(\mathbf{x}^* + t(\mathbf{x} - \mathbf{x}^*))\right](\mathbf{x} - \mathbf{x}^*)\mathrm{d}t = \int_0^1 \left[H_F(\mathbf{x}^* + t(\mathbf{x} - \mathbf{x}^*))\right](\mathbf{x} - \mathbf{x}^*)\mathrm{d}t.$$

Supposons qu'à l'étape $n \in \mathbb{N}$, on a $\mathbf{x}_n \in \mathcal{B}(\mathbf{x}^*, \varepsilon)$, alors d'après la formule d'itération (3.4), on a

$$\begin{aligned}(\mathbf{x}_{n+1} - \mathbf{x}^*) &= (\mathbf{x}_n - \mathbf{x}^*) - \tau \nabla F(\mathbf{x}_n) = (\mathbf{x}_n - \mathbf{x}^*) - \tau \int_0^1 H_F(\mathbf{x}^* + t(\mathbf{x}_n - \mathbf{x}^*))(\mathbf{x}_n - \mathbf{x}^*)\mathrm{d}t \\ &= \left(1 - \tau \int_0^1 H_F(\mathbf{x}^* + t(\mathbf{x}_n - \mathbf{x}^*))\mathrm{d}t\right)(\mathbf{x}_n - \mathbf{x}^*).\end{aligned}$$

La matrice entre parenthèse est une matrice symétrique dont les valeurs propres sont comprises entre $1 - \tau L$ et $1 - \tau l$. On en déduit que

$$\|\mathbf{x}_{n+1} - \mathbf{x}^*\| \leq \alpha \|\mathbf{x}_n - \mathbf{x}^*\|, \quad \text{avec} \quad \alpha := \max\{|1 - \tau l|, |1 - \tau L|\}.$$

Ainsi, si $\tau < 2/L$, on a $0 < \alpha < 1$. On en déduit premièrement que $\mathbf{x}_{n+1} \in \mathcal{B}(\mathbf{x}^*, \varepsilon)$ (et donc, par une récurrence immédiate, que $\mathbf{x}_n \in \mathcal{B}(\mathbf{x}^*, \varepsilon)$ pour tout $n \in \mathbb{N}$), puis que $|\mathbf{x}_n - \mathbf{x}^*| \leq \alpha |\mathbf{x}_{n-1} - \mathbf{x}^*| \leq \cdots \leq \alpha^n |\mathbf{x}_0 - \mathbf{x}^*|$. □

On a ainsi démontré la convergence linéaire de la méthode du gradient à pas constant. D'après le Théorème 3.16 et un raisonnement similaire au Lemme 3.12, on voit que le taux optimal est atteint pour un pas $\tau \approx 2/(\lambda_1 + \lambda_d)$. Dans ce cas, on peut s'attendre à une convergence linéaire à taux $\frac{\lambda_d - \lambda_1}{\lambda_d + \lambda_1}$. On retrouve le fait que la convergence est rapide si la hessienne $H_F(\mathbf{x}^*)$ est bien conditionnée.

Exercice 3.17

Avec les mêmes notations que le Lemme 3.15, montrer que pour tout $\mathbf{x}, \mathbf{y} \in \mathcal{B}(\mathbf{x}^*, \varepsilon)$, on a

$$F(\mathbf{x}) + \langle \nabla F(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle + \frac{l}{2}\|\mathbf{y} - \mathbf{x}\|^2 \leq F(\mathbf{y}) \leq F(\mathbf{x}) + \langle \nabla F(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle + \frac{L}{2}\|\mathbf{y} - \mathbf{x}\|^2.$$

et que

$$|\nabla F(\mathbf{x})| \leq L\|\mathbf{x}^* - \mathbf{x}\|.$$

3.3 Descente de gradient à pas (quasi-) optimal

Le principal défaut de la méthode du gradient à pas constant est que le pas τ doit être choisi adéquatement par l'utilisateur. De plus, même avec le choix du pas optimal, la vitesse est dictée par le conditionnement de la hessienne $H_F(\mathbf{x}^*)$. Nous nous intéressons maintenant à des méthodes où le pas τ est choisi automatiquement à chaque itération.

3.3.1 Pas optimal

On commence par étudier les propriétés du *pas optimal*. Dans ce cas, on considère $\mathbf{h}_n$ une direction de descente, et on pose

$$\boxed{\tau_n := \operatorname{argmin}\{F(\mathbf{x}_n + \tau \mathbf{h}_n), \tau \in \mathbb{R}^+\}, \quad \mathbf{x}_{n+1} := \mathbf{x}_n + \tau_n \mathbf{h}_n.} \qquad \text{(Gradient à pas optimal)} \quad (3.7)$$

Le problème de minimisation[3] (3.7) est uni-dimensionnel, de sorte qu'on peut utiliser les algorithmes du Chapitre 2. On fera cependant attention que cela peut augmenter considérablement le nombre d'appels à la fonction F.

Lemme 3.18 : Propriété d'orthogonalité au pas optimal

Si la suite $(\mathbf{x}_n)$ satisfait (3.7), alors, pour tout $n \in \mathbb{N}$, on a

$$\langle \nabla F(\mathbf{x}_{n+1}), \mathbf{h}_n \rangle = 0.$$

Démonstration. Soit $F_{\mathbf{x},\mathbf{h}}(t) := F(\mathbf{x}+t\mathbf{h})$. Si τ est un minimum (local) de $F_{\mathbf{x},\mathbf{h}}$, on doit avoir $F'_{\mathbf{x},\mathbf{h}}(\tau) = 0$, ou encore

$$\langle \nabla F(\mathbf{x} + \tau \mathbf{h}), \mathbf{h} \rangle = 0. \quad (3.8)$$

On obtient le résultat en prenant $\mathbf{x} = \mathbf{x}_n$ et $\mathbf{h} = \mathbf{h}_n$. □

En particulier, en prenant $\mathbf{h}_n := -\nabla F(\mathbf{x}_n)$, on a $\langle \nabla F(\mathbf{x}_{n+1}), F(\mathbf{x}_n) \rangle = 0$. Autrement dit, les directions de descente successives sont orthogonales. Cela peut se vérifier graphiquement, comme sur la Figure 3.3, où on optimise $Q(x_1, x_2) := \frac{1}{2}(\lambda_1 x_1^2 + \lambda_2 x_2^2)$ avec une descente de gradient à pas optimal (comparer avec la Figure 3.3).

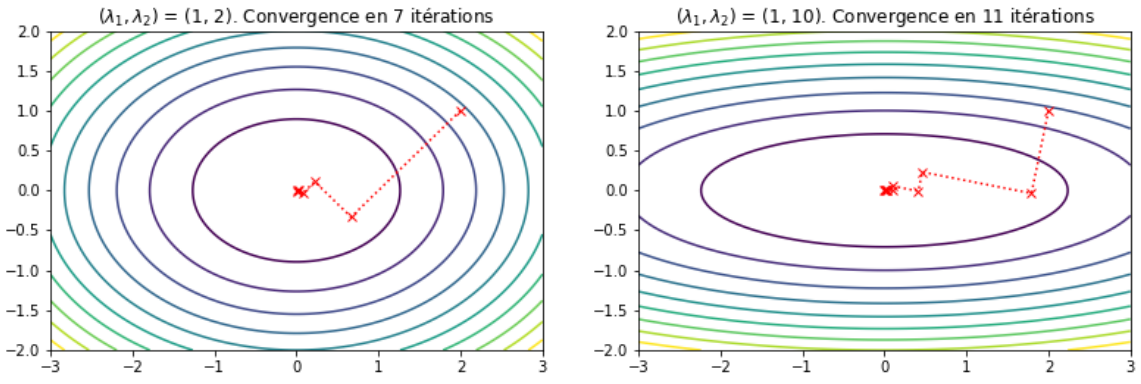

Figure 3.4 – Forme quadratique bien conditionnée (gauche), et mal conditionnée (droite).

En pratique, on observe une amélioration de la vitesse de convergence, surtout dans le cas mal conditionné, comme le montre la Figure 3.5. Sur cette figure, on voit que la vitesse de convergence de l'algorithme à pas optimal est toujours linéaire, mais avec un meilleur taux. Il est cependant difficile de montrer que cette convergence est effectivement linéaire, même dans des cas simples.

3. En réalité, on ne cherche pas le minimum global de la fonction, mais un minimum local, proche de $\tau \approx 0$.

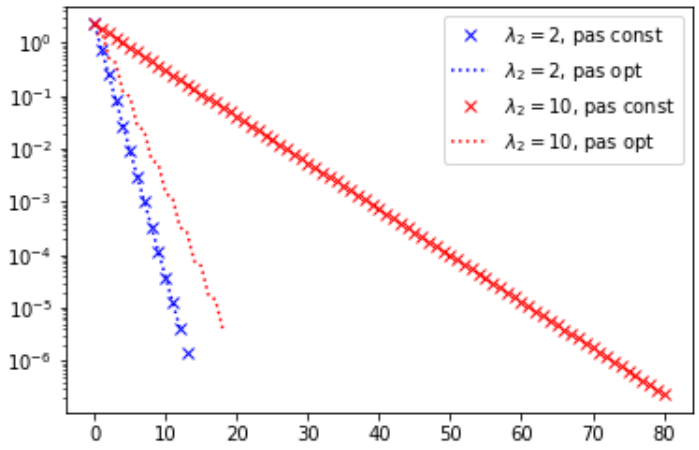

Figure 3.5 – Vitesse de convergence pour le gradient à pas optimal (-), et pas constant (x), dans le cas bien conditionné (bleu), et mal conditionné (rouge).

Exercice 3.19

Soit $A \in \mathcal{S}_d^{++}$ et soit $F : \mathbb{R}^d \to \mathbb{R}$ définie par $F(\mathbf{x}) = \mathbf{x}^T A\mathbf{x}$. Soit $\mathbf{x}_0 \in \mathbb{R}^n$, et $(\mathbf{x}_n)$ la suite obtenue avec le gradient à pas optimal (3.7).
a / Montrer qu'on a

$$\forall n \in \mathbb{N}, \quad \tau_n = -\frac{\mathbf{x}_n^T A^2 \mathbf{x}_n}{\mathbf{x}_n^T A^3 \mathbf{x}_n} \quad \text{et} \quad F(\mathbf{x}_{n+1}) = F(\mathbf{x}_n) - \frac{\left(\mathbf{x}_n A^2 \mathbf{x}_n\right)^2}{\mathbf{x}_n^T A^3 \mathbf{x}_n}.$$

b/ En utilisant l'Exercice 3.11, en déduire que

$$|F(\mathbf{x}_{n+1})| \leq \left(1 - \frac{\lambda_1}{\lambda_d}\right) |F(\mathbf{x}_n)|.$$

3.3.2 Pas quasi-optimal

Pour trouver le pas optimal, on est amené à minimiser le problème uni-dimensionel (3.7). En pratique, il n'est pas nécessaire de trouver l'optimal, et on peut se contenter de trouver un pas «raisonnable». La recherche d'un tel pas est appelé *recherche linéaire*. Dans cette section, $\mathbf{h}_n \in \mathbb{R}^d$ est une direction de descente au point $\mathbf{x}_n \in \mathbb{R}^d$. On rappelle que $F_{\mathbf{x}_n,\mathbf{h}_n}(\tau) := F(\mathbf{x}_n + \tau \mathbf{h}_n)$ est une fonction à seule variable.

Règle d'Armijo. La règle d'Armijo permet de choisir des pas «pas trop grands».

Définition 3.20 : Règle d'Armijo

Soit $G : \mathbb{R} \to \mathbb{R}$ une fonction à une seule variable telle que $G'(0) < 0$. On dit que $\tau > 0$ satisfait la règle d'Armijo pour le paramètre $0 < c_1 < 1$ si

$$G(\tau) \leq G(0) + c_1 \tau G'(0).$$

Dans le cas où $G = F_{\mathbf{x}_n,\mathbf{h}_n}$, cela s'écrit aussi

$$F(\mathbf{x}_n + \tau \mathbf{h}_n) \leq F(\mathbf{x}_n) + c_1 \tau \langle \nabla F(\mathbf{x}_n), \mathbf{h}_n \rangle.$$

Ici, la fonction $\widetilde{F}(t) := F(\mathbf{x}_n) + c_1 t\langle \nabla F(\mathbf{x}_n), \mathbf{h}_n\rangle$ représente la droite qui passe par $\widetilde{F}(t=0) = F(\mathbf{x}_n)$, et qui a une pente strictement plus grande que $\nabla F(\mathbf{x}_n)$ dans la direction $\mathbf{h}_n$ (voir Figure 3.6). On en déduit le résultat suivant.

Lemme 3.21 : Existence de paramètres pour la règle d'Armijo

Si G est de classe C^1 avec $G'(0) < 0$, alors pour tout $0 < c_1 < 1$, il existe $\tau_1 > 0$ tel que, pour tout $0 < \tau < \tau_1$, τ satisfait le critère d'Armijo de paramètre c_1.

Démonstration. On a $G'(0) < 0$, donc pour tout $0 < c_1 < 1$, on a $G'(0) < c_1 G'(0)$. Par continuité de G', il existe $\tau_1 > 0$ tel que,

$$\forall 0 < \tau < \tau_1, \quad G'(\tau) < c_1 G'(0) \quad (< 0).$$

On en déduit que pour tout $0 < \tau < \tau_1$, on a

$$G(\tau) = G(0) + \int_0^\tau G'(t)\mathrm{d}t \leq G(0) + \int_0^\tau c_1 G'(0)\mathrm{d}t = G(0) + c_1 \tau G'(0).$$

□

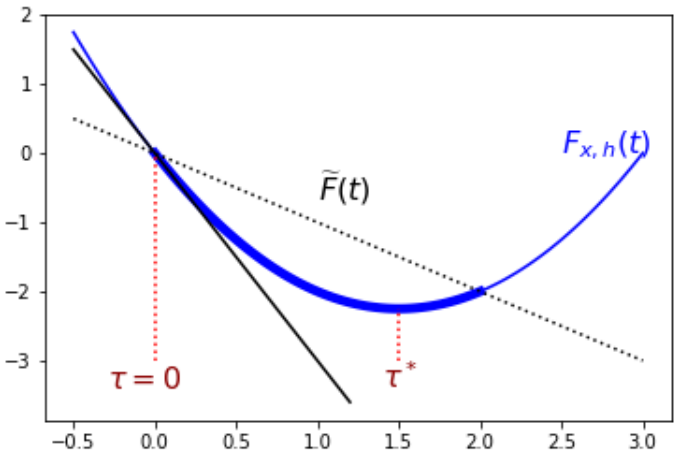

Figure 3.6 – Le critère d'Armijo. Ici, les pas satisfaisant le critère d'Armijo sont $\tau \in (0, 2)$.

La règle d'Armijo est un moyen de s'assurer que la suite $(F(\mathbf{x}_n))$ est décroissante. On voit sur la Figure 3.6 que lorsque c_1 est proche de 1, la droite $\widetilde{F}$ est proche de la tangente, de sorte que les points $(\mathbf{x}_n)$ sont rapprochés. En particulier, il se peut que le pas optimal τ^* ne satisfasse pas la règle d'Armijo si c_1 est trop proche de 1.

Règle de Wolfe. On s'intéresse maintenant à un critère pour prendre des pas «pas trop petits». Pour cela on rappelle que si τ^* est le pas optimal, alors on a la condition d'orthogonalité (3.8), i.e. $\langle \nabla F(\mathbf{x}_n + \tau^* \mathbf{h}_n), \mathbf{h}_n\rangle = 0$.

Définition 3.22 : Règle de Wolfe

Soit $G : \mathbb{R} \to \mathbb{R}$ une fonction à une seule variable telle que $G'(0) < 0$. On dit que $\tau > 0$ satisfait la règle de Wolfe pour les paramètres $0 < c_1 < c_2 < 1$ s'il satisfait la règle d'Armijo de paramètre c_1, et si

$$G'(\tau) \geq c_2 G'(0).$$

Dans le cas où $G = F_{\mathbf{x}_n, \mathbf{h}_n}$, cela s'écrit aussi

$$\langle \nabla F(\mathbf{x}_n + \tau \mathbf{h}_n), \mathbf{h}_n \rangle \geq c_2 \langle \nabla F(\mathbf{x}_n), \mathbf{h}_n \rangle. \tag{3.9}$$

Visuellement, la deuxième condition signifie que la dérivée de $F_{\mathbf{x},\mathbf{h}}$ au point τ doit-être significativement plus grande que celle au point 0, qu'on sait être négative (voir Figure 3.7). Le point optimal τ^* satisfait toujours cette condition, d'après la Proposition 3.18.

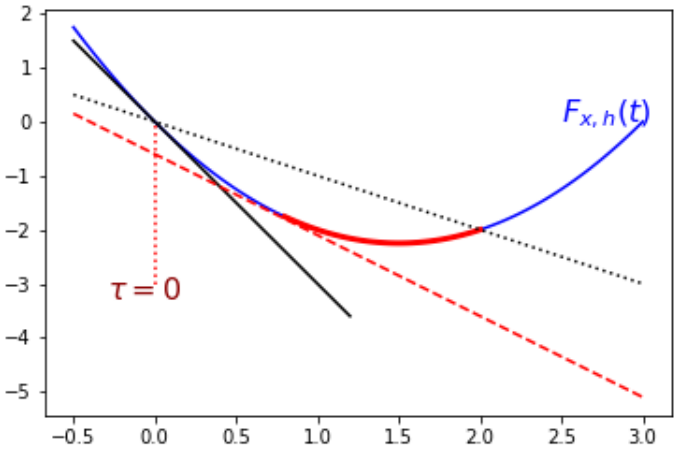

Figure 3.7 – Le critère de Wolfe. Ici, les pas satisfaisant le critère de Wolfe est $\tau \in (0.8, 2)$.

Lemme 3.23 : Existence de paramètres pour la règle de Wolfe

Si G est de classe C^1 telle que $G'(0) < 0$ et G est bornée inférieurement, alors pour tout $0 < c_1 < c_2 < 1$, il existe $0 < \tau_1 < \tau_2$ tel que pour tout $\tau \in (\tau_1, \tau_2)$, τ vérifie la règle de Wolfe.

Démonstration. D'après le Lemme 3.21, il existe $\tau^* > 0$ tel que pour tout $0 < \tau < \tau^*$, on a le critère d'Armijo $G(\tau) \leq G(0) + c_1 \tau G'(0)$. Soit τ_A le plus grand τ^* vérifiant cette propriété. Ce point existe car G est bornée inférieurement, alors que $\tau \mapsto G(0) + c_1 \tau G'(0)$ tend vers $-\infty$ lorsque $\tau \to \infty$.

Par continuité, ce point vérifie l'égalité

$$G(\tau_A) = G(0) + c_1 \tau_A G'(0), \quad \text{ou encore} \quad \frac{G(\tau_A) - G(0)}{\tau_A} = c_1 G'(0).$$

D'après le théorème des accroissements finis, il existe $\tau' \in (0, \tau_A)$ tel que $G'(\tau') = c_1 G'(0)$. De plus, comme $c_1 < c_2$ et $G'(0) < 0$, on a $G'(\tau') > c_2 G'(0)$. Donc τ' satisfait la règle de Wolfe. Par continuité de G', cette propriété est aussi satisfaite dans un voisinage de τ', ce qui démontre le résultat. □

Dichotomie pour les règles d'Armijo et de Wolfe. Pour trouver efficacement un pas $\tau > 0$ qui satisfait les régles d'Armijo et de Wolfe, on peut utiliser la méthode de dichotomie suivante. On

construit deux suites (τ_n) et (T_n) tel que $\tau_n < T_n$ et tel que les (τ_n) satisfont le critère d'Armijo, et les (T_n) ne le satisfont pas. On veut donc

$$\begin{cases} F(\mathbf{x}+\tau_n\mathbf{h}) \leq F(\mathbf{x}) + c_1\tau_n\langle\nabla F(\mathbf{x}_n),\mathbf{h}_n\rangle \\ F(\mathbf{x}+T_n\mathbf{h}) \geq F(\mathbf{x}) + c_1 T_n\langle\nabla F(\mathbf{x}_n),\mathbf{h}_n\rangle \end{cases}, \quad \text{et} \quad \lim\tau_n = \lim T_n.$$

On peut facilement construire de telles suites en posant à chaque itération $x_n := (\tau_n + T_n)/2$, puis

$$(\tau_{n+1}, T_{n+1}) = \begin{cases} (x_n, T_n) & \text{si } x_n \text{ satisfait le critère d'Armijo,} \\ (\tau_n, x_n) & \text{sinon.} \end{cases}$$

On arrête la construction dès que τ_n satisfait aussi la règle de Wolfe. On obtient le code suivant.

Code 3.3 – Recherche pour le critère de Wolfe

```
def pasWolfe(F, dF, x, h, tau0, T0, c1=0.2, c2=0.7, Niter=1000):
    Fx, dFx = F(x), dF(x)
    taun, Tn, xn = tau0, T0, (tau0 + T0)/2 #Initialisation

    for n in range(Niter):
        if dot(dF(x + taun*h), h) >= c2*dot(dFx,h): #xn satisfait Wolfe
            return taun
        if F(x + xn*h) <= Fx + c1*xn*dot(dFx,h): #xn satisfait Armijo
            taun, xn = xn, (xn+Tn)/2
        else:
            Tn, xn = xn, (taun+xn)/2
    print("Erreur, l'algorithme n'a pas convergé après", Niter, "itérations")
```

Exercice 3.24

Proposer un algorithme pour construire τ_0 et T_0 tel que τ_0 satisfait Armijo, mais pas T_0 ?

Lemme 3.25

Soit $F : \mathbb{R}^d \to \mathbb{R}$ de classe C^1, et soit $0 < c_1 < c_2 < 0$. Si $0 \leq \tau_0 < T_0$ est tel que τ_0 vérifie Armijo mais pas T_0, alors l'algorithme de dichotomie précédent termine en un nombre fini d'itérations.

Démonstration. Soit $\tau^* = \lim_n \tau_n = \lim_n T_n$. Comme τ_n vérifie Armijo, mais pas T_n, et $\tau_n < T_n$, on a

$$F(\mathbf{x}+T_n\mathbf{h}) - F(\mathbf{x}+\tau_n\mathbf{h}) \geq c_1(T_n-\tau_n)\langle\nabla F(\mathbf{x}_n),\mathbf{h}_n\rangle, \quad \text{ou} \quad \frac{F(\mathbf{x}+T_n\mathbf{h}) - F(\mathbf{x}+\tau_n\mathbf{h})}{(T_n-\tau_n)} \geq c_1\langle\nabla F(\mathbf{x}_n),\mathbf{h}_n\rangle.$$

Par continuité de ∇F, on obtient à la limite

$$\langle F'(\mathbf{x}+\tau^*\mathbf{h}),\mathbf{h}\rangle \geq c_1\langle\nabla F(\mathbf{x}_n),\mathbf{h}_n\rangle \quad > c_2\langle\nabla F(\mathbf{x}_n),\mathbf{h}_n\rangle.$$

Autrement dit, τ^* vérifie le critère de Wolfe. De plus, comme la fonction $\tau \mapsto \langle F'(\mathbf{x}+\tau\mathbf{h}),\mathbf{h}\rangle$ est continue, l'inégalité précédente est satisfaite dans un voisinage de τ^*. Comme (τ_n) converge vers τ^*, τ_n vérifie Wolfe à partir d'un certain rang. □

3.3.3 Convergence des méthodes de descente

On a vu plusieurs critères pour choisir le pas de descente, et nous verrons par la suite d'autres choix de direction de descente. Le théorème suivant donne un critère pour savoir si un algorithme converge.

Théorème 3.26 : Théorème de Zoutendijk

Soit $F : \mathbb{R}^d \to \mathbb{R}$ une fonction de classe C^1 bornée inférieurement, et tel que ∇F soit L-Lipschitz. Soit $(\mathbf{x}_n)$ la suite définie par $\mathbf{x}_0 \in \mathbb{R}^d$ et $\mathbf{x}_{n+1} = \mathbf{x}_n + \tau_n \mathbf{h}_n$, où $\mathbf{h}_n$ est un direction de descente de F au point $\mathbf{x}_n$, et où τ_n satisfait la règle de Wolfe. Alors

$$\sum_{n=0}^{\infty} \frac{|\langle \nabla F(\mathbf{x}_n), \mathbf{h}_n \rangle|^2}{\|\mathbf{h}_n\|^2} < \infty. \tag{3.10}$$

Par exemple, dans le cas où $\mathbf{h}_n = \nabla F(\mathbf{x}_n)$, le théorème de Zoutendijk affirme que la série des $\|(\nabla F(\mathbf{x}_n))\|^2$ est sommable, et donc en particulier que la suite $(\nabla F(\mathbf{x}_n))$ converge vers $\mathbf{0}$. Malheureusement, ceci ne permet pas toujours de conclure que la suite $(\mathbf{x}_n)$ converge vers $\mathbf{x}^*$. Par exemple, on ne sait pas *a priori* que $\mathbf{x}^*$ existe ! En revanche, ce sera le cas si F est par exemple coercive ($F(\mathbf{x}) \to +\infty$ si $\|\mathbf{x}\| \to \infty$).

Démonstration. On a, d'après le critère de Wolfe,

$$(0 \leq) \quad (c_2 - 1)\langle \nabla F(\mathbf{x}_n), \mathbf{h}_n \rangle \leq \langle \nabla F(\mathbf{x}_{n+1}) - \nabla F(\mathbf{x}_n), \mathbf{h}_n \rangle \leq \|\nabla F(\mathbf{x}_{n+1}) - \nabla F(\mathbf{x}_n)\| \cdot \|\mathbf{h}_n\|.$$

En utilisant le fait que F est L-Lipschitz, et que $\langle \nabla F(\mathbf{x}_n), \mathbf{h}_n \rangle < 0$, on obtient

$$(1-c_2)|\langle \nabla F(\mathbf{x}_n), \mathbf{h}_n \rangle| \leq L\|\mathbf{x}_{n+1} - \mathbf{x}_n\| \cdot \|\mathbf{h}_n\| = L\tau_n \|\mathbf{h}_n\|^2, \quad \text{ou encore,} \quad \tau_n \geq \frac{(1-c_2)}{L} \frac{|\langle \nabla F(\mathbf{x}_n), \mathbf{h}_n \rangle|}{\|\mathbf{h}_n\|^2}.$$

D'autre part, d'après le critère d'Armijo, on a

$$F(\mathbf{x}_n) - F(\mathbf{x}_{n+1}) \geq c_1 \tau_n |\langle \nabla F(\mathbf{x}_n), \mathbf{h}_n \rangle| \geq \frac{c_1(1-c_2)}{L} \frac{|\langle \nabla F(\mathbf{x}_n), \mathbf{h}_n \rangle|^2}{\|\mathbf{h}_n\|^2}.$$

En sommant ces inégalités avec $n \in \mathbb{N}$, on obtient

$$\sum_{n=0}^{\infty} \frac{|\langle \nabla F(\mathbf{x}_n), \mathbf{h}_n \rangle|^2}{\|\mathbf{h}_n\|^2} \leq \frac{L}{c_1(1-c_2)} \left(F(\mathbf{x}_0) - \inf F\right) \quad < \infty.$$

□

3.4 Exercices supplémentaire

Exercice 3.27

Soit $F(x, y) := y^2 - \cos(x)$. Montrer que F atteint des minima locaux en $(2\mathbb{Z}\pi, 0)$, et des points selles en $((2\mathbb{Z}+1)\pi, 0)$.

Exercice 3.28

On dit qu'une fonction $F : \mathbb{R}^d \to \mathbb{R}$ est *coercive* si pour tout $A \geq 0$, il existe $M \geq 0$ tel que pour tout $\mathbf{x} \in \mathbb{R}^d$, on a $F(\mathbf{x}) \geq A$ si $\|\mathbf{x}\| \geq M$.
Montrer qu'une fonction continue et coercive atteint son minimum.

Régression linéaire/polynômiale. Soit $g : \mathbb{R} \to \mathbb{R}$ une fonction inconnue qu'on cherche à calculer. On dispose d'un nombre fini d'échantillons $\{x_i\}_{i \in [1,m]}$ pour lesquels on connait $y_i := g(x_i)$. Le but de la régression linéaire est de trouver la meilleure approximation linéaire de g. C'est à dire qu'on

cherche une fonction P de la forme $P(x) := a_1\mathbf{x} + a_0$. L'erreur commise par la reconstruction avec P est mesurée grâce à la fonctionnelle

$$\mathcal{E}(P) := \sum_{i=1}^{m} |P(x_i) - g(x_i)|^2 \tag{3.11}$$

Dans la suite, on pose

$$X = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_m \end{pmatrix} \quad \text{et} \quad Y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}.$$

Exercice 3.29

a/ Soit $p := (a_0, a_1)^T \in \mathbb{R}^2$. Montrer que $\mathcal{E}(P) = \|Xp - Y\|^2$.
b/ Montrer que le minimiseur de $\mathcal{E}$ est aussi le minimiseur de $e(p) := \frac{1}{2}p^T(X^TX)p - (Y^TX)p$.
c/ En déduire que le minimiseur est $p^* := (X^TX)^{-1}(Y^TX)$.

Au lieu de chercher des fonctions linéaires, on peut aussi chercher à approcher la fonction g par un polynôme P de degré p, de la forme $P(x) = a_0 + a_1x + \ldots + a_px^p$. De nouveau, l'erreur est mesurée par la fonctionnelle $\mathcal{E}(P)$ définie en (3.11).

Exercice 3.30

a/ Comment modifier la matrice X pour avoir $\mathcal{E}(P) = \|Xp - Y\|^2$ avec $p = (a_0, a_1 \ldots a_p)^T \in \mathbb{R}^{p+1}$?
b/ Montrer que le minimum est de nouveau atteint pour $p^* = (X^TX)^{-1}(Y^TX)$.
c/ Que se passe-t-il si $m < p + 1$?

Conditionnement des matrices On appelle conditionnement d'une matrice $A \in \mathcal{S}_d^{++}(\mathbb{R})$ le nombre

$$\kappa(A) := \|A\|_{\text{op}} \cdot \|A^{-1}\|_{\text{op}}.$$

Exercice 3.31

a/ Montrer que $\kappa(\lambda A) = \kappa(A)$ et que $\kappa(A^p) = \kappa(A)^p$.
b/ Montrer que $\kappa(A) = \frac{\lambda_d}{\lambda_1}$, puis que $\kappa(A) \geq 1$.
c/ Montrer que $\kappa(A) = 1$ si et seulement si $A = \lambda\mathbb{I}_d$ pour un certain $\lambda > 0$.
d/ **(Reconstruction bruitée)** Soit $\mathbf{x}, \mathbf{b} \in \mathbb{R}^d$ tel que $A\mathbf{x} = \mathbf{b}$. On suppose qu'on connait $\mathbf{b}$ seulement de manière approchée, de sorte qu'on a accès seulement à $\mathbf{x} + \delta_{\mathbf{x}} = A^{-1}(\mathbf{b} + \delta_{\mathbf{b}})$. Montrer que

$$\frac{\|\delta_{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \kappa(A)\frac{\|\delta_{\mathbf{b}}\|}{\|\mathbf{b}\|} \quad \text{(contrôle de l'erreur relative)}.$$

Produit scalaire de Frobenius On définit sur l'ensemble des matrices $\mathcal{M}_{m,n}(\mathbb{R})$ le produit scalaire de Frobenius

$$\forall A, B \in \mathcal{M}_{m,n}(\mathbb{R}), \quad \langle A, B\rangle_{\mathcal{M}_{m,n}} := \text{Tr}(A^TB).$$

Exercice 3.32

a/ Montrer que $\langle \cdot, \cdot \rangle_{\mathcal{M}_{m,n}}$ est effectivement un produit scalaire.
b/ Dans le cas où $m = n$ et $A \in \mathcal{S}_n$, montrer que

$$\|A\|^2_{\mathcal{M}_n} = \lambda_1(A)^2 + \cdots + \lambda_n(A)^2.$$

c/ Montrer que toute forme linéaire de $\mathcal{M}_{m,n}(\mathbb{R}) \to \mathbb{R}$ est de la forme $M \mapsto \mathrm{Tr}(A^T M)$ pour un certain $A \in \mathcal{M}_{m,n}(\mathbb{R})$.
d/ Soit $F : \mathcal{M}_{m,n}(\mathbb{R}) \to \mathbb{R}$ une application de classe C^1. Montrer que pour tout $M \in \mathcal{M}_{m,n}(\mathbb{R})$, il existe $A_M \in \mathcal{M}_{m,n}(\mathbb{R})$ tel que $\mathrm{d}F(M) : H \mapsto \mathrm{Tr}(A_M^T H)$. En déduire que $A_M = \nabla F(M)$.

Remarque 3.33. *Dans l'exercice précédent, on arrive à donner un sens (plutôt simple) au gradient de F. Il est plus difficile de définir la Jacobienne (ce n'est plus la transposée du gradient : on a changé de produit scalaire). C'est pourquoi il est souvent plus simple de travailler avec le gradient qu'avec la Jacobienne.*

Exercice 3.34

Calculer le gradient (au sens de Frobenius) des fonctions suivantes :

$$F_1(M) = \mathrm{Tr}(M), \quad F_2(M) = \mathbf{x}^T M \mathbf{x}, \quad F_3(M) = \mathrm{Tr}(M^T M), \quad F_4(M) = \det(M).$$

CHAPITRE 4

LE GRADIENT CONJUGUÉ

Dans ce chapitre, nous présentons la *méthode du gradient conjugué*, qui est un algorithme qui permet de résoudre le problème linéaire $A\mathbf{x} = \mathbf{b}$, lorsque A est une matrice symétrique définie positive de taille $d \times d$. Cette méthode est itérative, et trouve la solution $\mathbf{x}^* := A^{-1}\mathbf{b}$ en d itérations au plus.

La présentation faite dans ce cours utilise des outils des espaces hilbertiens, que nous rappelons.

4.1 Rappels sur les espaces hilbertiens

Un espace de Hilbert est un espace vectoriel $\mathcal{H}$ dont la norme $\|\cdot\|$ vient d'un produit scalaire $\langle\cdot,\cdot\rangle$, et qui est complet pour cette norme. Dans ce cours, on s'intéresse uniquement au cas où $\mathcal{H}$ est de dimension finie, de dimension d (on parle aussi d'espace euclidien).

Une *base* de $\mathcal{H}$ est une famille libre de d vecteurs $\mathcal{B} := \{\mathbf{p}_1, \cdots, \mathbf{p}_d\}$. On a alors $\text{Vect}\{\mathbf{p}_1, \cdots, \mathbf{p}_d\} = \mathcal{H}$. La base est *orthogonale* si $\langle \mathbf{p}_i, \mathbf{p}_j\rangle = 0$ si $i \neq j$, et elle est *orthonormale* si $\langle \mathbf{p}_i, \mathbf{p}_j\rangle = \delta_{ij}$.

Si $\mathcal{B} = (\mathbf{p}_1, \cdots, \mathbf{p}_d)$ est une base orthogonale, alors, pour tout $\mathbf{x} \in \mathcal{H}$, la décomposition de $\mathbf{x}$ dans la base $\mathcal{B}$ est

$$\mathbf{x} := \sum_{i=1}^{d} x_i \mathbf{p}_i \quad \in \mathcal{H} \quad \text{avec} \quad x_i = \frac{\langle \mathbf{p}_i, \mathbf{x}\rangle}{\|\mathbf{p}_i\|^2} \in \mathbb{R}. \tag{4.1}$$

On en déduit, par bilinéarité du produit scalaire, que

$$\langle \mathbf{x}, \mathbf{y}\rangle = \left\langle \sum_{i=1}^{d} x_i \mathbf{p}_i, \sum_{j=1}^{d} y_j \mathbf{p}_j \right\rangle = \sum_{i=1}^{d}\sum_{j=1}^{d} x_i y_j \langle \mathbf{p}_i, \mathbf{p}_j\rangle = \sum_{i=1}^{d} x_i y_i \|\mathbf{p}_i\|^2,$$

où on a utilisé le fait que $\langle \mathbf{p}_i, \mathbf{p}_j\rangle = 0$ si $i \neq j$. En prenant $\mathbf{x} = \mathbf{y}$, on obtient l'identité de Parseval [1]

$$\|\mathbf{x}\|^2 = \sum_{i=1}^{d} x_i^2 \|\mathbf{p}_i\|^2 = \sum_{i=1}^{d} \frac{|\langle \mathbf{p}_i, \mathbf{x}\rangle|^2}{\|\mathbf{p}_i\|^2}.$$

Exercice 4.1

Soit $(\mathbf{p}_1, \cdots, \mathbf{p}_d) \in \mathbb{R}^d$ une base orthonormale de $\mathbb{R}^d$ muni du produit scalaire usuel. Montrer que

$$\sum_{i=1}^{d} \mathbf{p}_i \mathbf{p}_i^T = \mathbb{I}_d.$$

1. L'identité de Parseval est une généralisation du théorème de Pythagore.

Soit $E \subset \mathcal{H}$ un sous espace vectoriel de $\mathcal{H}$ de dimension $n \leq d$. Pour $\mathbf{x} \in \mathcal{H}$, la *projection orthogonale* de $\mathbf{x}$ sur E, notée $P_E(\mathbf{x})$, est la solution du problème de minimisation

$$P_E(\mathbf{x}) := \operatorname{argmin} \{\|\mathbf{x} - \mathbf{e}\|,\ \mathbf{e} \in E\}. \tag{4.2}$$

Lemme 4.2 : Propriétés de la projection

La projection $P_E(\mathbf{x})$ est bien définie, et est unique. De plus, si $\{\mathbf{p}_1, \dots, \mathbf{p}_n\}$ est une famille orthogonale telle que $E := \text{Vect}\{\mathbf{p}_1, \dots, \mathbf{p}_n\}$, alors

$$P_E(\mathbf{x}) = \sum_{i=1}^{n} \frac{\langle \mathbf{p}_i, \mathbf{x}\rangle}{\|\mathbf{p}_i\|^2}\mathbf{p}_i \quad \in E. \tag{4.3}$$

Enfin, le vecteur $\mathbf{x} - P_E(\mathbf{x})$ est orthogonal à l'espace vectoriel E, i.e.

$$\forall \mathbf{e} \in E, \quad \langle \mathbf{e}, \mathbf{x} - P_E(\mathbf{x})\rangle = 0.$$

Démonstration. On complète la famille orthogonale $\{\mathbf{p}_1, \dots, \mathbf{p}_n\}$ pour obtenir une base orthogonale $\{\mathbf{p}_1, \dots, \mathbf{p}_d\}$ de $\mathcal{H}$. Pour tout $\mathbf{e} \in \mathcal{E}$, on a la décomposition $\mathbf{e} = \sum_{i=1}^{n} e_i \mathbf{p}_i$. D'après l'identité de Parseval, on a

$$\|\mathbf{x} - \mathbf{e}\|^2 = \sum_{i=1}^{n} (x_i - e_i)^2 \|\mathbf{p}_i\|^2 + \sum_{i=n+1}^{d} x_i^2 \|\mathbf{p}_i\|^2, \quad \text{avec} \quad x_i = \frac{\langle \mathbf{p}_i, \mathbf{x}\rangle}{\|\mathbf{p}_i\|^2}.$$

La deuxième somme est indépendante de $\mathbf{e} \in E$, alors que la première partie est positive, et ne s'annule que si $e_i = x_i$ pour $1 \leq i \leq n$. On en déduit que l'unique minimum définissant $P_{\mathcal{E}}(\mathbf{x})$ est $\mathbf{e}^* = \sum_{i=1}^{n} x_i \mathbf{p}_i$, ce qui prouve (4.3). De plus, on a, pour tout $\mathbf{e} \in E$,

$$\langle \mathbf{e}, \mathbf{x} - P_E(\mathbf{x})\rangle = \sum_{i=1}^{n} e_i \cdot (x_i - x_i)\|\mathbf{p}_i\|^2 + \sum_{i=n+1}^{d} 0 \cdot x_i \|\mathbf{p}_i\|^2 = 0.$$

□

4.2 Principe des méthodes conjuguées

Soit $A \in \mathcal{S}_d^{++}(\mathbb{R})$ une matrice définie positive. Dans ce chapitre, on veut résoudre des problèmes de type $A\mathbf{x} = \mathbf{b}$, ou encore calculer $A^{-1}\mathbf{b}$ efficacement.

Dans le chapitre précédent, nous avons vu des méthodes itératives, ou la direction de descente était le gradient. Cette direction est définie à partir du produit scalaire usuel de $\mathbb{R}^d$. Il n'y cependant *a priori* aucune raison pour que ce produit scalaire soit pertinent pour résoudre notre problème : en faisant cela, on découple complètement la structure de notre espace de minimisation ($\mathbb{R}^d$), et la fonction qu'on veut minimiser (ici $Q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x}$). L'idée des méthodes conjuguées est de considérer un produit scalaire qui dépend du problème.

Pour $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$, on pose

$$\langle \mathbf{x}, \mathbf{y}\rangle_A := \mathbf{x}^T A\mathbf{y} \quad (= \mathbf{y}^T A\mathbf{x}).$$

Il est facile de vérifier que $\langle \cdot, \cdot\rangle_A$ est un produit scalaire sur $\mathbb{R}^d$, et que l'espace $\mathbb{R}^d$ muni du produit scalaire $\langle \cdot, \cdot\rangle_A$ est un espace euclidien. On appelle ce produit scalaire le *produit scalaire conjugué* (à la matrice A), et on dit que $\mathbf{x}$ et $\mathbf{y}$ sont A-orthogonaux si $\langle \mathbf{x}, \mathbf{y}\rangle_A = 0$. On note $\|\cdot\|_A$ la norme associée à $\langle \cdot, \cdot\rangle_A$ (cf. l'Exercice 3.11). On dit qu'une base $\mathcal{B} = (\mathbf{p}_1, \cdots, \mathbf{p}_d)$ est *conjuguée*, ou A-orthogonale, si elle est orthogonale pour le produit scalaire $\langle \cdot, \cdot\rangle_A$.

Exercice 4.3

Soit $Q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x}$, et soit $\mathbf{x}^* := A^{-1}\mathbf{b}$. Montrer que $Q(\mathbf{x}) = \frac{1}{2}\|\mathbf{x} - \mathbf{x}^*\|_A^2 - \frac{1}{2}\|\mathbf{x}^*\|_A^2$.

Soit $\mathbf{b} \in \mathbb{R}^d$ et soit $\mathbf{x}^*$ la solution de $A\mathbf{x} = \mathbf{b}$. Si $\mathcal{B} = (\mathbf{p}_1, \cdots, \mathbf{p}_d)$ est une base conjuguée, alors la décomposition (4.1) de $\mathbf{x}^*$ dans la base $\mathcal{B}$ pour le produit scalaire conjugué est

$$\mathbf{x}^* = \sum_{i=1}^{d} \alpha_i \mathbf{p}_i \quad \text{avec} \quad \alpha_i := \frac{\langle \mathbf{p}_i, \mathbf{x}^* \rangle_A}{\|\mathbf{p}_i\|_A^2} = \frac{\mathbf{p}_i^T A\mathbf{x}^*}{\mathbf{p}_i^T A\mathbf{p}_i} = \frac{\mathbf{p}_i^T \mathbf{b}}{\mathbf{p}_i^T A\mathbf{p}_i} \in \mathbb{R}.$$

Autrement dit, *si on connait une base conjuguée*, alors on pourrait calculer $\mathbf{x}^*$ directement avec la formule

$$\mathbf{x}^* = \sum_{i=1}^{d} \left(\frac{\mathbf{p}_i^T \mathbf{b}}{\mathbf{p}_i^T A\mathbf{p}_i} \right) \mathbf{p}_i. \tag{4.4}$$

Cette formule permet de résoudre le système $A\mathbf{x} = \mathbf{b}$ sans inverser A, et en utilisant uniquement des multiplications matrices/vecteurs (voir Section 3.2.3).

Le but des méthodes conjuguées est de construire une base conjuguée.

4.3 Le gradient conjugué

Il y a plusieurs méthodes pour construire une base conjuguée. La méthode la plus efficace aussi bien du point de vue de l'implémentation (simplicité du code), que de la stabilité numérique est celle du *gradient conjugué*, que nous présentons maintenant.

4.3.1 Algorithme

On appelle *espace de Krylov* de dimension (au plus) n l'espace

$$\mathcal{K}_n(\mathbf{b}) := \text{Vect}\{\mathbf{b}, A\mathbf{b}, \ldots, A^{n-1}\mathbf{b}\},$$

avec la convention $\mathcal{K}_0(\mathbf{b}) = \{\mathbf{0}\}$. On notera $\mathcal{K}_n := \mathcal{K}_n(\mathbf{b})$ pour simplifier les expressions. Il est facile de vérifier que

$$\{\mathbf{0}\} = \mathcal{K}_0 \subset \mathcal{K}_1 \subset \cdots \subset \mathcal{K}_d, \quad \text{et} \quad A\mathcal{K}_n := \text{Vect}\left\{A\mathbf{b}, A^2\mathbf{b}, \ldots, A^n\mathbf{b}\right\} \subset \mathcal{K}_{n+1}$$

On introduit trois familles de vecteurs $(\mathbf{x}_n)$, $(\mathbf{r}_n)$ et $(\mathbf{p}_n)$, respectivement définies par (attention aux indices)

$$\forall n \in \mathbb{N}, \quad \begin{cases} \mathbf{x}_{n+1} & := \text{argmin}\left\{Q(\mathbf{x}),\ \mathbf{x} \in \mathcal{K}_n\right\}, \\ \mathbf{r}_{n+1} & := -\nabla Q(\mathbf{x}_{n+1}) = \mathbf{b} - A\mathbf{x}_{n+1}, \\ \mathbf{p}_{n+1} & := \mathbf{r}_{n+1} - P_{\mathcal{K}_n}^A(\mathbf{r}_{n+1}). \end{cases} \tag{4.5}$$

Ici, la projection $P_{\mathcal{K}_n}^A$ est la projection orthogonale pour le produit scalaire conjugué.
Enfin, nous disons qu'une famille $\{\mathbf{u}_1, \cdots \mathbf{u}_n\}$ est *quasi-libre* (nom pas officiel) s'il existe $n_0 \leq n$ tel que $\{\mathbf{u}_1, \cdots, \mathbf{u}_{n_0}\}$ est libre et si $\mathbf{u}_{n_0+1} = \mathbf{u}_{n_0+2} = \cdots = \mathbf{u}_n = \mathbf{0}$.

Lemme 4.4 : Lemme clé du gradient conjugué

Pour tout $1 \leq n \leq d$, on a

(i) $\mathbf{x}_{n+1} = P_{\mathcal{K}_n}^A(\mathbf{x}^*)$;

(ii) $\{\mathbf{r}_1, \mathbf{r}_2, \cdots, \mathbf{r}_n\}$ est une quasi-base orthogonale de $\mathcal{K}_n$ (i.e. $\mathbf{r}_n$ est orthogonale à $\mathcal{K}_{n-1}$) ;

(iii) $\{\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_n\}$ est une quasi-base A-orthogonale de $\mathcal{K}_n$ (i.e. $\mathbf{p}_n$ est A-orthogonale à $\mathcal{K}_{n-1}$) ;

(iv) $\mathbf{r}_{n+1}$ est A-orthogonal à $\mathcal{K}_{n-1}$.

Autrement dit, à chaque itération, $\{\mathbf{r}_1, \dots, \mathbf{r}_n\}$ est une base orthogonale de $\mathcal{K}_n(\mathbf{b})$, et $\{\mathbf{p}_1, \dots, \mathbf{p}_n\}$ est une base A-orthogonale de cet espace.

Démonstration. Pour le point (i), on utilise l'Exercice 4.3, on a $Q(\mathbf{x}) = \frac{1}{2}\|\mathbf{x} - \mathbf{x}^*\|_A^2 - \frac{1}{2}\|\mathbf{x}^*\|_A^2$. Le terme $-\frac{1}{2}\|\mathbf{x}^*\|_A^2$ est une constante indépendante de $\mathbf{x}$, donc le minimiseur de Q sur $\mathcal{K}_n$ est aussi celui de $\|\mathbf{x} - \mathbf{x}^*\|_A^2$ sur $\mathcal{K}_n$. D'après la définition de la projection (4.2), on a bien $\mathbf{x}_{n+1} = P_{\mathcal{K}_n}^A(\mathbf{x}^*)$.

Montrons le point (ii) par récurrence. Pour $n = 0$, on a bien $\mathbf{r}_1 = \mathbf{b}$, donc $\mathcal{K}_1 = \mathrm{Vect}(\mathbf{r}_1)$. Supposons que $\{\mathbf{r}_1, \cdots, \mathbf{r}_n\}$ est une quasi-base orthogonale de $\mathcal{K}_n$. Montrons que $\mathbf{r}_{n+1}$ est orthogonal à $\mathcal{K}_n$. Comme $\mathbf{x}_{n+1}$ est le minimiseur de Q sur $\mathcal{K}_n$, on a

$$\forall \mathbf{x} \in \mathcal{K}_n,\ \forall t \in \mathbb{R}, \quad Q(\mathbf{x}_{n+1} + t\mathbf{x}) \geq Q(\mathbf{x}_{n+1}).$$

En dérivant par rapport à t, on obtient en $t = 0$, (ici, $\langle \cdot, \cdot \rangle$ est le produit scalaire *usuel*).

$$\forall \mathbf{x} \in \mathcal{K}_n,\ \forall t \in \mathbb{R}, \quad t\langle \nabla Q(\mathbf{x}_{n+1}), \mathbf{x}\rangle \geq 0, \quad \text{et donc} \quad \forall \mathbf{x} \in \mathcal{K}_n, \quad \langle \nabla Q(\mathbf{x}_{n+1}), \mathbf{x}\rangle = 0.$$

Avec la définition $\mathbf{r}_{n+1} = -Q(\mathbf{x}_{n+1})$, on en déduit que $\mathbf{r}_{n+1}$ est orthogonal à $\mathcal{K}_n$. Cela montre que $\{\mathbf{r}_1, \cdots \mathbf{r}_n, \mathbf{r}_{n+1}\}$ est orthogonale. Enfin, on a $\mathbf{r}_{n+1} = \mathbf{b} - A\mathbf{x}_{n+1}$ avec $\mathbf{b} \in \mathcal{K}_{n+1}$ et $A\mathbf{x}_{n+1} \in A\mathcal{K}_n \subset \mathcal{K}_{n+1}$, donc $\{\mathbf{r}_1, \cdots, \mathbf{r}_n, \mathbf{r}_{n+1}\}$ est une quasi-base de $\mathcal{K}_n$.

De même, on a $\mathbf{p}_1 = \mathbf{r}_1 = \mathbf{b}$ qui engendre $\mathcal{K}_1$ et si $\{\mathbf{p}_1, \cdots, \mathbf{p}_n\}$ est une quasi-base A-orthogonale de $\mathcal{K}_n$, on a

$$P_{\mathcal{K}_n}^A(\mathbf{p}_{n+1}) = P_{\mathcal{K}_n}^A(\mathbf{r}_{n+1}) - \left(P_{\mathcal{K}_n}^A\right)^2(\mathbf{r}_{n+1}) = \mathbf{0},$$

donc $\mathbf{p}_{n+1}$ est A-orthogonale à $\mathcal{K}_n$. De plus, comme $\mathbf{r}_{n+1} \in \mathcal{K}_{n+1}$ et $P_{\mathcal{K}_n}^A(\mathbf{r}_{n+1}) \in \mathcal{K}_n \subset \mathcal{K}_{n+1}$, on a aussi $\mathbf{p}_{n+1} \in \mathcal{K}_{n+1}$. Cela montre que $\{\mathbf{p}_1, \cdots \mathbf{p}_{n+1}\}$ est une quasi-base A-orthogonale de $\mathcal{K}_{n+1}$.

Enfin, pour le point (iv), si $\mathbf{y} \in \mathcal{K}_{n-1}$, alors $A\mathbf{y} \in \mathcal{K}_n$ est orthogonal à $\mathbf{r}_{n+1}$ d'après (ii). On obtient donc

$$\forall \mathbf{y} \in \mathcal{K}_{n-1}, \quad \langle \mathbf{r}_{n+1}, \mathbf{y}\rangle_A = \langle \mathbf{r}_{n+1}, A\mathbf{y}\rangle = 0.$$

□

Remarque 4.5. *Si $\mathbf{r}_{n_0} = \mathbf{0}$ pour un certain n_0, alors $\nabla Q(\mathbf{x}_{n_0}) = \mathbf{0}$, et $\mathbf{x}_{n_0}$ est le minimum de Q sur tout l'espace. À ce moment, les suites deviennent stationnaires, et en particulier $\mathbf{r}_n = \mathbf{0}$ et $\mathbf{p}_n = \mathbf{0}$ pour $n \geq n_0$. C'est pourquoi nous parlons de* quasi-bases.

Les équations (4.5) permettent déjà de définir notre base conjuguée $(\mathbf{p}_n)$. Cependant, telle quelle, cette méthode n'est pas encore praticable : il faut encore expliquer comment calculer numériquement et efficacement les suites qui apparaissent dans (4.5).

Pour $\mathbf{x}_{n+1}$, comme c'est la projection de $\mathbf{x}^*$ sur $\mathcal{K}_n$, on a d'après le Lemme 4.2,

$$\mathbf{x}_{n+1} = \underbrace{\sum_{j=1}^{n-1} \frac{\langle \mathbf{x}^*, \mathbf{p}_j\rangle_A}{\|\mathbf{p}_j\|_A^2}\mathbf{p}_j}_{\mathbf{x}_n} + \frac{\langle \mathbf{x}^*, \mathbf{p}_n\rangle_A}{\|\mathbf{p}_n\|_A^2}\mathbf{p}_n = \mathbf{x}_n + \alpha_n\mathbf{p}_n, \quad \text{avec} \quad \alpha_n := \frac{\langle \mathbf{x}^*, \mathbf{p}_n\rangle_A}{\|\mathbf{p}_n\|_A^2} = \frac{\langle \mathbf{b}, \mathbf{p}_n\rangle}{\|\mathbf{p}_n\|_A^2}.$$

Remarque 4.6. *Sous cette forme, on reconnait une méthode de type descente de gradient, où la direction de descente est $\mathbf{p}_n$, et où le pas est α_n.*

De même, on peut définir les $\mathbf{r}_n$ avec la même formule de récurrence, car

$$\mathbf{r}_{n+1} = \mathbf{b} - A\mathbf{x}_{n+1} = \mathbf{b} - A(\mathbf{x}_n + \alpha_n\mathbf{p}_n) = \mathbf{r}_n - \alpha_n A\mathbf{p}_n.$$

Enfin, pour calculer la projection qui apparaît dans la définition de $\mathbf{p}_{n+1}$, on utilise encore le Lemme 4.2 et on obtient

$$P_{\mathcal{K}_n}^A(\mathbf{r}_{n+1}) = \sum_{j=1}^{n} \frac{\langle \mathbf{r}_{n+1}, \mathbf{p}_j\rangle_A}{\|\mathbf{p}_j\|_A^2}\mathbf{p}_j.$$

D'après le point (iv) du Lemme précédent, $\mathbf{r}_{n+1}$ est A-orthogonal à $\mathcal{K}_{n-1}$. Tous les termes de la somme avec $j \leq n-1$ sont donc nuls. Ainsi, seul le terme $j = n$ survit dans la somme, puis

$$\mathbf{p}_{n+1} = \mathbf{r}_{n+1} + \beta_n \mathbf{p}_n \quad \text{avec} \quad \beta_n := -\frac{\langle \mathbf{r}_{n+1}, \mathbf{p}_n \rangle_A}{\|\mathbf{p}_n\|_A^2}. \tag{4.6}$$

Les constantes α_n et β_n, telles qu'elles sont actuellement définies, sont très sensibles au bruit numérique : $\langle \mathbf{x}, \mathbf{y} \rangle$ peut être très petit même si $\mathbf{x}$ et $\mathbf{y}$ sont grands. On préfère toujours calculer le produit scalaire de deux vecteurs presque colinéaires. Dans notre cas, cela est possible grâce au lemme suivant.

Lemme 4.7 : Formule de α et β

On a

$$\alpha_n = \frac{\|\mathbf{r}_n\|^2}{\|\mathbf{p}_n\|_A^2}, \qquad \text{et} \quad \beta_n = \frac{\|\mathbf{r}_{n+1}\|^2}{\|\mathbf{r}_n\|^2}.$$

Démonstration. Comme $\mathbf{p}_n$ est A-orthogonal à $\mathcal{K}_{n-1}$, et que $\mathbf{x}_n \in \mathcal{K}_{n-1}$, on a $\langle \mathbf{p}_n, \mathbf{x}_n \rangle_A = 0$, donc

$$\langle \mathbf{p}_n, \mathbf{b} \rangle = \langle \mathbf{p}_n, \mathbf{b} \rangle - \langle \mathbf{p}_n, \mathbf{x}_n \rangle_A = \langle \mathbf{p}_n, \mathbf{b} - A\mathbf{x}_n \rangle = \langle \mathbf{p}_n, \mathbf{r}_n \rangle.$$

Par ailleurs, comme $\mathbf{r}_n$ est orthogonal à $\mathcal{K}_{n-1}$, on trouve

$$\langle \mathbf{p}_n, \mathbf{r}_n \rangle = \langle \mathbf{r}_n - P^A_{\mathcal{K}_{n-1}}(\mathbf{r}_n), \mathbf{r}_n \rangle = \langle \mathbf{r}_n, \mathbf{r}_n \rangle.$$

Ces deux égalités montrent que $\langle \mathbf{p}_n, \mathbf{b} \rangle = \|\mathbf{r}_n\|^2$, ce qui donne la formule pour α_n. Pour β_n, on utilise d'abord que $\mathbf{r}_{n+1}$ est orthogonal à $\mathbf{r}_n$, donc

$$\langle \mathbf{r}_{n+1}, \mathbf{r}_{n+1} \rangle = \langle \mathbf{r}_{n+1}, \mathbf{r}_n - \alpha_n A\mathbf{p}_n \rangle = -\alpha_n \langle \mathbf{r}_{n+1}, \mathbf{p}_n \rangle_A,$$

donc

$$\beta_n = -\frac{\langle \mathbf{r}_{n+1}, \mathbf{p}_n \rangle_A}{\|\mathbf{p}_n\|_A^2} = \frac{1}{\alpha_n} \frac{\|\mathbf{r}_{n+1}\|^2}{\|\mathbf{p}_n\|_A^2} = \frac{\|\mathbf{r}_{n+1}\|^2}{\|\mathbf{r}_n\|^2}.$$

□

Pour résumer, on obtient l'algorithme (robuste) du gradient conjugué avec les itérations suivantes :

$$\begin{cases} \mathbf{x}_1 = \mathbf{0}, \quad \mathbf{r}_1 = \mathbf{b}, \quad \mathbf{p}_1 = \mathbf{b}, \\ \alpha_n = \dfrac{\|\mathbf{r}_n\|^2}{\|\mathbf{p}_n\|_A^2}, \quad \mathbf{x}_{n+1} = \mathbf{x}_n + \alpha_n \mathbf{p}_n, \quad \mathbf{r}_{n+1} = \mathbf{r}_n - \alpha_n A\mathbf{p}_n, \quad \beta_n = \dfrac{\|\mathbf{r}_{n+1}\|^2}{\|\mathbf{r}_n\|^2}, \quad \mathbf{p}_{n+1} = \mathbf{r}_{n+1} + \beta_n \mathbf{p}_n. \end{cases}$$

Voici une implémentation Python de l'algorithme. On remarquera que le vecteur $A\mathbf{p}_n$ est calculé une seule fois par itération : on n'effectue qu'une seule multiplication matrice/vecteur par itération.

Code 4.1 – Algorithme du gradient conjugué.

```
1  def gradientConjugue(A, b, tol=1e-6):
2      d = len(b)
3      xn, pn, rn, L = zeros(d), b, b, [] #Initialisation
4
5      for n in range(d+2):
6          if norm(rn) < tol: #Condition de sortie "usuelle"
7              return xn, L
8          L.append(xn)
9          Apn = dot(A, pn) #une seule multiplication matrice/vecteur
10         alphan = dot(rn, rn)/dot(pn, Apn)
11         xn, rnp1 = xn + alphan*pn, rn - alphan*Apn
12         pn, rn = rnp1 + dot(rnp1, rnp1)/dot(rn, rn)*pn, rnp1
13     print("Probleme, l'algorithme n'a pas convergé après",n,"itérations")
```

La condition de sortie est très importante : il est inutile de calculer $\mathbf{x}^*$ parfaitement, et on peut stopper les itérations dès qu'on a une bonne approximation de $\mathbf{x}^*$. En pratique, on observe un gain de temps très important : si A est une matrice définie positive aléatoire de taille 1000×1000, l'algorithme convergera en 100 ou 200 itérations environ !

Exercice 4.8

En considérant des matrices symétriques définies positives aléatoires, tracer le nombre d'itérations en fonction de la taille des matrices, en échelle `loglog`. Qu'observez-vous ?

4.3.2 Etude théorique

On étudie maintenant les propriétés de convergence de l'algorithme du gradient.

Théorème 4.9 : Vitesse de convergence pour la méthode du gradient conjugué

On note $0 < \lambda_1 \leq \cdots \leq \lambda_d$ les valeurs propres de A rangés en ordre croissant.

(i) (estimation exacte) Si A a au plus r valeurs propres distinctes, alors l'algorithme du gradient conjugué trouve la solution en au plus r itérations.

(ii) (estimation avec les valeurs propres) On a

$$\|\mathbf{x}_{n+1} - \mathbf{x}^*\|_A \leq \left(\frac{\lambda_{d-n} - \lambda_1}{\lambda_{d-n} + \lambda_1}\right) \|\mathbf{x}^*\|_A.$$

(iii) (estimation avec le conditionnement) On a

$$\|\mathbf{x}_{n+1} - \mathbf{x}^*\|_A \leq 2\left(\frac{\sqrt{\lambda_d} - \sqrt{\lambda_1}}{\sqrt{\lambda_d} + \sqrt{\lambda_1}}\right)^n \|\mathbf{x}^*\|_A.$$

Le point (i) montre en particulier qu'on trouve la solution exacte en *au plus* d itérations.

Démonstration. On ne donne pas une preuve complète de ce théorème, mais on donne les idées principales et élégantes qui permettent de démontrer ce type de résultat. On commence par remarquer que $\mathcal{K}_n = \text{Vect}\{\mathbf{b}, A\mathbf{b}, \cdots, A^{n-1}\mathbf{b}\} = \{P(A)\mathbf{b},\ P \in \mathbb{R}_{n-1}[X]\}$, où $\mathbb{R}_{n-1}[X]$ est l'ensemble des polynômes de degré inférieur ou égaux à $n-1$. De plus, on a, d'après le Lemme 4.4, que

$$\mathbf{x}_{n+1} = P_{\mathcal{K}_n}(\mathbf{x}^*) = \text{argmin}\{\|\mathbf{x}^* - \mathbf{x}\|_A,\ \mathbf{x} \in \mathcal{K}_n\},$$

donc, comme $\mathbf{b} = A\mathbf{x}^*$, on a

$$\|\mathbf{x}^* - \mathbf{x}_{n+1}\|_A = \min\{\|\mathbf{x}^* - P(A)\mathbf{b}\|_A,\ P \in \mathbb{R}_{n-1}[X]\} = \min\{\|\left(1 - AP(A)\right)\mathbf{x}^*\|_A,\ P \in \mathbb{R}_{n-1}[X]\}.$$

Pour évaluer cette dernière quantité, on peut se mettre dans une base de diagonalisation de A. On note $0 < \lambda_1 \leq \cdots \leq \lambda_d$ les valeurs propres de A rangés en ordre croissant, et on a (on note x_i^* les coefficients de $\mathbf{x}^*$ dans cette base)

$$\begin{aligned}\|\left(1 - AP(A)\right)\mathbf{x}^*\|_A^2 &= \sum_{i=1}^d \langle (1 - AP(A))\mathbf{x}^*, A(1 - AP(A))\mathbf{x}^*\rangle = \sum_{i=1}^d (1 - \lambda_i P(\lambda_i))^2 \lambda_i |x_i^*|^2 \\ &\leq \left(\sup_{1\leq i\leq d} |1 - \lambda_i P(\lambda_i)|\right)^2 \sum_{i=1}^d \lambda_i |x_i^*|^2 = \left(\sup_{1\leq i\leq d} |1 - \lambda_i P(\lambda_i)|\right)^2 \|\mathbf{x}^*\|_A^2.\end{aligned}$$

On obtient donc

$$\|\mathbf{x}^* - \mathbf{x}_{n+1}\|_A \leq \left(\min_{P\in\mathbb{R}_{n-1}[X]} \max_{1\leq i\leq d} |1 - \lambda_i P(\lambda_i)|\right) \|\mathbf{x}^*\|_A.$$

Le théorème se démontre alors en choisissant des polynômes particuliers. Par exemple, pour le point *(i)*, si on note $\mu_1 < \mu_2 < \cdots < \mu_r$ les valeurs propres distinctes de A, il suffit de considérer le polynôme interpolateur de Lagrange de degré $r-1$ tel que $P(\mu_i) = \mu_i^{-1}$ pour $1 \leq i \leq r$.

Pour le point *(ii)* et *(iii)*, il faut considérer des polynômes plus complexes, dont l'étude sort du cadre de ce cours. □

Nous avons vu dans la Lemme 3.12 qu'en utilisant un algorithme à pas constant, même en choisissant le pas constant optimal, on avait une convergence à taux (on note $\kappa := \lambda_d/\lambda_1 \geq 1$)

$$\tau_{\text{pas const}} := \frac{\lambda_d - \lambda_1}{\lambda_d + \lambda_1} = \frac{\kappa - 1}{\kappa + 1}.$$

Le troisième point du Théorème 4.9 montre qu'on a une convergence à taux

$$\tau_{\text{GC}} := \frac{\sqrt{\lambda_d} - \sqrt{\lambda_1}}{\sqrt{\lambda_d} + \sqrt{\lambda_1}} = \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}.$$

On remarque qu'on a

$$\tau_{\text{GC}} = \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} = \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\frac{\sqrt{\kappa} + 1}{\sqrt{\kappa} + 1} = \frac{\kappa - 1}{\kappa + 1 + 2\sqrt{\kappa}} \leq \frac{\kappa - 1}{\kappa + 1} = \tau_{\text{pas const}}.$$

Ainsi, la convergence avec le gradient conjugué est toujours meilleure que celle avec le gradient à pas constant. En fait, lorsque κ devient très grand (mauvais conditionnement), on a

$$\tau_{\text{pas const}} \approx 1 - \frac{1}{\kappa} \quad \text{et} \quad \tau_{\text{GC}} \approx 1 - \frac{1}{\sqrt{\kappa}}.$$

Par exemple, si $\kappa = 100$, alors $\tau_{\text{pas const}} = 0.99$, alors que $\tau_{\text{GC}} = 0.9$. Pour obtenir une précision de 10^{-1}, il faudrait $\ln(0.1)/\ln(0.99) \approx 230$ itérations avec le pas constant, et $\ln(0.1)/\ln(0.9) \approx 22$ itérations avec le gradient conjugué. Le gain de temps est déjà conséquent !

Exercice 4.10

Soit $N_{\text{pas const}}(\varepsilon)$ et $N_{\text{GC}}(\varepsilon)$ le nombre d'itérations pour atteindre une précision ε avec l'algorithme du pas constant, et l'algorithme du gradient conjugué respectivement. Montrer que, pour $\varepsilon > 0$ petit, on a

$$\frac{N_{\text{pasconst}}(\varepsilon)}{N_{\text{GC}}(\varepsilon)} = \frac{\ln(1 - \sqrt{\kappa}^{-1})}{\ln(1 - \kappa^{-1})} \sim_{\kappa \to \infty} \sqrt{\kappa}.$$

4.4 Extensions non-linéaires

Dans la méthode du gradient conjugué, on considère à chaque itération 2 directions. La première est l'opposée du gradient $\mathbf{r}_n = -\nabla Q(\mathbf{x}_n)$, et la deuxième est la direction $\mathbf{p}_n$ qu'on prend effectivement. En fait, à partir de $\mathbf{x}_n$ et $\mathbf{p}_n$,

— on pose $\mathbf{x}_{n+1} = \text{argmin}\{Q(\mathbf{x}_n + t\mathbf{p}_n),\ t \in \mathbb{R}\}$;
— on construit la direction du gradient $\mathbf{r}_n = -\nabla Q(\mathbf{x}_n)$;
— on modifie cette direction avec une formule de type $\mathbf{p}_n = \mathbf{r}_n + \beta_n \mathbf{p}_{n-1}$.

La première étape est une recherche linéaire optimale dans la direction $\mathbf{p}_n$. Dans le cas *linéaire*, où on résout $A\mathbf{x} = \mathbf{b}$, et où on minimise la fonctionnelle quadratique $Q(\mathbf{x})$, cette recherche est explicite, et on trouve $\mathbf{x}_{n+1} = \mathbf{x}_n + \alpha_n \mathbf{p}_n$. De plus, dans le cas *linéaire*, on a aussi une formule explicite pour β_n, avec $\beta_n = \dfrac{\|\mathbf{r}_{n+1}\|^2}{\|\mathbf{r}_n\|^2}$.

Cette interprétation permet d'étendre *par analogie* la méthode du gradient conjugué à des fonctionnelles $F : \mathbb{R}^d \to \mathbb{R}$ plus générales, non nécessairement quadratiques ! Dans ce cas, les deux premiers points restent les mêmes, et il ne reste qu'à expliquer comment choisir les nombres $\beta_n \in \mathbb{R}$. Plusieurs choix ont été proposés, et donnent des méthodes différentes. Nous décrivons deux d'entre elles, qui sont très utilisées en pratique. L'étude théorique de ces méthodes sort du cadre de ce cours.

Méthode de Fletcher–Reeves Dans ce cas, on prend, par *analogie de formule* avec le cas linéaire,

$$\beta_n^{\mathrm{FR}} := \frac{\|\mathbf{r}_{n+1}\|^2}{\|\mathbf{r}_n\|^2} = \frac{\|\nabla F(\mathbf{x}_{n+1})\|^2}{\|\nabla F(\mathbf{x}_n)\|^2}.$$

Du point de vue théorique, cette méthode fonctionne : on peut prouver que cette méthode converge vers un minimum de F. Cependant, en pratique, la convergence est très lente, et on lui préfère généralement la méthode de Polak–Ribière.

Polak–Ribière Dans ce cas, on prend

$$\beta_n^{\mathrm{PR}} := \frac{\langle \mathbf{r}_{n+1}, \mathbf{r}_{n+1} - \mathbf{r}_n \rangle}{\|\mathbf{r}_n\|^2} = \frac{\langle \nabla F(\mathbf{x}_{n+1}), \nabla F(\mathbf{x}_{n+1}) - \nabla F(\mathbf{x}_n) \rangle}{\|\nabla F(\mathbf{x}_n)\|^2}.$$

Pour le cas linéaire, on a bien $\beta_n^{\mathrm{PR}} = \beta_n$, car la famille $(\mathbf{r}_n)$ est orthogonale. Mais dans le cas non-linéaire, ces deux quantités sont différentes. La méthode de Polak-Ribière peut ne pas converger (il existe des cas pathologiques), mais en pratique, elle converge dans la majorité des cas, et surtout, elle converge plus rapidement que la méthode de Fletcher-Reeves.

Dans le cas linéaire, une propriété de β_n est d'être toujours positif. La méthode de Polak-Ribière est souvent modifiée pour prendre en compte cette propriété. En général, on *recommence* la méthode lorsque qu'on trouve un $\beta_n^{\mathrm{PR}} \leq 0$. Cela donne la formule suivante :

$$\beta_n^{\mathrm{PR}} := \max \left\{ \frac{\mathbf{r}_{n+1}^T (\mathbf{r}_{n+1} - \mathbf{r}_n)}{\mathbf{r}_n^T \mathbf{r}_n}, 0 \right\}.$$

CHAPITRE 5

RETOUR SUR L'ALGORITHME DE NEWTON

5.1 La méthode de Newton en plusieurs dimensions

Dans cette section, nous revisitons la méthode de Newton, déjà vue à la Section 2.2.1 (pour les fonctions uni-dimensionnelles), dans le cas des fonctions à plusieurs variables.

Dans le cas des fonctions à une seule variable, nous avons insisté sur le fait que la méthode de Newton permettait de *résoudre* des équations de type $f(x) = 0$, et qu'en particulier, si $f = F'$, on pouvait calculer les points critiques de F. En plusieurs dimensions, les algorithmes de Newton sont surtout utilisés pour *optimiser* F. En particulier, on utilisera beaucoup le fait que la hessienne H_F est *symétrique définie positive*.

5.1.1 Newton classique

Soit $F : \mathbb{R}^d \to \mathbb{R}$ de classe C^2, et soit $\mathbf{x}^*$ un minimum non dégénéré de F. Alors, pour tout $\mathbf{x}_0$ proche de $\mathbf{x}^*$, on a l'approximation quadratique

$$F(\mathbf{x}) \approx \widetilde{F}_{\mathbf{x}_0} := F(\mathbf{x}_0) + \langle \nabla F(\mathbf{x}_0), \mathbf{x} - \mathbf{x}_0 \rangle + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0) H_F(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0),$$

et la matrice hessienne $H_F(\mathbf{x}_0)$ est définie positive, donc inversible. La fonction $\widetilde{F}_{\mathbf{x}_0}$ est quadratique, de la forme $\widetilde{F}_{\mathbf{x}_0}(\mathbf{h}) = \text{cst} + \frac{1}{2}\mathbf{h}^T A \mathbf{h} - \mathbf{b}^T \mathbf{h}$ avec $\mathbf{h} := \mathbf{x} - \mathbf{x}_0$, $A = H_F(\mathbf{x}_0)$ et $\mathbf{b} = -\nabla F(\mathbf{x}_0)$. Le minimiseur est donc le point $\mathbf{h}^* := A^{-1}\mathbf{b}$, ou encore $\mathbf{x}^* = \mathbf{x}_0 - [H_F(\mathbf{x}_0)]^{-1} \nabla F(\mathbf{x}_0)$. Cela suggère d'utiliser la formule de récurrence (comparer avec (2.6))

$$\mathbf{x}_{n+1} := \mathbf{x}_n - [H_F(\mathbf{x}_n)]^{-1} \nabla F(\mathbf{x}_n). \tag{5.1}$$

Exercice 5.1

Soit $\mathbf{b} \in \mathbb{R}^d$ et $A \in \mathcal{S}_n^{++}$, et soit $F : \mathbb{R}^d \to \mathbb{R}$ définie par $F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A \mathbf{x} - \mathbf{b}^T \mathbf{x}$. Comment s'écrit l'algorithme de Newton dans ce cas ? Que se passe-t-il ?

Exercice 5.2

Montrer que pour $\mathbf{x}_n$ proche de $\mathbf{x}^*$, le vecteur $\mathbf{h}_n := -[H_F(\mathbf{x}_n)]^{-1} \nabla F(\mathbf{x}_n)$ est effectivement une direction de descente de F.

Dans la Figure 5.1, on calcule le minimum de la fonction $F(x_1, x_2) := x_1^4(1 + x_2^2) - \cos(x_1) + x_2^2$ (qui est le point $\mathbf{x}^* = (0, 0)$). On observe une convergence super-linéaire. En fait, on peut montrer que

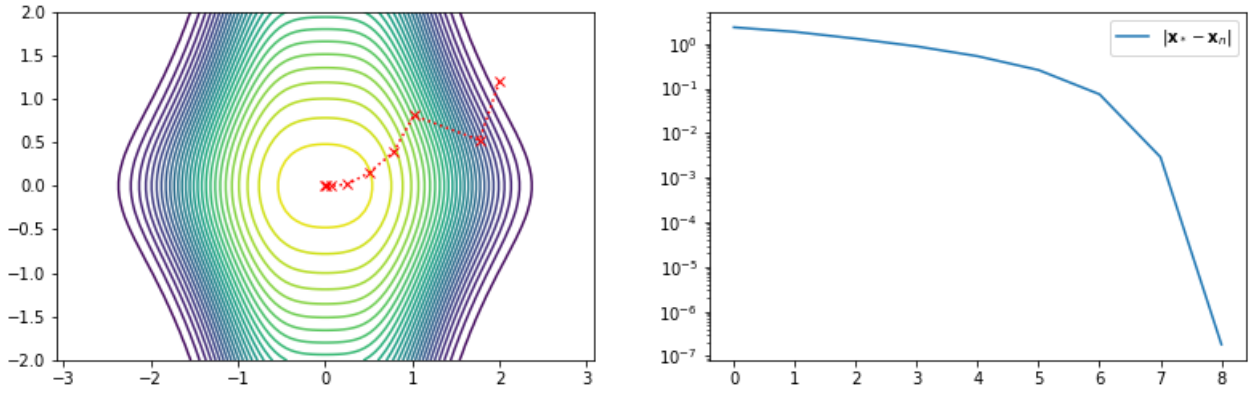

Figure 5.1 – La convergence super-linéaire de l'algorithme de Newton en plusieurs dimensions.

la convergence est quadratique (la preuve est similaire à celle dans l'Exercice 2.12). On peut l'observer en calculant la suite $\|\mathbf{x}^* - \mathbf{x}_n\| = \|\mathbf{x}_n\|$, ce qui donne

$$\texttt{[norm(xk)for xk in L]} = [2.3, 1.8, 1.2, 0.87, 0.53, 0.26, 0.074, 0.0029, 0.000000183].$$

Comme nous l'avons vu à la Section 2.2.1, la méthode de Newton résout l'équation $\nabla F = \mathbf{0}$. On peut donc aussi trouver des points critiques de F, comme des *points selles*, c'est à dire des points $\mathbf{x}^* \in \mathbb{R}^d$ où $\nabla F(\mathbf{x}^*) = \mathbf{0}$, mais où $H_F(\mathbf{x}^*)$ n'est pas nécessairement positive.

Dans la figure suivante, nous considérons la fonction

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

qui a 2 minima $(\pm 3/2, \mp 3/2)$ et 1 point selle $(0, 0)$. Pour chaque point $\mathbf{x}_0 \in \mathbb{R}^d$, nous lançons l'algorithme de Newton à partir de ce point, et nous colorions $\mathbf{x}_0$ de différentes couleurs suivant le point où l'algorithme converge. On obtient une fractale (voir Figure 5.2). Cela montre d'une part que l'algorithme de Newton peut converger vers un point qui n'est ni un minimum, ni un maximum, et de l'autre que cette méthode est très sensible à la condition initiale.

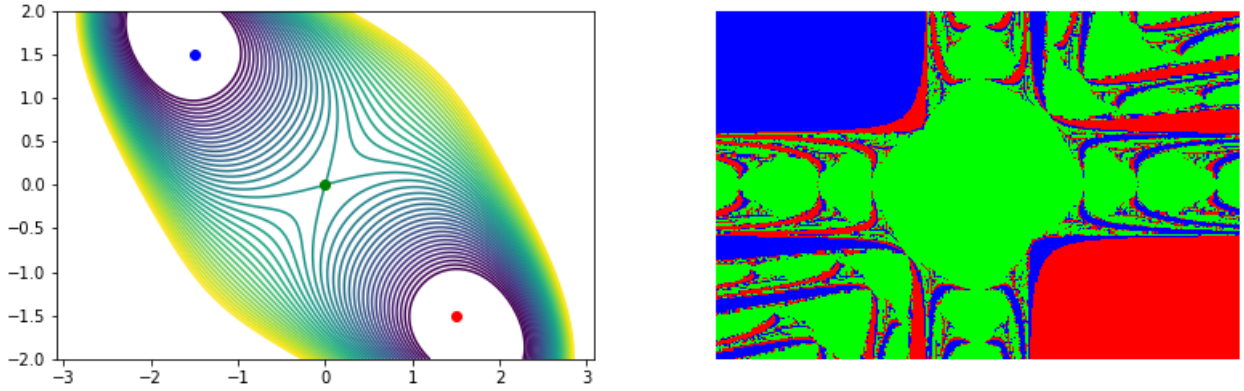

Figure 5.2 – La méthode de Newton avec différentes initialisations.

La méthode de Newton en plusieurs dimensions a un grand défaut : si F a un grand nombre de variables d, alors la hessienne H_F, qui est une matrice de taille $d \times d$, peut être très/trop grande. Non

seulement cela peut poser des problèmes en terme de stockage de la matrice, mais il faut aussi inverser cette matrice, ce qui peut prendre beaucoup de temps.

Plusieurs algorithmes ont été proposés pour remédier à ce problème de grande dimension.

5.1.2 Méthode de Newton + gradient conjugué

Pour commencer, on peut remarquer qu'il est en réalité inutile d'inverser la matrice hessienne $H_F(\mathbf{x}_n)$. En effet, l'idée est de remplacer la formule $\mathbf{x}_{n+1} = \mathbf{x}_n - [H_F(\mathbf{x}_n)]^{-1}\nabla F(\mathbf{x}_n)$ par

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \mathbf{h}_n, \quad \text{où } \mathbf{h}_n \text{ est la solution de } \quad H_F(\mathbf{x}_n)\mathbf{h}_n = \nabla F(\mathbf{x}_n).$$

Ainsi, au lieu d'inverser la matrice $H_F(\mathbf{x}_n)$, on se ramène à résoudre un système linéaire, ce qui peut être fait efficacement par l'algorithme du gradient conjugué. On obtient alors la méthode de Newton + gradient conjugué (*Newton-GC*).

En pratique, on n'effectue que quelques itérations du gradient conjugué, et on trouve un vecteur $\widetilde{\mathbf{h}_n}$, qui est proche de $\mathbf{h}_n$, mais qui n'est pas forcément la vraie direction de Newton. Cependant, si l'approximation est suffisamment bonne, $\widetilde{\mathbf{h}_n}$ est une direction de descente, et on peut effectuer une recherche linéaire [1] (critère de Wolfe) dans la direction $\widetilde{\mathbf{h}_n}$.

5.2 La méthode BFGS

5.2.1 Présentation

Une autre idée est d'approcher la Hessienne $H_F(\mathbf{x}_{n+1})$ par une matrice B_{n+1} qui peut être facilement stockée, dans le sens où on peut faire la multiplication matrice/vecteur avec B_n de manière efficace (cf Section 3.2.3), et dont l'inverse B_{n+1}^{-1} peut aussi être facilement calculée et stockée. L'idée de la méthode BFGS [2] est d'utiliser les propriétés des matrices de faible rang.

Exercice 5.3

Soit $A \in \mathcal{S}_d^+(\mathbb{R})$ une matrice de rang r, avec $r \ll d$.
a/ Montrer qu'il existe des nombres positifs $0 < \lambda_1 \leq \cdots \leq \lambda_r$ et des vecteurs $u_1, \dots u_r \in \mathbb{R}^d$ tel que

$$A = \sum_{i=1}^{r} \lambda_i u_i u_i^T.$$

b/ En déduire une méthode efficace pour calculer la multiplication $\mathbf{x} \mapsto A\mathbf{x}$.

Afin de construire une bonne approximation B_{n+1} de $H_F(\mathbf{x}_{n+1})$, on cherche des *bonnes* propriétés de $H_F(\mathbf{x}_{n+1})$ à partir des vecteurs $(\mathbf{x}_n, \mathbf{x}_{n+1}, \nabla F(\mathbf{x}_n), \nabla F(\mathbf{x}_{n+1}))$. Pour commencer, $H_F(\mathbf{x}_{n+1})$ est symétrique définie positive. De plus, on a, d'après la formule de Taylor pour ∇F,

$$\nabla F(\mathbf{x}_n) \approx \nabla F(\mathbf{x}_{n+1}) + H_F(\mathbf{x}_{n+1})(\mathbf{x}_n - \mathbf{x}_{n+1}) \qquad \text{(attention à l'ordre !)}.$$

Il est donc naturel de chercher B_{n+1} tel que B_{n+1} soit proche de B_n, que $B_{n+1} \in \mathcal{S}_d^{++}(\mathbb{R})$ et que

$$B_{n+1}\mathbf{s}_n = \mathbf{y}_n \quad \text{avec} \quad \mathbf{s}_n := \mathbf{x}_{n+1} - \mathbf{x}_n \quad \text{et} \quad \mathbf{y}_n := \nabla F(\mathbf{x}_{n+1}) - \nabla F(\mathbf{x}_n). \tag{5.2}$$

1. En dimension $d = 1$, *faire une recherche linéaire* est presque une tautologie. S'il n'y a qu'une variable, faire une recherche linéaire consiste justement à trouver le minimum de $F(x)$...
2. BFGS pour les auteurs de la méthode : Broyden–Fletcher–Goldfarb–Shanno.

Exercice 5.4

a/ Montrer que si $B_{n+1} \in \mathcal{S}_d^{++}$ vérifie (5.2), alors

$$\langle \mathbf{x}_{n+1} - \mathbf{x}_n, \nabla F(\mathbf{x}_{n+1}) - \nabla F(\mathbf{x}_n) \rangle > 0. \tag{5.3}$$

b/ Montrer que (5.3) est vérifiée si $\mathbf{x}_{n+1} - \mathbf{x}_n$ vérifie la règle de Wolfe au point $\mathbf{x}_n$.

En dimension $d = 1$, et en notant $F' = f$, cela donne

$$B_{n+1} = \frac{f(x_{n+1}) - f(x_n)}{x_{n+1} - x_n}, \quad \text{et donc} \quad x_{n+2} = x_{n+1} - \frac{x_{n+1} - x_n}{f(x_{n+1}) - f(x_n)} f(x_{n+1}).$$

On reconnait la méthode de la sécante (2.7). La méthode BFGS est donc une généralisation à plusieurs dimensions de la méthode de la sécante.

Alors qu'en dimension $d = 1$, l'équation (5.2) détermine de manière unique B_n, ce n'est plus vrai en dimension $d > 1$. Cela vient du fait qu'imposer (5.2) donne d contraintes, alors que l'ensemble des matrices symétriques est de dimension $d(d+1)/2$. La méthode BFGS consiste à choisir $B_0 = \mathbb{I}_d$, puis [3]

$$B_{n+1} = B_n - \frac{|B_n \mathbf{s}_n\rangle\langle B_n \mathbf{s}_n|}{\langle \mathbf{s}_n, B_n \mathbf{s}_n \rangle} + \frac{|\mathbf{y}_n\rangle\langle \mathbf{y}_n|}{\langle \mathbf{y}_n, \mathbf{s}_n \rangle}. \tag{5.4}$$

Lemme 5.5

La matrice B_{n+1} vérifie $B_{n+1}\mathbf{s}_n = \mathbf{y}_n$. De plus, la matrice $B_{n+1} - B_n$ est de rang au plus 2. En particulier, $B_n - \mathbb{I}_d$ est de rang au plus $2n$.

Démonstration. Pour le premier point, un calcul donne

$$B_{n+1}\mathbf{s}_n = B_n\mathbf{s}_n - |B_n\mathbf{s}_n\rangle \frac{\langle B_n\mathbf{s}_n, \mathbf{s}_n\rangle}{\langle \mathbf{s}_n, B_n\mathbf{s}_n\rangle} + |\mathbf{y}_n\rangle \frac{\langle \mathbf{y}_n, \mathbf{s}_n\rangle}{\langle \mathbf{y}_n, \mathbf{s}_n\rangle} = \mathbf{y}_n.$$

Ensuite, on voit que $B_{n+1} - B_n = \alpha_u |\mathbf{u}\rangle\langle \mathbf{u}| + \alpha_v |\mathbf{v}\rangle\langle \mathbf{v}|$ avec $\alpha_u := 1/\langle \mathbf{s}_n, B_n\mathbf{s}_n\rangle \in \mathbb{R}$, $\alpha_v := 1/\langle \mathbf{y}_n, \mathbf{s}_n\rangle \in \mathbb{R}$, $\mathbf{u} := B_n\mathbf{s}_n \in \mathbb{R}^d$ et $\mathbf{v} := \mathbf{y}_n \in \mathbb{R}^d$. Les matrices $|\mathbf{u}\rangle\langle \mathbf{u}|$ et $|\mathbf{v}\rangle\langle \mathbf{v}|$ sont de rang 1, ce qui démontre le Lemme. □

D'après le lemme précédent B_{n+1} est un bon candidat pour approcher $H_F(\mathbf{x}_{n+1})$. Il reste encore à construire l'inverse de B_{n+1} de manière efficace. Dans la suite, on note $W_n := B_n^{-1}$. On retiendra dans la mémoire à la fois la matrice B_n, et la matrice $W_n = B_n^{-1}$ comme deux matrices indépendantes. On a le résultat suivant.

Lemme 5.6

On a $W_0 = \mathbb{I}_d$, et

$$W_{n+1} := \left(\mathbb{I}_d - \frac{|\mathbf{s}_n\rangle\langle \mathbf{y}_n|}{\langle \mathbf{s}_n, \mathbf{y}_n\rangle}\right) W_n \left(\mathbb{I}_d - \frac{|\mathbf{y}_n\rangle\langle \mathbf{s}_n|}{\langle \mathbf{s}_n, \mathbf{y}_n\rangle}\right) + \frac{|\mathbf{s}_n\rangle\langle \mathbf{s}_n|}{\langle \mathbf{s}_n, \mathbf{y}_n\rangle}.$$

Démonstration. En utilisant le fait que $B_{n+1}\mathbf{s}_n = \mathbf{y}_n$ et la formule (5.4), on obtient

$$B_{n+1}\left(\mathbb{I}_d - \frac{|\mathbf{s}_n\rangle\langle \mathbf{y}_n|}{\langle \mathbf{s}_n, \mathbf{y}_n\rangle}\right) = B_{n+1} - \frac{|\mathbf{y}_n\rangle\langle \mathbf{y}_n|}{\langle \mathbf{s}_n, \mathbf{y}_n\rangle} = B_n - \frac{|B_n\mathbf{s}_n\rangle\langle B_n\mathbf{s}_n|}{\langle \mathbf{s}_n, B_n\mathbf{s}_n\rangle}, \quad \text{et} \quad B_{n+1}\frac{|\mathbf{s}_n\rangle\langle \mathbf{s}_n|}{\langle \mathbf{s}_n, \mathbf{y}_n\rangle} = \frac{|\mathbf{y}_n\rangle\langle \mathbf{s}_n|}{\langle \mathbf{s}_n, \mathbf{y}_n\rangle}.$$

3. Ce choix est fait pour que B_n soit bien conditionné. La preuve de ce résultat dépasse le cadre de ce cours.

On a donc

$$B_{n+1}\left[\left(\mathbb{I}_d - \frac{|\mathbf{s}_n\rangle\langle\mathbf{y}_n|}{\langle\mathbf{s}_n,\mathbf{y}_n\rangle}\right)B_n^{-1}\left(\mathbb{I}_d - \frac{|\mathbf{y}_n\rangle\langle\mathbf{s}_n|}{\langle\mathbf{s}_n,\mathbf{y}_n\rangle}\right) + \frac{|\mathbf{s}_n\rangle\langle\mathbf{s}_n|}{\langle\mathbf{s}_n,\mathbf{y}_n\rangle}\right] =$$
$$= \left(\mathbb{I}_d - \frac{|B_n\mathbf{s}_n\rangle\langle\mathbf{s}_n|}{\langle\mathbf{s}_n,B_n\mathbf{s}_n\rangle}\right)\left(\mathbb{I}_d - \frac{|\mathbf{y}_n\rangle\langle\mathbf{s}_n|}{\langle\mathbf{s}_n,\mathbf{y}_n\rangle}\right) + \frac{|\mathbf{y}_n\rangle\langle\mathbf{s}_n|}{\langle\mathbf{s}_n,\mathbf{y}_n\rangle} = \mathbb{I}_d.$$

Cela montre que le terme entre crochet est effectivement l'inverse de B_n. □

Exercice 5.7

Soit $A \in \mathcal{S}_d^{++}$ et $\mathbf{u} \in \mathbb{R}^d$. Montrer que l'inverse de $B := A + |\mathbf{u}\rangle\langle\mathbf{u}|$ est formellement

$$B^{-1} := A^{-1} - \frac{|A^{-1}\mathbf{u}\rangle\langle A^{-1}\mathbf{u}|}{1+\langle\mathbf{u},A\mathbf{u}\rangle} \quad \text{(Formule de Sherman–Morrison).}$$

En déduire que $A + |\mathbf{u}\rangle\langle\mathbf{u}|$ est inversible si et seulement si $1 + \langle\mathbf{u}, A\mathbf{u}\rangle \neq 0$.

Au point $\mathbf{x}_n$, une fois que les matrices B_n et W_n sont construites, et par analogie avec la méthode de Newton, on choisit la direction de descente $\mathbf{h}_n := W_n\nabla F(\mathbf{x}_n)$, puis on recherche un pas qui satisfait le critère de Wolfe (cf Exercice 5.4).

On obtient l'algorithme suivant. On part de $\mathbf{x}_{-1} = \mathbf{0} \in \mathbb{R}^d$ et $\nabla F(\mathbf{x}_{-1}) = \mathbf{0}$ par convention, $\mathbf{x}_0 \in \mathbb{R}^d$ et $W_0 = \mathbb{I}_n$, puis, au point $\mathbf{x}_{n+1}$ (pour faciliter les notations),

$$\begin{aligned}
&\mathbf{s}_n := \mathbf{x}_{n+1} - \mathbf{x}_n, \quad \mathbf{y}_n := \nabla F(\mathbf{x}_{n+1}) - \nabla F(\mathbf{x}_n),\\
&W_{n+1} = \left(\mathbb{I}_d - \frac{|\mathbf{s}_n\rangle\langle\mathbf{y}_n|}{\langle\mathbf{s}_n,\mathbf{y}_n\rangle}\right)W_n\left(\mathbb{I}_d - \frac{|\mathbf{y}_n\rangle\langle\mathbf{s}_n|}{\langle\mathbf{s}_n,\mathbf{y}_n\rangle}\right) + \frac{|\mathbf{s}_n\rangle\langle\mathbf{s}_n|}{\langle\mathbf{s}_n,\mathbf{y}_n\rangle},\\
&\mathbf{h}_{n+1} = -W_{n+1}\nabla F(\mathbf{x}_{n+1}), \quad \mathbf{x}_{n+2} = \mathbf{x}_{n+1} + \tau_{\text{Wolfe}}\,\mathbf{h}_{n+1}.
\end{aligned}$$

Tout comme l'algorithme de la sécante, l'algorithme BFGS exhibe une convergence super-linéaire. De plus, son implémentation est à la fois simple et stable (les erreurs s'atténuent au cours des itérations). Cela en fait une méthode de choix pour optimiser des fonctions. C'est à ce jour l'algorithme le plus usité pour l'optimisation de fonctions non linéaires !

5.2.2 BFGS dans le cas quadratique

Dans cette section, on s'intéresse à l'algorithme de BFGS dans le cas particulier où on veut minimiser $Q(\mathbf{x}) := \frac{1}{2}\langle\mathbf{x}, A\mathbf{x}\rangle - \langle\mathbf{b},\mathbf{x}\rangle$, avec $A \in \mathcal{S}_d^{++}(\mathbb{R})$ (on veut donc résoudre $A\mathbf{x} = \mathbf{b}$). On montre le résultat suivant.

Lemme 5.8 : BGFS et gradient conjugué

L'algorithme BFGS utilisé pour minimiser $Q(\mathbf{x}) := \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x}$ en partant de $\mathbf{x}_0 = \mathbf{0}$, et avec pas optimal ($\tau_{\text{Wolfe}} = \tau_{\text{opt}}$) génère la même suite $(\mathbf{x}_n)$ que le gradient conjugué.

En particulier, on peut appliquer le Théorème 4.9 sur la vitesse de convergence pour le gradient conjugué, et en déduire par exemple que l'algorithme finira en au plus d itérations.

Démonstration. On montre la propriété par récurrence. A l'étape 0, on a bien $\mathbf{x}_0^{(\text{BFGS})} = \mathbf{0} = \mathbf{x}_0^{(\text{GC})}$. Supposons qu'à l'étape $n+1$, on a bien $\mathbf{x}_{k+1}^{\text{BFGS}} = \mathbf{x}_{k+1}^{\text{GC}}$ pour $k \leq n$. En particulier, on a le dictionnaire suivant entre les quantités BFGS et les quantités GC, valide pour tout $0 \leq k \leq n$:

$$\mathbf{s}_k^{(\text{BFGS})} = \mathbf{x}_{k+1} - \mathbf{x}_k = \alpha_k^{(\text{GC})}\mathbf{p}_k^{(\text{GC})}, \qquad \mathbf{y}_k^{(\text{BFGS})} = \nabla Q(\mathbf{x}_{k+1}) - \nabla Q(\mathbf{x}_k) = A\mathbf{x}_{k+1} - A\mathbf{x}_k = \alpha_k^{(\text{GC})}A\mathbf{p}_k^{(\text{GC})},$$

et on peut utiliser les propriétés d'orthogonalité des vecteurs $\mathbf{r}_k$ et $\mathbf{p}_k$ pour $k \leq n$ (voir Lemme 4.4). En réécrivant la formule pour W_{k+1} en fonction de $\mathbf{r}_k$ et $\mathbf{p}_k$, on a

$$W_{k+1} = \left(\mathbb{I}_d - \frac{|\mathbf{p}_k\rangle\langle A\mathbf{p}_k|}{\langle \mathbf{p}_k, A\mathbf{p}_k\rangle}\right) W_k \left(\mathbb{I}_d - \frac{|A\mathbf{p}_k\rangle\langle \mathbf{p}_k|}{\langle \mathbf{p}_k, A\mathbf{p}_k\rangle}\right) + \frac{|\mathbf{p}_k\rangle\langle \mathbf{p}_k|}{\langle \mathbf{p}_k, A\mathbf{p}_k\rangle}.$$

Enfin, on a $\nabla Q(\mathbf{x}_{n+1}) = A\mathbf{x}_{n+1} - \mathbf{b} = -\mathbf{r}_{n+1}^{(\mathrm{GC})}$, de sorte que $\mathbf{h}_{n+1}^{(\mathrm{BFGS})} = W_{n+1}\mathbf{r}_{n+1}$. Comme $\mathbf{r}_{n+1}$ est orthogonal à $\mathbf{p}_k$ pour tout $k \leq n$, on a, par récurrence, et avec $W_0 = \mathbb{I}_d$,

$$\begin{aligned}\mathbf{h}_{n+1}^{(\mathrm{BFGS})} &= \left(\mathbb{I}_d - \frac{|\mathbf{p}_n\rangle\langle A\mathbf{p}_n|}{\mathbf{p}_n^T A\mathbf{p}_n}\right) W_n\mathbf{r}_{n+1} = \cdots = \\ &= \left(\mathbb{I}_d - \frac{|\mathbf{p}_n\rangle\langle A\mathbf{p}_n|}{\langle \mathbf{p}_n, A\mathbf{p}_n\rangle}\right)\left(\mathbb{I}_d - \frac{|\mathbf{p}_{n-1}\rangle\langle A\mathbf{p}_{n-1}|}{\langle \mathbf{p}_{n-1}, A\mathbf{p}_{n-1}\rangle}\right)\cdots\left(\mathbb{I}_d - \frac{|\mathbf{p}_0\rangle\langle A\mathbf{p}_0|}{\langle \mathbf{p}_0, A\mathbf{p}_0\rangle}\right)\mathbf{r}_{n+1}.\end{aligned}$$

Pour tout $k \leq n-1$, $A\mathbf{p}_k \in \mathcal{K}_n(\mathbf{b})$ est orthogonal à $\mathbf{r}_{n+1}$, donc

$$\langle A\mathbf{p}_0, \mathbf{r}_{n+1}\rangle = \langle A\mathbf{p}_1, \mathbf{r}_{n+1}\rangle = \cdots = \langle A\mathbf{p}_{n-1}, \mathbf{r}_{n+1}\rangle = 0.$$

Ainsi, dans les n dernières parenthèse, seule la matrice identité survit, et l'expression se simplifie en

$$\mathbf{h}_{n+1}^{(\mathrm{BFGS})} = \mathbf{r}_{n+1} - \frac{|\mathbf{p}_n\rangle\langle A\mathbf{p}_n|}{\langle \mathbf{p}_n, A\mathbf{p}_n\rangle}\mathbf{r}_{n+1} = \mathbf{r}_{n+1} - \left(\frac{\langle \mathbf{p}_n, A\mathbf{r}_{n+1}\rangle}{\langle \mathbf{p}_n, A\mathbf{p}_n\rangle}\right)\mathbf{p}_n.$$

On reconnaît la formule de $\beta_n^{(\mathrm{GC})}$ donnée en (4.6), et on obtient $\mathbf{h}_{n+1}^{(\mathrm{BFGS})} = \mathbf{p}_{n+1}^{(\mathrm{GC})}$. Les deux algorithmes ont dont la même direction de descente. En faisant une recherche optimale dans cette direction pour les deux algorithmes, on a bien $\mathbf{x}_{n+2}^{(\mathrm{BFGS})} = \mathbf{x}_{n+2}^{(\mathrm{GC})}$. □

Deuxième partie

Optimisation avec contraintes

CHAPITRE 6

INTRODUCTION À L'OPTIMISATION SOUS CONTRAINTES

On s'intéresse maintenant à des problèmes d'optimisation sous contraintes, de la forme

$$\text{argmin}\{F(\mathbf{x}),\ g(\mathbf{x}) = \mathbf{0}\} \qquad \text{(contraintes d'égalité)}$$

ou de la forme

$$\text{argmin}\{F(\mathbf{x}),\ g(\mathbf{x}) \preccurlyeq \mathbf{0}\} \qquad \text{(contraintes d'inégalité)},$$

où on a utilisé la notation $\mathbf{a} \succcurlyeq \mathbf{b}$ pour dire que les coordonnées de $\mathbf{a} \in \mathbb{R}^m$ sont plus grandes ou égales que les coordonnées de $\mathbf{b} \in \mathbb{R}^m$ (par exemple $(2,0) \succcurlyeq (1,0)$).

L'ensemble de *points admissibles* est le sous ensemble des $\mathbf{x} \in \mathbb{R}^d$ qui satisfont les contraintes. Dans la suite, on appellera cet ensemble $K \subset \mathbb{R}^d$. Ces deux problèmes peuvent alors s'écrire sous la forme

$$\text{argmin}\{F(\mathbf{x}),\ \mathbf{x} \in K\}.$$

6.1 Exemples

Commençons par présenter quelques exemples importants. Chacun de ces exemples peut faire l'objet d'un cours spécifique. Nous nous contenterons ici d'exposer des idées générales qui sont à la base des méthodes pour résoudre des problèmes d'optimisation sous contraintes.

Optimisation linéaire (ou programmation linéaire). Ces problèmes sont de la forme

$$\text{argmin}\left\{\mathbf{b}^T\mathbf{x},\ C\mathbf{x} \preccurlyeq \mathbf{g}\right\}, \qquad \text{avec} \quad \mathbf{b} \in \mathbb{R}^n,\ \mathbf{g} \in \mathbb{R}^m \text{ et } \quad C \in \mathcal{M}_{m,n}(\mathbb{R}). \tag{6.1}$$

On veut optimiser une fonction linéaire sous contraintes linéaires. Voici un exemple : *un cuisinier veut faire un plat avec au moins P protéines et G glucides. Il peut mettre x_1 parts de viande, chaque part apportant p_1 protéines et g_1 glucides, et coûtant c_1 euros, et x_2 parts de légumes, chaque part apportant p_2 protéines et g_2 glucides, et coûtant c_2 euros. Il veut optimiser le coût du plat.* De plus, comme il ne peut mettre que des quantités positives de parts, son problème peut se mettre sous la forme

$$\text{argmin}\{x_1c_1 + c_2x_2, \quad x_1p_1 + x_2p_2 \geq P,\ x_1g_1 + x_2g_2 \geq G,\ x_1 \geq 0,\ x_2 \geq 0\}.$$

On retrouve un problème de la forme (6.1) une fois qu'on a posé [1] (on a $n = 2$ et $m = 4$)

$$\mathbf{x} := \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \quad \mathbf{b} := \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}, \quad C := \begin{pmatrix} -p_1 & -p_2 \\ -l_1 & -l_2 \\ -1 & 0 \\ 0 & -1 \end{pmatrix} \quad \text{et} \quad \mathbf{g} := \begin{pmatrix} -P \\ -L \\ 0 \\ 0 \end{pmatrix}.$$

1. La contrainte $\mathbf{a} \succcurlyeq \mathbf{b}$ s'écrit aussi $-\mathbf{a} \preccurlyeq -\mathbf{b}$, et la contrainte $\mathbf{a} = \mathbf{b}$ s'écrit aussi $\mathbf{a} \preccurlyeq \mathbf{b}$ et $-\mathbf{a} \preccurlyeq -\mathbf{b}$.

Optimisation quadratique. Ces problèmes sont de la forme

$$\operatorname{argmin}\left\{\frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x},\ C\mathbf{x} \preccurlyeq \mathbf{g}\right\}. \tag{6.2}$$

On veut optimiser une fonction quadratique sous contraintes linéaires. L'exemple le plus connu est celui de l'optimisation de risque de portefeuille, qui peut-être formulé comme suit : *Picsou veut investir X euros dans N types d'actions boursières. La valeur de l'action i après un an est une variable aléatoire R_i. Si Picsou investit x_i dans l'action i, son portefeuille après un an est la variable aléatoire $R := \sum_{i=1}^N x_i R_i$. Le rendement est l'espérance $r := \mathbb{E}[R] = \sum_i x_i \mathbb{E}[R_i]$, et le risque est la variance $\sigma^2(R) := \mathbb{E}[(R - \mathbb{E}(R))^2]$. Picsou cherche à minimiser le risque tout en s'assurant un rendement d'au moins r_0.* Pour formuler mathématiquement le problème, on remarque que

$$\sigma^2(R) = \mathbb{E}\left[\left(\sum_{i=1}^N x_i(R_i - \mathbb{E}(R_i))\right)^2\right] = \sum_{i=1}^N\sum_{j=1}^N x_i x_j \mathbb{E}\left[(R_i - \mathbb{E}(R_i))(R_j - \mathbb{E}(R_j))\right] = \mathbf{x}^T A\mathbf{x},$$

où $A \in \mathcal{M}_N(\mathbb{R})$ est la matrice symétrique dont les composantes sont $A_{ij} := \mathbb{E}\left[(R_i - \mathbb{E}(R_i))(R_j - \mathbb{E}(R_j))\right]$ (matrice de covariance). En posant $\mathbf{r} := (\mathbb{E}(\mathbf{R}_1), \cdots, \mathbb{E}(\mathbf{R}_n))^T$ et $\mathbf{1} := (1, \cdots, 1)^T$, le problème est de la forme

$$\operatorname{argmin}\left\{\frac{1}{2}\mathbf{x}^T A\mathbf{x}, \quad \mathbf{1}^T\mathbf{x} \le X,\ \mathbf{r}^T\mathbf{x} \ge r_0,\ \mathbf{x} \succcurlyeq \mathbf{0}\right\}.$$

On peut aussi avoir le problème dual où Picsou cherche le meilleur rendement, mais où le risque est borné par σ_0^2. Dans ce cas, le problème est plutôt de la forme (optimisation linéaire sous contrainte quadratique)

$$\operatorname{argmax}\left\{\mathbf{r}^T\mathbf{x}, \quad \mathbf{1}^T\mathbf{x} \le X,\ \mathbf{x}^T A\mathbf{x} \le \sigma_0^2,\ \mathbf{x} \succcurlyeq \mathbf{0}\right\}.$$

Problèmes inverses. Dans les problèmes inverses, on a accès a des informations partielles sur des données $\mathbf{x} \in \mathbb{R}^n$, de type $C\mathbf{x} = \mathbf{g} \in \mathbb{R}^m$, où $C \in \mathcal{M}_{m,n}(\mathbb{R})$ et $\mathbf{g} \in \mathbb{R}^m$ sont connus, et on cherche $\mathbf{x}$. En général, on a $m \ll n$ (beaucoup moins d'informations que de données à trouver), et C n'est pas inversible. On suppose cependant que $J(\mathbf{x})$ doit être petit, où J est une certaine fonctionnelle connue. Le problème devient de la forme

$$\operatorname{argmin}\left\{J(\mathbf{x}), \quad C\mathbf{x} = \mathbf{g}\right\}.$$

Un exemple est celui de l'imagerie médicale : *On cherche la densité interne $\mathbf{x} \in \mathbb{R}^n$ d'un objet (n est le nombre de «pixels» de l'objet, chaque pixel a une densité «constante»), et on mesure l'intégrale de la densité le long de m lignes par des rayons-X. Comment retrouver $\mathbf{x}$?* Les mesures sont de la forme $C\mathbf{x} = \mathbf{g}$, C est une matrice qui indique quels pixels sont sur quelles lignes, et où $\mathbf{g}$ contient le résultat des intégrales. En pratique, on a des information *a priori* sur une image médicale. Par exemple, on s'attend à avoir peu de *frontières* entre les zones de densité différentes (les organes sont bien séparés). On peut donc prendre pour $J(\mathbf{x})$ une fonctionnelle qui mesure le nombre de frontières.

Optimisation avec contraintes quadratiques d'égalité. Mentionnons enfin les problèmes d'optimisation où $\mathbf{x}$ doit vivre sur une surface, (dans notre exemple, une ellipsoïde ou une sphère). Ces problèmes sont de la forme

$$\operatorname{argmin}\left\{f(\mathbf{x}), \quad \mathbf{x}^T K\mathbf{x} = 1\right\}.$$

Contrairement à tous les exemples précédents, l'ensemble de minimisation **n'est pas convexe**. Ce problème apparait notamment pour calculer les valeurs propres d'une matrice : la plus petite valeur propre de $A \in S_d(\mathbb{R})$ est donnée par le problème d'optimisation

$$\lambda_1(A) := \operatorname{argmin}\left\{\mathbf{x}^T A\mathbf{x}, \quad \mathbf{x}^T\mathbf{x} = 1\right\}.$$

6.2 Contraintes d'inégalités linéaires, et représentation graphique

Dans tous les exemples précédents, sauf le dernier, les contraintes sont des inégalités linéaires : l'ensemble de minimisation est de la forme

$$K := \left\{ \mathbf{x} \in \mathbb{R}^d, \quad C\mathbf{x} \preccurlyeq \mathbf{g} \right\},$$

avec $C \in \mathcal{M}_{m,d}(\mathbb{R})$ et $\mathbf{g} \in \mathbb{R}^m$. Le nombre m de lignes de C est le nombre de contraintes. Il peut-être grand en pratique. Si $\mathbf{c}_1, \cdots, \mathbf{c}_m$ sont les lignes de C, alors les m contraintes peuvent s'écrire $\mathbf{c}_i\mathbf{x} - g_i \leq 0$ pour tout $1 \leq i \leq m$.

L'ensemble des $\mathbf{x} \in \mathbb{R}^n$ tel que $\mathbf{c}_i\mathbf{x} = g_i$ est un hyperplan affine de $\mathbb{R}^d$. Cet hyperplan sépare $\mathbb{R}^d$ en deux demi-espaces, et l'ensemble des $\mathbf{x} \in \mathbb{R}^d$ vérifiant $\mathbf{c}_i\mathbf{x} - g_i \leq 0$ est l'un de ces demi-espaces. Ainsi, l'ensemble K est une intersection de m demi-espaces. On appellera un tel ensemble un *polyèdre* (pas forcément borné).

Exercice 6.1

Montrer que K est un ensemble convexe fermé.

Par exemple, sur la Figure 6.1, on regarde l'ensemble des $\mathbf{x} = (x_1, x_2) \in \mathbb{R}^2$ vérifiant $x_1 \geq 0$, $x_2 \geq 0$, $x_1 + x_2 \leq 1$ et $4x_2 + x_1 \leq 2$, ou encore $C\mathbf{x} \preccurlyeq \mathbf{g}$ avec

$$C := \begin{pmatrix} -1 & 0 \\ 0 & -1 \\ 1 & 1 \\ 1 & 4 \end{pmatrix} \quad \text{et} \quad \mathbf{g} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 2 \end{pmatrix}. \tag{6.3}$$

On obtient la partie rouge de la Figure 6.1.

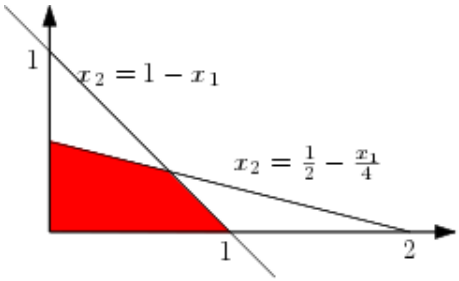

Figure 6.1 – Exemple de polyèdres en 2 dimensions.

Résoudre un problème de minimisation sous la contrainte $\mathbf{x} \in K$, c'est trouver le point du polyèdre K qui minimise une fonction F. Il est donc intéressant de superposer ce polyèdre et les courbes de niveau de F. Dans la figure 6.2, le point noir est la solution graphique des problèmes de minimisation $\operatorname{argmin}\{F(\mathbf{x}),\ \mathbf{x} \in K\}$, pour différentes fonctions F. Dans l'image de gauche, le minimiseur se trouve sur un coin, dans l'image du centre, il est sur un bord, et dans l'image de droite, le vrai minimiseur de F (celui sans contrainte) appartient déjà à K, et est dans l'intérieur de K.

Exercice 6.2

Soit C et $\mathbf{g}$ définies en (6.3), et soit $\mathbf{b} := (1, -1)^T$. Montrer (graphiquement) que le point $\mathbf{x}^* := (1, 0)$ est la solution du problème d'optimisation linéaire

$$\operatorname{argmax} \left\{ \mathbf{c}^T\mathbf{x},\ C\mathbf{x} \preccurlyeq \mathbf{g} \right\}.$$

On pourra tracer les courbes de niveau de $\mathbf{x} \mapsto \mathbf{c}^T\mathbf{x}$.

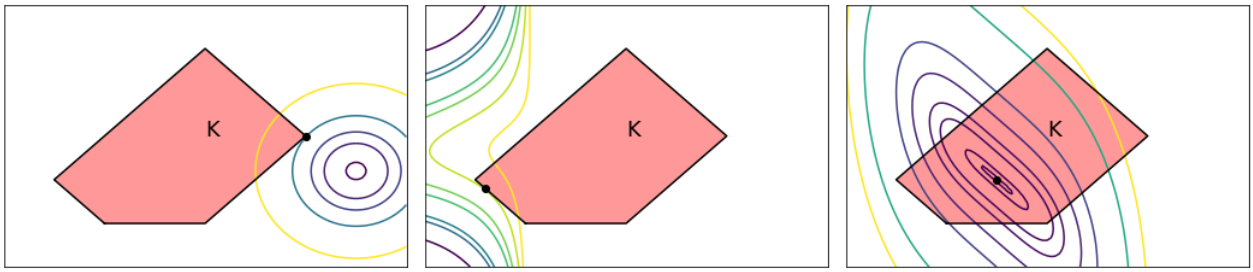

Figure 6.2 – Courbes de niveau de différentes fonctions F, avec un polyèdre K (rouge). Le point noir indique le minimiseur $\operatorname{argmin}\{F(\mathbf{x}), \mathbf{x} \in K\}$.

CHAPITRE 7

MÉTHODES PAR PROJECTION

Dans ce chapitre, nous présentons les idées principales des méthodes par projection.

7.1 Projection sur un ensemble convexe fermé

On rappelle la définition suivante. Soit $K \subset \mathbb{R}^d$ un ensemble convexe fermé. On appelle *projection* sur K l'application $P_K : \mathbb{R}^d \to X$ définie par

$$P_K(\mathbf{x}) := \operatorname{argmin}\{\|\mathbf{k} - \mathbf{x}\|,\ \mathbf{k} \in K\}. \tag{7.1}$$

Cette notion a été déjà vue dans le cas où K était un sous-espace vectoriel (on parlait dans ce cas de projection orthogonale). Ici, on étend la définition à tous les ensembles fermés convexes.

7.1.1 Premières propriétés

Lemme 7.1 : Projection sur un convexe fermé

Si $K \neq \emptyset$, alors l'application P_K est bien définie, et pour tout $\mathbf{k} \in K$, on a

$$\langle \mathbf{x} - P_K(\mathbf{x}), \mathbf{k} - P_K(\mathbf{x}) \rangle \leq 0. \tag{7.2}$$

Démonstration. **Existence d'un minimiseur.** Soit $\mathbf{k}_n \in K$ une suite minimisante pour le problème $\inf\{\|\mathbf{k} - \mathbf{x}\|,\ \mathbf{k} \in K\}$. La suite $(\|\mathbf{x} - \mathbf{k}_n\|)_n$ est bornée, donc la suite $(\mathbf{k}_n)_n$ est aussi bornée. Cette suite admet une sous-suite qui converge vers un certain $\mathbf{k}^* \in K$ (car K est fermé). Par continuité de la norme, et passage à la limite, on a $\|\mathbf{x} - \mathbf{k}^*\| = \inf_K\{\|\mathbf{x} - \mathbf{k}\|\}$, et $\mathbf{k}^*$ est un minimiseur.

Unicité. Supposons par l'absurde qu'il existe $\mathbf{k}_1, \mathbf{k}_2 \in E$, $\mathbf{k}_1 \neq \mathbf{k}_2$ deux minimiseurs de (7.1), et soit $m = \|\mathbf{x} - \mathbf{k}_1\| = \|\mathbf{x} - \mathbf{k}_2\|$ ce minimum. Comme K est convexe, le point $\mathbf{k} := \frac{1}{2}(\mathbf{k}_1 + \mathbf{k}_2)$ appartient aussi à K. On rappelle l'identité du parallélogramme :

$$\forall \mathbf{a}, \mathbf{b} \in \mathbb{R}^d, \quad \|\mathbf{a}\|^2 + \|\mathbf{b}\|^2 = \frac{1}{2}\|\mathbf{a} - \mathbf{b}\|^2 + \frac{1}{2}\|\mathbf{a} + \mathbf{b}\|^2.$$

Avec $\mathbf{a} = \mathbf{x} - \frac{1}{2}(\mathbf{k}_1 + \mathbf{k}_2)$ et $\mathbf{b} := \frac{1}{2}(\mathbf{k}_1 - \mathbf{k}_2)$, on obtient

$$\|\mathbf{x} - \mathbf{k}\|^2 + \frac{1}{4}\|\mathbf{k}_1 - \mathbf{k}_2\|^2 = \frac{1}{2}\|\mathbf{x} - \mathbf{k}_1\|^2 + \frac{1}{2}\|\mathbf{x} - \mathbf{k}_2\|^2, \quad \text{et donc} \quad \|\mathbf{x} - \mathbf{k}\| < m,$$

ce qui contredit la minimalité de m. Ainsi, l'application P_K est bien définie.

Preuve de l'inégalité (7.2)**.** Soit $\mathbf{k}^* := P_K(\mathbf{x})$, et soit $\mathbf{k} \in K$ un autre point quelconque de K. Pour tout $t \in [0,1]$, le point $\mathbf{k}(t) := (1-t)\mathbf{k}^* + t\mathbf{k} = \mathbf{x}^* + t(\mathbf{k} - \mathbf{k}^*)$ appartient à K (convexité de K). En particulier, on a, par unicité du minimum,

$$\forall t \in (0,1], \quad \|\mathbf{x}-\mathbf{k}^*\|^2 < \|\mathbf{x}-\mathbf{k}(t)\|^2 = \|(\mathbf{x}-\mathbf{k}^*)-t(\mathbf{k}-\mathbf{k}^*)\|^2 = \|\mathbf{x}-\mathbf{k}^*\|^2 - 2t\langle \mathbf{x}-\mathbf{k}^*, \mathbf{k}-\mathbf{k}^*\rangle + t^2\|\mathbf{k}-\mathbf{k}^*\|^2,$$

ou encore $2t\langle \mathbf{x}-\mathbf{k}^*, \mathbf{k}-\mathbf{k}^*\rangle + o(t) < 0$. En divisant par $t > 0$, et en prenant la limite $t \to 0^+$, on obtient $\langle \mathbf{x}-\mathbf{k}^*, \mathbf{k}-\mathbf{k}^*\rangle \leq 0$, ce qui démontre le résultat. □

On a évidemment $P_K\big|_K = \mathrm{id}_K$, donc $P_K \circ P_K = P_K$. La proposition suivante montre que P_K est 1-Lipschitzienne.

Lemme 7.2 : La projection est une contraction

Si $K \neq \emptyset$, alors pour tout $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$, on a $\|P_K(\mathbf{x}) - P_K(\mathbf{y})\| \leq \|\mathbf{x} - \mathbf{y}\|$.

Démonstration. D'après le Lemme 7.1 appliqué en $\mathbf{x}$ et en $\mathbf{y}$, on a

$$\langle \mathbf{x} - P_K(\mathbf{x}), P_K(\mathbf{y}) - P_K(\mathbf{x})\rangle \leq 0 \quad \text{et} \quad \langle P_K(\mathbf{y}) - \mathbf{y}, P_K(\mathbf{y}) - P_K(\mathbf{x})\rangle \leq 0.$$

En sommant, on trouve

$$\langle \mathbf{x}-\mathbf{y}-(P_K(\mathbf{x}) - P_K(\mathbf{y})), P_K(\mathbf{y})-P_K(\mathbf{x})\rangle \leq 0, \text{ ou encore } \|P_K(\mathbf{x})-P_K(\mathbf{y})\|^2 \leq \langle \mathbf{x}-\mathbf{y}, P_K(\mathbf{x})-P_K(\mathbf{y})\rangle.$$

Avec l'inégalité de Cauchy-Schwarz, on obtient

$$\|P_K(\mathbf{x}) - P_K(\mathbf{y})\|^2 \leq \|\mathbf{x} - \mathbf{y}\| \cdot \|P_K(\mathbf{x}) - P_K(\mathbf{y})\|,$$

et la preuve suit. □

Attention, contrairement au cas où K est un sous-espace vectoriel, P_K n'est pas une application linéaire dans le cas général. On a $P_K(\mathbf{a}+\mathbf{b}) \neq P_K(\mathbf{a}) + P_K(\mathbf{b})$ et $P_K(t\mathbf{x}) \neq tP_K(\mathbf{x})$.

7.2 Méthode du gradient projeté

7.2.1 Algorithme

Nous décrivons maintenant la méthode du *gradient projeté*. Comme son nom l'indique, on cherche à résoudre un problème sous contraintes

$$\mathrm{argmin}\{F(\mathbf{x}),\ \mathbf{x} \in K\} \quad \text{avec } K \text{ un fermé convexe}$$

par une méthode de type descente de gradient. Le problème est que le gradient de F peut nous faire sortir de K. L'idée est donc de considérer *le projeté* du gradient. Les itérations sont de la forme (ici pour un gradient projeté à pas constant)

$$\mathbf{x}_{n+1} = P_K\left(\widetilde{\mathbf{x}_{n+1}}\right), \quad \text{avec} \quad \widetilde{\mathbf{x}_{n+1}} := \mathbf{x}_n - \tau\nabla F(\mathbf{x}_n).$$

Ainsi, on construit l'itération donnée par la méthode du gradient usuel $\widetilde{\mathbf{x}_{n+1}}$, puis on projette le résultat sur K. Par construction, la suite $(\mathbf{x}_n)$ est dans K. Pour mettre cette suite sous la forme d'un algorithme de descente, on écrit plutôt

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \mathbf{h}_n \quad \text{avec} \quad \mathbf{h}_n := P_K\left(\mathbf{x}_n - \tau\nabla F(\mathbf{x}_n)\right) - \mathbf{x}_n. \tag{7.3}$$

On fera attention qu'ici, on ne peut pas considérer n'importe quelle direction de descente de F, et qu'on considère exclusivement le gradient. Cela vient du lemme suivant.

Lemme 7.3

Soit $\mathbf{x} \in K$ et $\tau > 0$ tel que $P_K(\mathbf{x} - \tau\nabla F(\mathbf{x})) \neq \mathbf{x}$. Alors le vecteur

$$\mathbf{h}(\tau) := P_K(\mathbf{x} - \tau\nabla F(\mathbf{x})) - \mathbf{x}$$

est une direction de descente de F en $\mathbf{x}$.

Démonstration. Pour montrer que $\mathbf{h}(\tau)$ est une direction de descente, il suffit de démontrer que $\langle \nabla F(\mathbf{x}), \mathbf{h}(\tau)\rangle < 0$. En utilisant (7.2) et le fait que $\mathbf{x} \in K$, on a

$$\langle [\mathbf{x} - \tau\nabla F(\mathbf{x})] - P_K\left[\mathbf{x} - \tau\nabla F(\mathbf{x})\right], \mathbf{x} - P_K\left[\mathbf{x} - \tau\nabla F(\mathbf{x})\right]\rangle \leq 0,$$

ce qui s'écrit aussi

$$\tau\langle \nabla F(\mathbf{x}), \underbrace{\mathbf{x} - P_K\left[\mathbf{x} - \tau\nabla F(\mathbf{x})\right]}_{-\mathbf{h}(\tau)}\rangle \geq \|\mathbf{x} - P_K\left[\mathbf{x} - \tau\nabla F(\mathbf{x})\right]\|^2 \quad (> 0).$$

□

D'après le Lemme 7.3, le minimum $\mathbf{x}^*$ doit vérifier la condition

$$\forall \tau > 0, \quad \mathbf{x}^* = P_K(\mathbf{x}^* - \tau\nabla F(\mathbf{x}^*)).$$

Cela nous donne un critère pour stopper notre algorithme : on vérifie si $\mathbf{x}_{n+1} \approx \mathbf{x}_n$. On obtient l'algorithme suivant (ici, on suppose qu'on a déjà codé une fonction de projection P_K -voir Section suivante).

Code 7.1 – Algorithme du gradient projeté à pas constant

```
def gradientProj(dF, PK, x0, tau, tol=1e-6, Niter=1000):
    xn, L = PK(x0), []
    for n in range(Niter):
        xnp1 = PK(xn-tau*dF(xn))
        if norm(xnp1 - xn) < tol:
            return xnp1, L
        L.append(xn)
        xn = xnp1
    print("Problème, l'algorithme n'a pas convergé après", Niter, "itérations")
```

7.2.2 Analyse dans le cas quadratique

Dans le cas où F est une fonction quadratique, on a un résultat théorique (à comparer avec le Lemme 3.12).

Lemme 7.4 : Convergence du gradient projeté

Soit $A \in \mathcal{S}_d^{++}(\mathbb{R})$, $\mathbf{b} \in \mathbb{R}^d$, et $Q(\mathbf{x}) := \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T$, et soit K un fermé convexe non vide. Le problème $\operatorname{argmin}\{Q(\mathbf{x}),\ \mathbf{x} \in K\}$ a une unique solution $\mathbf{x}^*$.
Si $0 < \lambda_1 \leq \cdots \leq \lambda_d$ sont les valeurs propres de A rangées dans l'ordre croissant, alors, pour tout $\tau > 0$ tel que $\tau\lambda_d < 2$, et pour tout $\mathbf{x}_0 \in K$, la suite définie par (7.2.1) converge vers $\mathbf{x}^*$ à taux au moins $\max\{|1 - \tau\lambda_1|, |1 - \tau\lambda_d|\}$.

Démonstration. Le fait qu'il existe un unique minimum vient du fait qu'on minimise une fonction strictement convexe sur un ensemble fermé convexe. Par ailleurs, comme P_K est une contraction, et que $\nabla Q(\mathbf{x}) = A\mathbf{x} - \mathbf{b}$, on a, d'après (7.2.1),

$$\begin{aligned}\|\mathbf{x}_{n+1} - \mathbf{x}_n\| &= \|P_K\left(\mathbf{x}_n + \tau\nabla Q(\mathbf{x}_n)\right) - P_K\left(\mathbf{x}_{n-1} + \tau\nabla Q(\mathbf{x}_{n-1})\right)\| \\ &\leq \|\mathbf{x}_n - \mathbf{x}_{n-1} - \tau\left[\nabla Q(\mathbf{x}_n) - \nabla Q(\mathbf{x}_{n-1})\right]\| \\ &\leq \|\left(\mathbb{I} - \tau A\right)\left(\mathbf{x}_n - \mathbf{x}_{n-1}\right)\| \leq \|\mathbb{I} - \tau A\|_{\text{op}} \cdot \|\mathbf{x}_n - \mathbf{x}_{n-1}\|\,.\end{aligned}$$

On a $\alpha := \|\mathbb{I} - \tau A\|_{\text{op}} = \max\{|1 - \tau\lambda_1|, |1 - \tau\lambda_d|\}$, et si $\tau\lambda_d < 2$, alors $\alpha < 1$ (cf preuve du Lemme 3.12). Le résultat est alors une conséquence du Lemme 1.6. □

7.3 Calcul de projections

Nous avons décrit l'algorithme du gradient projeté, et analysé théoriquement son efficacité dans le cas quadratique. Cependant, pour son implémentation pratique, il nous faut encore expliquer comment calculer numériquement la projection P_K. Ce problème est plus complexe qu'il n'en a l'air, car P_K est défini par (7.1), à savoir

$$P_K(\mathbf{b}) := \text{argmin}\{\|\mathbf{b} - \mathbf{x}\|,\ \mathbf{x} \in K\},$$

qui est aussi un problème de minimisation sous contraintes.

Ainsi, pour résoudre un problème de minimisation sous contraintes, il faut déjà savoir résoudre un problème de minimisation sous contrainte (la projection)...

Heureusement, dans un grand nombre de cas, cette projection est facile à calculer. Nous expliquons dans ce chapitre comment calculer des projections, pour différents ensembles convexe fermé K. On rappelle que minimiser $\mathbf{x} \mapsto \|\mathbf{x} - \mathbf{b}\|$, c'est aussi minimiser $\mathbf{x} \mapsto \frac{1}{2}\|\mathbf{x} - \mathbf{b}\|^2 - \frac{1}{2}\|\mathbf{b}\|^2 = \frac{1}{2}\|\mathbf{x}\|^2 - \langle \mathbf{b}, \mathbf{x}\rangle$, qui est une fonction quadratique en $\mathbf{x}$. Dans la suite, on explique comment minimiser des fonctionnelles quadratiques générales, de la forme $\frac{1}{2}\langle \mathbf{x}, A\mathbf{x}\rangle - \langle \mathbf{b}, \mathbf{x}\rangle$. Dans le cas de la projection, on a $A = \mathbb{I}_d$.

7.3.1 Optimisation quadratique sous contraintes d'égalités linéaires

On commence par étudier ce qui se passe lorsque K est un espace affine, définit par $K := C\mathbf{x} = \mathbf{g}$, avec $C \in \mathcal{M}_{m,d}$ et $\mathbf{g} \in \mathbb{R}^m$. On cherche à résoudre

$$\text{argmin}\left\{\frac{1}{2}\langle \mathbf{x}, A\mathbf{x}\rangle - \langle \mathbf{b}, \mathbf{x}\rangle, \quad C\mathbf{x} = \mathbf{g}\right\}. \tag{7.4}$$

Afin de résoudre ce problème, on s'intéresse aux propriétés des points critiques. Dans le cas sans contraintes, on sait que si $\mathbf{x}^*$ est un minimiseur de $F : \mathbb{R}^d \to \mathbb{R}$, alors $\nabla F(\mathbf{x}^*) = \mathbf{0}$. Dans le cas avec contraintes, on a le résultat suivant.

Lemme 7.5 : Équations d'Euler-Lagrange

Soit $F : \mathbb{R}^d \to \mathbb{R}$ une fonction de classe C^1, et soit $C \in \mathcal{M}_{m\times d}$ et $\mathbf{g} \in \mathbb{R}^m$. Si $\mathbf{x}^*$ un minimiseur de $\{F(\mathbf{x}),\ C\mathbf{x} = \mathbf{g}\}$, alors $\nabla F(\mathbf{x}^*) \in \text{Im}\, C^T$: il existe $\boldsymbol{\lambda}^* \in \mathbb{R}^m$ tel que $\nabla F(\mathbf{x}^*) = C^T\boldsymbol{\lambda}^*$.

Ce sont les équations d'Euler-Lagrange dans le cas de contraintes linéaires. Le vecteur $\boldsymbol{\lambda}^*$ est un vecteur de taille m (autant que le nombre de contraintes), et s'appelle le *multiplicateur de Lagrange*.

Démonstration. Soit $K := \left\{\mathbf{x} \in \mathbb{R}^d,\ C\mathbf{x} = \mathbf{g}\right\}$. Si $\mathbf{x}^*$ est un minimiseur de F sur K, alors $C\mathbf{x}^* = \mathbf{g}$, et, pour tout $\mathbf{x} \in K$, $F(\mathbf{x}) \geq F(\mathbf{x}^*)$. De plus, pour tout $\mathbf{x} \in K$ et pour tout $t \in \mathbb{R}$, le point $\mathbf{x}(t) := (1-t)\mathbf{x}^* + t\mathbf{x}$ vérifie $C\mathbf{x}(t) = \mathbf{g}$, donc appartient à K. On obtient

$$\forall t \in \mathbb{R}, \quad F(\mathbf{x}(t)) \geq F(\mathbf{x}^*) \qquad = F(\mathbf{x}(t=0)).$$

Le point $t = 0$ est un minimum de $t \mapsto F(\mathbf{x}(t))$ sur $\mathbb{R}$. En dérivant en $t = 0$, on obtient

$$\forall \mathbf{x} \in \mathbb{R}^d \text{ tel que } C\mathbf{x} = \mathbf{g}, \quad \langle \nabla F(\mathbf{x}^*), \mathbf{x} - \mathbf{x}^* \rangle = 0.$$

En faisant le changement de variable $\mathbf{h} := \mathbf{x} - \mathbf{x}^*$, cette identité s'écrit aussi

$$\forall \mathbf{h} \in \operatorname{Ker} C, \quad \langle \nabla F(\mathbf{x}^*), \mathbf{h} \rangle = 0, \quad \text{ou encore} \quad \nabla F(\mathbf{x}^*) \in (\operatorname{Ker} C)^\perp . \tag{7.5}$$

La preuve du Lemme 7.5 suit du résultat suivant d'algèbre linéaire. □

Lemme 7.6

Pour tout $C \in \mathcal{M}_{m,d}(\mathbb{R})$, on a $(\operatorname{Ker} C)^\perp = \operatorname{Im} C^T$.

Démonstration du Lemme 7.6. On a

$$(\forall \mathbf{x} \in \operatorname{Ker} C,\ \forall \mathbf{y} \in \mathbb{R}^m, \quad \langle \mathbf{y}, C\mathbf{x} \rangle_{\mathbb{R}^m} = 0) \quad \text{et donc} \quad \left(\forall \mathbf{y} \in \mathbb{R}^m,\ \forall \mathbf{x} \in \operatorname{Ker} C, \quad \langle C^T\mathbf{y}, \mathbf{x} \rangle_{\mathbb{R}^d} = 0\right),$$

ce qui implique que $\operatorname{Im} C^T \subset (\operatorname{Ker} C)^\perp$. De plus, par le théorème du rang et le fait que $\operatorname{rg} C = \operatorname{rg} C^T$, on a $\dim (\operatorname{Ker} C)^T = d - \dim \operatorname{Ker} C = \operatorname{rg} C = \dim \operatorname{Im} C^T$. Ainsi, ces deux espaces ont la même dimension, et donc sont égaux. □

Dans le cas où F est la fonction quadratique $F(\mathbf{x}) := \frac{1}{2}\langle \mathbf{x}, A\mathbf{x} \rangle - \langle \mathbf{b}, \mathbf{x} \rangle$, on a $\nabla F(\mathbf{x}) = A\mathbf{x} - \mathbf{b}$. Ainsi, si $\mathbf{x}^* \in \mathbb{R}^d$ est une solution de (7.4), alors, d'après le Lemme 7.5, on a

$$\exists \boldsymbol{\lambda}^* \in \mathbb{R}^d, \quad A\mathbf{x}^* + C^T\boldsymbol{\lambda}^* = \mathbf{b} \quad \text{et} \quad C\mathbf{x}^* = \mathbf{g} \qquad \text{(Euler-Lagrange)}.$$

On réécrit ces équations sous la forme

$$\begin{pmatrix} A & C^T \\ C & 0 \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ \mathbf{g} \end{pmatrix}. \tag{7.6}$$

L'ensemble des couples $(\mathbf{x}, \boldsymbol{\lambda}) \in \mathbb{R}^d \times \mathbb{R}^m$ qui vérifie (7.6) est l'ensemble des *points critiques*. Ce système est posé en dimension $(m + d)$: il faut chercher le minimiseur $\mathbf{x} \in \mathbb{R}^d$, et les multiplicateurs de Lagrange associés $\boldsymbol{\lambda} \in \mathbb{R}^m$.

On a montré qu'un minimiseur était en particulier un point critique (une solution de (7.4) est en particulier une solution de (7.6)). La réciproque n'est pas toujours vraie (les maximiseurs sont aussi des points critiques). Le lemme suivant donne un cas où les deux problèmes sont équivalents.

Lemme 7.7

Si $A \in \mathcal{S}_d^{++}(\mathbb{R})$ et si $C \in \mathcal{M}_{d,m}(\mathbb{R})$ est surjectif, alors les problèmes (7.4) et (7.6) ont une unique solution. En particulier, ces solutions sont égales.

Démonstration. L'ensemble $K := \left\{\mathbf{x} \in \mathbb{R}^d,\ C\mathbf{x} = \mathbf{g}\right\}$ est un ensemble convexe non vide d'après la surjectivité de C. De plus, F est convexe et coercive sur K. Elle admet donc un unique minimum sur K. Cela montre que le problème (7.4) a une unique solution.

Pour montrer que (7.6) a une unique solution, il suffit de montrer que la matrice carrée $\mathcal{M} := \begin{pmatrix} A & C^T \\ C & 0 \end{pmatrix}$ est injective (donc inversible). Soit $(\mathbf{x}_0, \boldsymbol{\lambda}_0) \in \mathbb{R}^d \times \mathbb{R}^m \in \operatorname{Ker} \mathcal{M}$. On a

$$A\mathbf{x}_0 + C^T\boldsymbol{\lambda}_0 = \mathbf{0} \quad \text{et} \quad C\mathbf{x}_0 = \mathbf{0}.$$

Comme A est inversible, on a, avec la première équation, $\mathbf{x}_0 = -A^{-1}C^T\boldsymbol{\lambda}_0$. La deuxième équation donne alors $\left(CA^{-1}C^T\right)\boldsymbol{\lambda}_0 = \mathbf{0}$. Par stricte positivité de A^{-1}, on obtient

$$\langle \boldsymbol{\lambda}_0, \left(CA^{-1}C^T\right)\boldsymbol{\lambda}_0\rangle = 0, \quad \text{ou encore} \quad \langle C^T\boldsymbol{\lambda}_0, A^{-1}\left(C^T\boldsymbol{\lambda}_0\right)\rangle = 0 \quad \text{et donc} \quad C^T\boldsymbol{\lambda}_0 = 0.$$

Ainsi, $\boldsymbol{\lambda}_0 \in \operatorname{Ker} C^T$. Comme C est surjectif, on a $\operatorname{Im} C = \mathbb{R}^d$. D'après le Lemme 7.6 appliqué à C^T, cela implique que $\operatorname{Ker} C = \{\mathbf{0}\}$. Ainsi, $\boldsymbol{\lambda}_0 = \mathbf{0}$, et de même $\mathbf{x}_0 = -A^{-1}C^T\boldsymbol{\lambda}_0 = \mathbf{0}$. □

Attention, la matrice $\mathcal{M}$ n'est pas définie positive en général (prendre $A = a \in \mathbb{R}$ et $C = 1 \in \mathbb{R}$). Cependant, d'après le Lemme 7.7, elle est inversible, et on peut résoudre le système (7.6) avec par exemple un pivot de Gauss.

Pour conclure cette section, en prenant $A = \mathbb{I}_d$, on a montré qu'on pouvait résoudre le problème de projection $\mathbf{x} := P_K(\mathbf{b})$ où K est l'espace affine $K := \{\mathbf{x} \in \mathbb{R}^d,\ C\mathbf{x} = \mathbf{g}\}$, en résolvant le système linéaire

$$\begin{pmatrix} \mathbb{I}_d & C^T \\ C & 0 \end{pmatrix}\begin{pmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ \mathbf{g} \end{pmatrix}.$$

7.3.2 Optimisation quadratique sous contraintes d'inégalités linéaires

On regarde maintenant le cas où les contraintes sont de la forme $C\mathbf{x} \preccurlyeq \mathbf{g}$, avec $C \in \mathcal{M}_{m,d}(\mathbb{R})$ et $\mathbf{g} \in \mathbb{R}^m$. On regarde donc le problème

$$\operatorname{argmin}\{F(\mathbf{x}), \quad C\mathbf{x} \preccurlyeq \mathbf{g}\}. \tag{7.7}$$

Comme nous l'avons déjà vu, l'ensemble $K := \{C\mathbf{x} \preccurlyeq \mathbf{g}\}$ est un polyèdre. Dans le cas de la projection, on a $F(\mathbf{x}) = \frac{1}{2}\|\mathbf{x}\|^2 - \langle \mathbf{b}, \mathbf{x}\rangle$, mais nous donnons des résultats plus généraux.

Les conditions KKT

Comme dans le cas précédent, on a une notion de *points critiques*, mais la définition est plus complexe. Les condition définissant ces points critiques s'appellent les *conditions KKT*[1]. Dans la suite, afin de simplifier les notations, on introduit la notation $\mathbf{a} \odot \mathbf{b}$ pour la multiplication terme à terme (multiplication de Hadamard) : si $\mathbf{a}, \mathbf{b} \in \mathbb{R}^d$, alors $\mathbf{a}\odot\mathbf{b} \in \mathbb{R}^d$ a pour composante $(\mathbf{a}\odot\mathbf{b})_i := a_i b_i$.

Lemme 7.8 : Conditions KKT

Soit $F : \mathbb{R}^d \to \mathbb{R}$ une fonction de classe C^1, et soit $C \in \mathcal{M}_{m\times d}$ et $\mathbf{g} \in \mathbb{R}^m$. Si $\mathbf{x}^*$ un minimiseur de $\{F(\mathbf{x}),\ C\mathbf{x} \preccurlyeq \mathbf{g}\}$, alors

$$\exists \boldsymbol{\lambda}^* \in \mathbb{R}^m,\ \boldsymbol{\lambda}^* \preccurlyeq \mathbf{0}, \quad \nabla F(\mathbf{x}^*) = C^T\boldsymbol{\lambda}^*, \quad C\mathbf{x}^* \preccurlyeq \mathbf{g} \quad \text{et} \quad \boldsymbol{\lambda}^* \odot (C\mathbf{x}^* - \mathbf{g}) = \mathbf{0}. \tag{7.8}$$

La dernière équation s'écrit aussi : pour tout $1 \le i \le m$, on a soit $\lambda_i^* = 0$, soit $\mathbf{c}_i\mathbf{x}^* = g_i$, où $\mathbf{c}_i$ est la i-ème ligne de C. Une notion importante qui intervient dans la preuve est celle de *contraintes saturées*. On dit que la i-ème contrainte est saturée si $\mathbf{c}_i\mathbf{x}^* = g_i$. Si la i-ème contrainte n'est pas saturée, on a l'inégalité stricte $\mathbf{c}_i\mathbf{x}^* < g_i$, et donc $\lambda_i^* = 0$. Réciproquement, si $\lambda_i^* < 0$, alors $\mathbf{c}_i\mathbf{x}^* = g_i$, et la i-ème contrainte est saturée.

Démonstration. Soit $\mathbf{x}^*$ une solution de (7.7), et soit $0 \le s \le m$ le nombre de contraintes saturées en ce point. Quitte à intervertir les lignes de C, on peut supposer que C est de la forme

$$C = \begin{pmatrix} C_E \\ C_I \end{pmatrix} \quad \text{où } C_E \in \mathcal{M}_{s\times d} \text{ contient les contraintes saturées.}$$

1. KKT sont les initiales de Karush, Kuhn et Tucker.

L'indice E indique les contraintes où il y a *égalité*, et l'indice I indique les contraintes où il y a *inégalité stricte*. De même, on écrit $\mathbf{g} = (\mathbf{g}_E, \mathbf{g}_I)^T$. Ainsi, on a $C_E\mathbf{x}^* = \mathbf{g}_E$ et $C_I\mathbf{x}^* \prec \mathbf{g}_I$. En particulier, il existe $\varepsilon > 0$ tel que pour tout $\mathbf{x} \in \mathcal{B}(\mathbf{x}^*, \varepsilon)$, on a aussi $C_I\mathbf{x} \prec \mathbf{g}_I$. Soit $\mathbf{h} \in \mathcal{B}(\mathbf{0}, \varepsilon)$ tel que $C_E\mathbf{h} \preccurlyeq \mathbf{0}$. Alors pour tout $0 \leq t \leq 1$, on a $C(\mathbf{x}^* + t\mathbf{h}) \preccurlyeq \mathbf{g}$, donc $\mathbf{x}^* + t\mathbf{h} \in K$. Par optimalité de $\mathbf{x}^*$, on en déduit que

$$\forall \mathbf{h} \in \mathbb{R}^d \text{ tel que } C_E\mathbf{h} \preccurlyeq \mathbf{0},\ \forall 0 \leq t \leq 1, \quad F(\mathbf{x}^* + t\mathbf{h}) \geq F(\mathbf{x}^*).$$

On fera attention que le réel t est toujours positif. Avec le développement de Taylor pour la partie gauche, on en déduit que

$$\forall t \geq 0, \quad t\langle \nabla F(\mathbf{x}^*), \mathbf{h}\rangle \geq 0, \quad \text{et donc} \quad \langle \nabla F(\mathbf{x}^*), \mathbf{h}\rangle \geq 0.$$

On en déduit (comparer avec (7.5))

$$\forall \mathbf{h} \in \mathbb{R}^d, \quad C_E\mathbf{h} \preccurlyeq \mathbf{0} \implies \langle \nabla F(\mathbf{x}^*), \mathbf{h}\rangle \geq 0.$$

Lemme 7.9 : Lemme de Farkas

Soit $C \in \mathcal{M}_{s,d}$ et $\boldsymbol{\delta} \in \mathbb{R}^d$. Les deux assertions suivantes sont équivalentes.

(i) $\forall \mathbf{h} \in \mathbb{R}^d, \quad C\mathbf{h} \preccurlyeq \mathbf{0} \implies \langle \boldsymbol{\delta}, \mathbf{h}\rangle \geq 0.$

(ii) Il existe $\boldsymbol{\lambda} \in \mathbb{R}^s$ avec $\boldsymbol{\lambda} \preccurlyeq \mathbf{0}$ tel que $\boldsymbol{\delta} = C^T\boldsymbol{\lambda}$.

On en déduit que $\nabla F(\mathbf{x}^*) = C_E^T\boldsymbol{\lambda}_E$ avec $\boldsymbol{\lambda} \in \mathbb{R}^s$, $\boldsymbol{\lambda}_E \preccurlyeq \mathbf{0}$. Quitte à poser $\boldsymbol{\lambda}_i = \mathbf{0}_{\mathbb{R}^{m-s}}$ et $\boldsymbol{\lambda}^* = (\boldsymbol{\lambda}_E, \boldsymbol{\lambda}_I)^T \in \mathbb{R}^m$, on a aussi $\boldsymbol{\lambda} \preccurlyeq \mathbf{0}$ et $\nabla F(\mathbf{x}^*) = C^T\boldsymbol{\lambda}^*$, ce qui conclut la preuve. □

Preuve du lemme de Farkas. Montrons $(ii) \implies (i)$. Si $\boldsymbol{\delta}$ est de la forme $\boldsymbol{\delta} = C^T\boldsymbol{\lambda}$ avec $\boldsymbol{\lambda} \preccurlyeq \mathbf{0}$, on a $\langle \boldsymbol{\delta}, \mathbf{h}\rangle = \langle C^T\boldsymbol{\lambda}, \mathbf{h}\rangle = \langle \boldsymbol{\lambda}, C\mathbf{h}\rangle$. Comme $\boldsymbol{\lambda} \preccurlyeq \mathbf{0}$ et $C\mathbf{h} \preccurlyeq \mathbf{0}$, on en déduit que $\langle \boldsymbol{\delta}, \mathbf{h}\rangle$ est positif.

Montrons maintenant $(i) \implies (ii)$. On note $K := \left\{C^T\boldsymbol{\lambda},\ \boldsymbol{\lambda} \preccurlyeq \mathbf{0}\right\}$, et $\boldsymbol{\delta}_K := P_K(\boldsymbol{\delta})$ la projection de $\boldsymbol{\delta}$ sur K. Notre but est de montrer que $\boldsymbol{\delta} \in K$, ou encore que $\boldsymbol{\delta}_K = \boldsymbol{\delta}$.
D'après la Proposition 7.1, on a

$$\forall \mathbf{k} \in K, \quad \langle \boldsymbol{\delta} - \boldsymbol{\delta}_K, \mathbf{k} - \boldsymbol{\delta}_K\rangle \leq 0. \tag{7.9}$$

En prenant $\mathbf{k} = \mathbf{0} \in K$, on obtient $\langle \boldsymbol{\delta} - \boldsymbol{\delta}_K, \boldsymbol{\delta}_K\rangle \geq 0$, et en prenant $\mathbf{k} = 2\boldsymbol{\delta}_K \in K$, on obtient $\langle \boldsymbol{\delta} - \boldsymbol{\delta}_K, \boldsymbol{\delta}_K\rangle \leq 0$. On en déduit $\langle \boldsymbol{\delta} - \boldsymbol{\delta}_K, \boldsymbol{\delta}_K\rangle = 0$, et (7.9) se réécrit

$$\forall \mathbf{k} \in K, \quad \langle \boldsymbol{\delta} - \boldsymbol{\delta}_K, \mathbf{k}\rangle \leq 0. \tag{7.10}$$

Montrons que $C(\boldsymbol{\delta}_K - \boldsymbol{\delta}) \preccurlyeq 0$. Si $\mathbf{c}_i$ est la i-ème ligne de C, cela est équivalent à montrer que pour tout $1 \leq i \leq m$, on a $\langle \mathbf{c}_i^T, \boldsymbol{\delta}_K - \boldsymbol{\delta}\rangle \leq 0$. D'après la définition de K, en prenant $\lambda := -\mathbf{e}_i$ (le i-ème vecteur canonique), on a $-\mathbf{c}_i^T \in K$. Avec (7.10), on a bien $\langle \boldsymbol{\delta} - \boldsymbol{\delta}_K, (-\mathbf{c}_i)^T\rangle \leq 0$ comme voulu.

Ainsi, $C(\boldsymbol{\delta}_K - \boldsymbol{\delta}) \preccurlyeq 0$. D'après (i), cela implique que $\langle \boldsymbol{\delta}, \boldsymbol{\delta}_K - \boldsymbol{\delta}\rangle \geq 0$. Par ailleurs, on a déjà vu que $\langle \boldsymbol{\delta} - \boldsymbol{\delta}_K, \boldsymbol{\delta}_K\rangle = 0$. On en déduit que $\|\boldsymbol{\delta} - \boldsymbol{\delta}_K\|^2 \leq 0$, et enfin que $\boldsymbol{\delta} = \boldsymbol{\delta}_K$. □

Exercice 7.10

Montrer que si $\mathbf{x}^*$ ne sature aucune contrainte, alors $\nabla F(\mathbf{x}^*) = \mathbf{0}$.

Exercice 7.11

Montrer que $\lambda_i(\mathbf{c}_i\mathbf{x} - g_i) = 0$ pour tout $1 \leq i \leq m$, alors $\langle \boldsymbol{\lambda}, C\mathbf{x}\rangle = \langle \boldsymbol{\lambda}, \mathbf{g}\rangle$.

Nous avons vu qu'un minimiseur vérifie les conditions KKT, mais la réciproque est fausse en général. Le lemme suivant donne un cas où il y a équivalence.

Lemme 7.12

Soit $K := \{C\mathbf{x} \preccurlyeq \mathbf{g}\} \neq \emptyset$, et soit $F : \mathbb{R}^d \to \mathbb{R}$ une fonction de classe C^1 strictement convexe sur K. Alors les problèmes (7.7) et (7.8) ont une unique solution. En particulier, elles sont égales.

Démonstration. Le fait que (7.7) a une unique solution vient du fait qu'on minimise une fonction strictement convexe sur un ensemble convexe. Soit $\mathbf{x}^* \in K$ le minimum de (7.8), qui vérifie donc (7.8). Supposons par l'absurde qu'il existe $\tilde{\mathbf{x}} \in K$ différent de $\mathbf{x}^*$ qui vérifie aussi (7.8) : il existe $\tilde{\boldsymbol{\lambda}} \preccurlyeq \mathbf{0}$ tel que $\nabla F(\tilde{\mathbf{x}}) = C^T\tilde{\boldsymbol{\lambda}}$, et $\tilde{\boldsymbol{\lambda}} \odot (C\tilde{\mathbf{x}} - \mathbf{g}) = 0$.

Pour tout $t \in [0,1]$, le point $\mathbf{x}(t) := \tilde{\mathbf{x}} + t(\mathbf{x}^* - \tilde{\mathbf{x}})$ est dans K (car K est convexe), et $t \mapsto F(\mathbf{x}(t))$ est strictement décroissante (car F est une fonction strictement convexe). En dérivant en $t = 0^+$, on obtient

$$\langle \nabla F(\tilde{\mathbf{x}}), \mathbf{x}^* - \tilde{\mathbf{x}} \rangle < 0, \quad \text{ou encore} \quad \langle \tilde{\boldsymbol{\lambda}}, C(\mathbf{x}^* - \tilde{\mathbf{x}}) \rangle < 0.$$

D'après l'Exercice 7.11, on a $\langle \tilde{\boldsymbol{\lambda}}, C\tilde{\mathbf{x}} \rangle = \langle \tilde{\boldsymbol{\lambda}}, \mathbf{g} \rangle$, et donc $\langle \tilde{\boldsymbol{\lambda}}, C\mathbf{x}^* - \mathbf{g} \rangle < 0$. Par ailleurs, on sait que $\tilde{\boldsymbol{\lambda}} \preccurlyeq \mathbf{0}$ et $C\mathbf{x}^* \preccurlyeq \mathbf{g}$, donc $\langle \tilde{\boldsymbol{\lambda}}, C\mathbf{x}^* - \mathbf{g} \rangle \geq 0$, d'où la contradiction. □

Dans le cas quadratique, où $Q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x}$, et $\nabla Q(\mathbf{x}) = A\mathbf{x} - \mathbf{b}$, la condition $\nabla Q(\mathbf{x}^*) = C^T\boldsymbol{\lambda}$ s'écrit aussi $A\mathbf{x}^* = \mathbf{b} + C^T\boldsymbol{\lambda}^*$. *Si on connait l'ensemble des contraintes saturées E*, alors, pour trouver les points critiques, on pourrait résoudre comme en (7.6) le système

$$\begin{pmatrix} A & C_E^T \\ C_E & 0 \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ \mathbf{g} \end{pmatrix}. \tag{7.11}$$

On peut donc imaginer la procédure combinatoire suivante : pour chaque possibilité d'ensembles de contraintes saturées $C_E \subset C$, on résout le problème (7.11) et on vérifie si $C\mathbf{x}^* \preccurlyeq \mathbf{g}$, et si $\boldsymbol{\lambda}_E \preccurlyeq \mathbf{0}$. Si c'est le cas, on a trouvé un point critique, au sens KKT.

Malheureusement, en pratique, s'il y a m contraintes, il y a 2^m possibilités de créer des ensembles de contraintes saturées. Dès que $m \gg 1$, cette méthode est impraticable[2]. Nous présentons dans la section suivante une méthode différente.

L'algorithme d'Uzawa Nous présentons l'algorithme d'Uzawa, qui permet de calculer numériquement la projection sur $K := \{\mathbf{x} \in \mathbb{R}^d,\ C\mathbf{x} \preccurlyeq \mathbf{g}\}$, c'est à dire de résoudre

$$P_K(\mathbf{b}) = \operatorname{argmin}\left\{\|\mathbf{x} - \mathbf{b}\|^2,\ \ C\mathbf{x} \preccurlyeq \mathbf{g}\right\} = \operatorname{argmin}\left\{\frac{1}{2}\|\mathbf{x}\|^2 - \langle \mathbf{b}, \mathbf{x} \rangle,\ \ C\mathbf{x} \preccurlyeq \mathbf{g}\right\}, \tag{7.12}$$

Pour commencer, on remarque que si $C = \mathbb{I}_d$ et $\mathbf{g} = \mathbf{0}$, alors $K = \mathbb{K}_- := \left\{\mathbf{x} \in \mathbb{R}^d,\ \mathbf{x} \preccurlyeq \mathbf{0}\right\}$. Dans ce cas, il suffit de poser, pour tout $1 \leq i \leq m$, $\big(P_{\mathbb{K}_-}(\mathbf{x})\big)_i = \min\{0, x_i\}$. En Python, on peut effectuer cette projection avec

```
def PKmoins(x):
    return minimum(0, x)
```

L'idée de la méthode d'Uzawa est de transformer le problème (7.12) en une succession de problèmes ne faisant intervenir que des projections sur $\mathbb{K}_-$. Notons $Q(\mathbf{x}) = \frac{1}{2}\|\mathbf{x}\|^2 - \langle \mathbf{b}, \mathbf{x} \rangle$. La fonctionnelle Q est strictement convexe, et admet un unique minimum $\mathbf{x}^*$ sur K. Les condition KKT s'écrivent

$$\exists \boldsymbol{\lambda}^* \in \mathbb{R}^m,\ \boldsymbol{\lambda}^* \preccurlyeq \mathbf{0}, \quad \mathbf{x}^* = C^T\boldsymbol{\lambda}^* + \mathbf{b}, \quad C\mathbf{x}^* \preccurlyeq \mathbf{g} \quad \text{et} \quad \boldsymbol{\lambda}^* \odot (C\mathbf{x}^* - \mathbf{g}) = \mathbf{0}.$$

On a donc

$$\boldsymbol{\lambda}^* \in \mathbb{K}_-, \quad \langle \boldsymbol{\lambda}^*, \mathbf{g} - C\mathbf{x}^* \rangle = 0, \quad \text{et} \quad \forall \boldsymbol{\mu} \in \mathbb{K}_-, \quad \langle \boldsymbol{\lambda}^* - \boldsymbol{\mu}, \mathbf{g} - C\mathbf{x}^* \rangle \geq 0.$$

2. L'algorithme du simplexe est un algorithme qui explore ces différentes possibilités. Elle sort du cadre de ce cours.

En particulier, pour tout $\tau > 0$, on a

$$\forall \boldsymbol{\mu} \in \mathbb{K}_-, \quad \langle \boldsymbol{\lambda}^* - \boldsymbol{\mu}, \boldsymbol{\lambda}^* - [\boldsymbol{\lambda}^* + \tau(\mathbf{g} - C\mathbf{x}^*)]\rangle \leq 0.$$

D'après la Proposition 7.1, cela signifie que $\boldsymbol{\lambda}^*$ est la projection de $\boldsymbol{\lambda}^* + \tau(\mathbf{g} - C\mathbf{x}^*)$ sur $\mathbb{K}_-$, pour tout $\tau > 0$:

$$\forall \tau > 0, \quad \boldsymbol{\lambda}^* = P_{\mathbb{K}_-}\left(\boldsymbol{\lambda}^* + \tau(\mathbf{g} - C\mathbf{x}^*)\right). \tag{7.13}$$

L'idée de l'algorithme d'Uzawa est de résoudre à la fois l'équation (7.13), et l'équation $\mathbf{x}^* = C^T\boldsymbol{\lambda}^* + \mathbf{b}$. On obtient les itérations suivantes (avec ici un pas constant)

$$\boldsymbol{\lambda}_{n+1} = P_{\mathbb{K}_-}\left(\boldsymbol{\lambda}_n + \tau(\mathbf{g} - C\mathbf{x}_n)\right), \quad \text{puis} \quad \mathbf{x}_{n+1} = C^T\boldsymbol{\lambda}_{n+1} + \mathbf{b}. \tag{7.14}$$

Le lemme suivant montre que cet algorithme converge vers la solution $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$ si le pas τ est choisi suffisamment petit.

Lemme 7.13 : Vitesse de convergence de l'algorithme d'Uzawa

Soit $C \in \mathcal{M}_{m\times d}$ et $\mathbf{g} \in \mathbb{R}^m$, et soit $K := \{C\mathbf{x} \preccurlyeq \mathbf{g}\}$. Soit $\mathbf{b} \in \mathbb{R}^d$ et $\mathbf{x}^* := P_K(\mathbf{b})$, et soit $\boldsymbol{\lambda}^*$ le multiplicateur de Lagrange associé.
Alors pour tout $\boldsymbol{\lambda}_0 \in \mathbb{R}^m$ et pour tout $\tau > 0$ tel que $\tau\|CC^T\|_{\text{op}} < 2$, la suite $(\mathbf{x}_n, \boldsymbol{\lambda}_n)$ définie par (7.14) converge linéairement vers $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$ à taux au plus $\left|1 - \tau\|CC^T\|_{\text{op}}\right|$.

Démonstration. D'après (7.13), et en utilisant le fait que $P_{\mathbb{K}_-}$ est une contraction, on a

$$\begin{aligned}
\|\boldsymbol{\lambda}_{n+1} - \boldsymbol{\lambda}^*\| &= \left\|P_{\mathbb{K}_-}\left(\boldsymbol{\lambda}_n + \tau(\mathbf{g} - C[C^T\boldsymbol{\lambda}_n + \mathbf{b}])\right) - P_{\mathbb{K}_-}\left(\boldsymbol{\lambda}^* + \tau(\mathbf{g} - C[C^T\boldsymbol{\lambda}^* + \mathbf{b}])\right)\right\| \\
&\leq \left\|\boldsymbol{\lambda}_n + \tau(\mathbf{g} - CC^T\boldsymbol{\lambda}_n - C\mathbf{b}) - \boldsymbol{\lambda}^* - \tau(\mathbf{g} - CC^T\boldsymbol{\lambda}^* - C\mathbf{b})\right\| \\
&= \left\|\left(\mathbb{I}_m - \tau CC^T\right)(\boldsymbol{\lambda}_n - \boldsymbol{\lambda}^*)\right\| \leq \left\|\mathbb{I}_m - \tau CC^T\right\|_{\text{op}} \|\boldsymbol{\lambda}_n - \boldsymbol{\lambda}^*\| \\
&= \left(\left\|\mathbb{I}_m - \tau CC^T\right\|_{\text{op}}\right)^{n+1} \|\boldsymbol{\lambda}_0 - \boldsymbol{\lambda}^*\|.
\end{aligned}$$

Si $\tau\|CC^T\|_{\text{op}} < 2$, on a bien $\alpha := \left\|\mathbb{I}_m - \tau CC^T\right\|_{\text{op}} < 1$, et on en déduit que $\boldsymbol{\lambda}_n$ converge linéairement vers $\boldsymbol{\lambda}^*$ à taux au plus α. On a aussi

$$\|\mathbf{x}_n - \mathbf{x}^*\| = \left\|(C^T\boldsymbol{\lambda}_n + \mathbf{b}) - (C^T\boldsymbol{\lambda}^* + \mathbf{b})\right\| = \left\|C^T(\boldsymbol{\lambda}_n - \boldsymbol{\lambda}^*)\right\| \leq \|C\|_{\text{op}}\|\boldsymbol{\lambda}_n - \boldsymbol{\lambda}^*\| \leq \|C\|_{\text{op}}\alpha^n.$$

□

On obtient l'algorithme suivant.

Code 7.2 – Algorithme d'Uzawa pour le calcul de projection.

```
def Uzawa(C, g, b, tau=0.2, tol=1e-6, Niter=500):
    # Calcule la projection de b sur le convexe Cx <= g.
    xn, lambdan = b, dot(C, b)
    for n in range(Niter):
        xnp1 = dot(C.transpose(), lambdan) + b
        if norm(xnp1-xn) < tol:
            return xnp1
        xn = xnp1
        lambdan = minimum(0, (lambdan + tau*(g - dot(C, xn))))
    print("Erreur, l'algorithme d'Uzawa n'a pas convergé")
```

CHAPITRE 8

PÉNALISATION DES CONTRAINTES

Dans cette section, on transforme la contrainte en un terme de pénalisation (on dit qu'on *relâche* la contrainte). Dans la suite, on écrit le problème d'optimisation sous contraintes sous la forme

$$\operatorname{argmin}\{F(\mathbf{x}), \quad \mathbf{x} \in E\} \tag{8.1}$$

où E représente l'ensemble des points vérifiant les contraintes. Dans la suite, on suppose que $E \neq \emptyset$.

8.1 Pénalisation extérieure pour les contraintes d'égalité

L'idée de la *pénalisation extérieure* est de remplacer la contrainte $\mathbf{x} \in E$ par un terme qui pénalise les $\mathbf{x} \in \mathbb{R}^d$ qui n'appartiennent pas à E (qui sont extérieurs à E). Plus exactement, on suppose qu'on a une fonction $p : \mathbb{R}^d \to \mathbb{R}$ (la fonction de pénalisation), telle que

$$\forall \mathbf{x} \in \mathbb{R}^d, \quad p(\mathbf{x}) \geq 0 \quad \text{et} \quad p(\mathbf{x}) = 0 \implies \mathbf{x} \in E.$$

La fonction p mesure en quelque sorte une distance entre $\mathbf{x}$ et E. Pour $\mu \in \mathbb{R}$, on introduit le problème

$$\operatorname{argmin}\{F_\mu(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^d\} \quad \text{avec} \quad F_\mu(\mathbf{x}) = F(\mathbf{x}) + \mu p(\mathbf{x}). \tag{8.2}$$

Le réel μ est le *facteur de pénalisation*, et est destinée à tendre vers $+\infty$. Par exemple, si la contrainte est de la forme $g(\mathbf{x}) = \mathbf{0}$, on pourra prendre $p(\mathbf{x}) := \|g(\mathbf{x})\|$. En général, les solutions $\mathbf{x}_\mu$ de (8.2) ne vérifierons pas en général $\mathbf{x}_\mu \in E$. Cependant, on peut espérer que la suite $(\mathbf{x}_\mu)$ converge vers la vraie solution lorsque $\mu \to \infty$. C'est l'objet du lemme suivant.

Lemme 8.1 : Existence de solution pour le problème pénalisé

On suppose que pour tout $\mu > 0$, le problème (8.2) a au moins une solution $\mathbf{x}_\mu$. Alors la fonction $\mu \mapsto p(\mathbf{x}_\mu)$ est décroissante, et les fonctions $\mu \mapsto F(\mathbf{x}_\mu)$ et $\mu \mapsto F_\mu(\mathbf{x}_\mu)$ sont croissantes. Si de plus les fonctions F et p sont continues, alors toute valeur d'adhérence de $(\mathbf{x}_\mu)_{\mu>0}$ est solution de (8.1).

En particulier, si la famille $(\mathbf{x}_\mu)$ a un point d'adhérence, alors l'équation (8.1) a une solution (ce qui n'est pas évident *a priori*).

Démonstration. Soit $0 < \lambda < \mu$. Comme $\mathbf{x}_\lambda$ est le minimiseur de F_λ on a

$$F(\mathbf{x}_\lambda) + \lambda p(\mathbf{x}_\lambda) \leq F(\mathbf{x}_\mu) + \lambda p(\mathbf{x}_\mu) \quad \text{et, de même,} \quad F(\mathbf{x}_\mu) + \mu p(\mathbf{x}_\mu) \leq F(\mathbf{x}_\lambda) + \mu p(\mathbf{x}_\lambda).$$

En sommant ces deux inégalités, on trouve $(\mu - \lambda)\,(p(\mathbf{x}_\lambda) - p(\mathbf{x}_\mu)) \geq 0$. Comme $\lambda \leq \mu$, cela implique que $p(\mathbf{x}_\lambda) \geq p(\mathbf{x}_\mu)$, et donc $\mu \mapsto p(\mathbf{x}_\mu)$ est décroissant.
La première inégalité s'écrit aussi $F(\mathbf{x}_\mu) - F(\mathbf{x}_\lambda) \geq \lambda(p(\mathbf{x}_\lambda) - p(\mathbf{x}_\mu)) \geq 0$, ou encore $F(\mathbf{x}_\lambda) \leq F(\mathbf{x}_\mu)$. Cela montre que la fonction $\mu \mapsto F(\mathbf{x}_\mu)$ est croissante.
Enfin, on a évidemment $F_\lambda \leq F_\mu$ et donc $F_\lambda(\mathbf{x}_\lambda) \leq F_\lambda(\mathbf{x}_\mu) \leq F_\mu(\mathbf{x}_\mu)$, et la fonction $\lambda \mapsto F_\mu(\mathbf{x}_\mu)$ est croissante.

Par ailleurs, soit $\mathbf{x} \in E$ un point quelconque de E. Comme $p(\mathbf{x}) = 0$, on a

$$F(\mathbf{x}_\mu) \leq F_\mu(\mathbf{x}_\mu) \leq F_\mu(\mathbf{x}) = F(\mathbf{x}), \quad \text{et donc} \quad F(\mathbf{x}_\mu) \leq \inf_E F \quad \text{et} \quad F_\mu(\mathbf{x}_\mu) \leq \inf_E F. \tag{8.3}$$

Cela montre d'une part que les fonctions $F(\mathbf{x}_\mu)$ et $F_\mu(\mathbf{x}_\mu)$ sont bornées supérieurement, mais aussi que $\inf_E F > -\infty$.

Soit maintenant $\tilde{\mathbf{x}}$ une valeur d'adhérence de $(\mathbf{x}_\mu)_{\mu>0}$. Cela signifie qu'il existe une suite croissante (μ_n) tendant vers $+\infty$ tel que $\mathbf{x}_{\mu_n} \to \tilde{\mathbf{x}}$. D'après (8.3), la suite $F(\mathbf{x}_{\mu_n})$ est croissante et majorée par $\inf_E F$. Par continuité de F, on a $F(\tilde{\mathbf{x}}) = \inf_E F$. Le point $\tilde{\mathbf{x}}$ est donc un minimum de F, si et seulement si $\tilde{\mathbf{x}} \in E$.

Il reste à démontrer que $\tilde{\mathbf{x}} \in E$, ou encore que $p(\tilde{\mathbf{x}}) = 0$. La suite $(p(\mathbf{x}_{\mu_n}))$ est décroissante, et minorée par 0, donc converge vers une limite $P \geq 0$. De plus, comme $\mu_n \to \infty$ et que $F(\mathbf{x}_{\mu_n}) \to F(\tilde{x}) < \infty$, on a

$$\lim_{n\to\infty} \frac{1}{\mu_n} F_{\mu_n}(\mathbf{x}_{\mu_n}) = \lim_{n\to\infty} \frac{1}{\mu_n} \left(F_{\mu_n}(\mathbf{x}_{\mu_n}) - F(\mathbf{x}_{\mu_n})\right) = \lim_{n\to\infty} p(\mathbf{x}_{\mu_n}) = P.$$

Si $P \neq 0$, alors la suite F_{μ_n} aurait le même comportement asymptotique que $\mu_n P$, et tendrait vers $+\infty$, ce qui contredit le fait que $F_{\mu_n}(\mathbf{x}_{\mu_n})$ est bornée. On a donc $P = 0$, ce qui conclut la preuve. □

Exemple où le problème pénalisé n'est pas bien posé. On fera attention qu'il existe des cas où le problème pénalisé n'a pas de solutions, comme par exemple dans le cas $F(x) = x^3$ et $E = \{0\}$. En prenant $p(x) = x^2$ (qui est bien une fonction de pénalisation de la contrainte $x = 0$, on a toujours

$$\forall \mu \geq 0, \quad \inf\left\{x^3 + \mu x^2\right\} = -\infty.$$

Exercice 8.2

Montrer qu'en prenant $F(x) = x^3$ sur $E = \{0\}$ et la pénalisation $p(x) = x^4$, alors pour tout $\mu > 0$, il existe une unique solution x_μ au problème pénalisé (8.2), et que $x_\mu \to 0$.

8.2 Pénalisation intérieure pour les contraintes d'inégalités

Nous nous intéressons maintenant au cas où E est plutôt de la forme

$$E = \left\{\mathbf{x} \in \mathbb{R}^d,\ \mathbf{x} \succcurlyeq \mathbf{0}\right\}.$$

On pourra évidemment généraliser les idées présentées dans cette section à des cas beaucoup plus généraux. Dans ce cas, on pourrait par exemple utiliser une pénalisation extérieure de la forme

$$p(\mathbf{x}) := \Theta(x_1) + \Theta(x_2) + \cdots + \Theta(x_d), \qquad \text{(pénalisation extérieure)}$$

où $\Theta(x) = |x|$ si $x < 0$ et $\Theta(x) = 0$ si $x \geq 0$. On se ramène alors au cas précédent, où on favorise les $\mathbf{x} \in \mathbb{R}^d$ qui satisfont la contrainte.

On peut aussi imaginer mettre une *barrière* qui empêche $\mathbf{x}$ de quitter la région E. C'est l'idée de la pénalisation intérieure, où on pénalise les $\mathbf{x} \in \mathbb{R}^d$ qui s'approchent trop de la frontière de E. On peut par exemple prendre

$$p(\mathbf{x}) := -\sum_{i=1}^{d} \ln(x_i). \tag{8.4}$$

On se ramène alors à étudier des problèmes pénalisés de la forme (ici, $\kappa \geq 0$)

$$\operatorname{argmin}\Big\{F_\kappa(\mathbf{x}),\ \mathbf{x} \in \mathbb{R}^d\Big\} \quad \text{avec} \quad F_\kappa(\mathbf{x}) := F(\mathbf{x}) + \kappa p(\mathbf{x}).$$

Cette fois-ci, on est intéressé par la limite $\kappa \to 0$. Par construction, les minimiseurs de F_κ, s'ils existent, sont toujours dans notre espace E.

8.3 Algorithmes par pénalisation

La méthode de pénalisation permet de se ramener au cas de l'optimisation sans contrainte. On peut alors utiliser les algorithmes présentés dans les chapitres précédents.

Dans le cas de la pénalisation intérieure par exemple, on peut imaginer un algorithme de la forme suivante :

1) On choisit $\kappa_0 > 0$, et on résout $\mathbf{x}_0 := \operatorname{argmin} F_{\kappa_0}$ par la méthode de notre choix.
2) Puis à l'étape n, on choisit $\kappa_n < \kappa_{n-1}$ et on résout $\mathbf{x}_n := \operatorname{argmin} F_{\kappa_n}$.
3) On s'arrête quand on a trouvé une précision satisfaisante.

Il y a plusieurs commentaires à faire. Pour commencer, pour le point 2), on remarque qu'à chaque itération, il faut résoudre un problème sans contrainte, ce qui peut être coûteux. Il est d'usage à l'étape n de d'initialiser les méthodes en partant de la solution précédente $\mathbf{x}_{n-1}$. On espère en effet que $\mathbf{x}_{n-1}$ est déjà une bonne approximation de $\mathbf{x}_n$, et que peu d'itérations du sous-problème seront nécessaires.

De plus, toujours pour le point 2), il n'est généralement pas nécessaire de calculer parfaitement le point $\mathbf{x}_n$. Une solution approchée suffit, dans le sens où $\mathbf{x}_n$ ne sera utilisé que comme point de départ pour calculer $\mathbf{x}_{n+1}$. On peut donc modifier l'algorithme précédent par l'algorithme plus rapide (présenté ici avec Newton)

1') On choisit $\kappa_0 > 0$, et on effectue M itérations de Newton pour obtenir $\mathbf{x}_0 \approx \operatorname{argmin} F_{\kappa_0}$.
2') Puis à l'étape n, on choisit $\kappa_n < \kappa_{n-1}$ et on effectue M itérations de Newton en partant de $\mathbf{x}_{n-1}$ pour obtenir $\mathbf{x}_n \approx \operatorname{argmin} F_{\kappa_0}$.
3') On s'arrête quand on a trouvé une précision satisfaisante.

Pour le point 1') et 2'), il faut résoudre un problème de Newton, et donc connaître le gradient et la hessienne de F_κ. On trouve, en prenant la pénalisation (8.4),

$$\nabla F_\kappa(\mathbf{x}) = \nabla F(\mathbf{x}) - \kappa \begin{pmatrix} \frac{1}{x_1} \\ \vdots \\ \frac{1}{x_d} \end{pmatrix} \quad \text{et} \quad H_{F_\kappa}(\mathbf{x}) = H_F(\mathbf{x}) + \kappa \begin{pmatrix} \frac{1}{(x_1)^2} & 0 & \dots & \\ 0 & \frac{1}{(x_2)^2} & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \vdots & \frac{1}{(x_d)^2} \end{pmatrix}$$

Ainsi, si on connaît ∇F et H_F, alors calculer ∇F_κ et H_{F_κ} est à la fois facile et rapide numériquement.

On notera pour conclure que le point 3) et/ou 3') ne sont pas évidents. Dans le cas où il n'y avait pas de contraintes, on arrêtait les algorithme lorsque la norme du gradient était petite : $\|\nabla F\| < \text{tol}$. Dans le cas avec contrainte, le gradient de F peut encore être grand au minimiseur sur X.

CHAPITRE 9

SUPPLÉMENTS

9.1 Réseaux de neurones

Dans cette section, nous présentons les réseaux de neurones. Nous en exposons volontairement une version simplifiée (une présentation complète ferait l'objet d'un cours entier), mais qui contient cependant certaines idées importantes.

9.1.1 Notations et définitions

Soit $\sigma : \mathbb{R} \to \mathbb{R}$ une fonction à une seule variable, et $\mathbf{x} \in \mathbb{R}^d$ un vecteur à d composantes. On introduit la notation $\sigma \odot \mathbf{x} \in \mathbb{R}^d$ pour indiquer que la fonction $\sigma : \mathbb{R} \to \mathbb{R}$ agit composantes par composantes sur le vecteur $\mathbf{x}$, c'est à dire que pour toute dimension $d > 0$,

$$\forall \mathbf{x} \in \mathbb{R}^d, \quad \sigma \odot \mathbf{x} = \begin{pmatrix} \sigma(x_1) \\ \vdots \\ \sigma(x_d) \end{pmatrix}.$$

En Python, $\sigma \odot \mathbf{x}$ est simplement obtenu avec `sigma(x)`.

Définition 9.1 : Couche de neurones

Une *couche de neurones* est une fonction $N_\sigma[W, \mathbf{b}] : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}$ de la forme

$$\forall \mathbf{x} \in \mathbb{R}^d, \quad N_\sigma[W, b](\mathbf{x}) := \sigma \odot (W\mathbf{x} + \mathbf{b}),$$

où $\mathbf{b} \in \mathbb{R}^{d_2}$ est un vecteur de taille d_2, $W \in \mathcal{M}_{d_2 \times d_1}$ est une matrice de taille $d_2 \times d_1$, et $\sigma : \mathbb{R} \to \mathbb{R}$ est une fonction réelle croissante. L'ensemble des couches de neurones de $\mathbb{R}^{d_1}$ dans $\mathbb{R}^{d_2}$ est notée $\mathcal{N}(d_1, d_2)$.

En pratique, σ est une fonction fixée une fois pour toute. Les choix les plus courants sont

$$\sigma(x) = \arctan(x) \quad \text{ou} \quad \sigma(x) = \frac{1}{1 + \mathrm{e}^{-x}}.$$

Une couche de neurones est alors paramétrée par une matrice W et un vecteur b, c'est à dire par $d_1 \times d_2 + d_2$ paramètres réels. Notre but sera d'approximer n'importe quelle fonction de $\mathbb{R}^{d_1} \to \mathbb{R}^{d_2}$ par une couche de neurones. Pour cela, il faudra *optimiser* tous ces paramètres. En pratique, une couche de neurones ne suffit pas, et il faut superposer des couches.

Exercice 9.2

Soit $N_\sigma(W, \mathbf{b}) \in \mathcal{N}(d_1, d_2)$ une couche de neurones avec $W \neq 0$ et $\sigma = \arctan(x)$. Montrer que $N_\sigma[W, \mathbf{b}](\mathbf{x}) > 0$ si et seulement si $\mathbf{x}$ est dans un demi-espace qu'on précisera.

Définition 9.3 : Réseau de neurones

Un *réseau de neurones* à K couches de taille $(d_1, d_2, \cdots d_{K+1})$ est une fonction $N_{\sigma_1,\cdots,\sigma_K}[W_1, \mathbf{b}_1, \cdots W_K, \mathbf{b}_k] : \mathbb{R}^{d_1} \to \mathbb{R}^{d_{K+1}}$ de la forme

$$N_{\sigma_1,\cdots,\sigma_K}[W_1, \mathbf{b}_1, \cdots W_K, \mathbf{b}_k] = N_{\sigma_K}[W_K, \mathbf{b}_K] \circ \cdots \circ N_{\sigma_2}[W_2, \mathbf{b}_2] \circ N_{\sigma_1}[W_1, \mathbf{b}_1],$$

où, pour tout $1 \leq k \leq K$, $N_{\sigma_k}[W_k, \mathbf{b}_k]$ est une couche de neurones de $\mathcal{N}(d_k, d_{k+1})$. Dans la suite, on notera $\mathcal{N}(d_1, d_2, \cdots, d_{K+1})$ l'ensemble de tels réseaux de neurones.

L'ensemble $\mathcal{N}(d_1, d_2, \cdots, d_{K+1})$ est donc un ensemble de fonctions. On peut faire le parallèle avec l'ensemble des polynômes $\mathbb{R}_n[X]$, qui est aussi un ensemble de fonctions. Comme nous allons le voir, les couches de neurones bénéficient des mêmes «bonnes» propriétés que les polynômes :

- on peut évaluer rapidement un réseau de neurones, tout comme on peut évaluer rapidement un polynôme ;
- on peut approcher des fonctions par des réseaux de neurones, tout comme on peut approcher des fonctions (continues) par des polynômes (théorème de Stone-Weierstrass) ;
- on peut mettre un nombre arbitraire de paramètres dans les réseaux de neurones, tout comme on peut prendre des polynômes de grand degré ;

et enfin le point le plus important, que nous démontrerons dans la suite,

- on peut facilement évaluer le gradient du réseau de neurones par rapport à ses paramètres. Dans $\mathbb{R}_n[X]$, un polynôme est linéaire en ses coefficients, donc son gradient est trivial (cf Exercice 3.29).

Cela nous permettra d'optimiser facilement les paramètres d'un réseau de neurones pour approcher n'importe quelle fonction. *Last but not least*, on peut faire un parallèle entre ces fonctions et ce qui se passe dans le cerveau, d'où le nom de «réseau de neurones».

Soit $N := N_{\sigma_1,\cdots,\sigma_K}[W_1, \mathbf{b}_1, \cdots, W_K, \mathbf{b}_K] \in \mathcal{N}(d_1, d_2, \cdots, d_{K+1})$ un réseau de neurones. Les entiers d_1 et d_{K+1} sont respectivement le nombre d'entrées et de sortie du réseau de neurones. En plus de ces deux dimensions, un réseau de neurones comprend des dimensions internes $(d_2, \cdots d_K)$ qu'on est libre de choisir. Le nombre de couches K s'appelle la profondeur du réseau de neurones. Lorsque K est grand, on parle de *deep learning*.

De nouveau, les fonctions σ_k sont souvent fixées une fois pour toute, et les paramètres du réseau de neurones sont les matrices W_k et $\mathbf{b}_k$. Un réseau de neurones de taille $(d_1, d_2, \cdots d_{K+1})$ comprend donc

$$d_1 d_2 + d_2 + d_2 d_3 + d_3 + \cdots + d_K d_{K+1} + d_{K+1}$$

paramètres, et ils doivent tous être optimisés pour approcher une fonction donnée.

Exercice 9.4

Montrer que si $\sigma_1 = \sigma_2 = \cdots = \sigma_K = \mathbb{I}$ (la fonction identité de $\mathbb{R}$ dans $\mathbb{R}$), alors il existe $W^* \in \mathcal{M}_{d_1,d_{K+1}}$ et $\mathbf{b}^* \in \mathbb{R}^{d_{K+1}}$ tel que

$$\forall \mathbf{x} \in \mathbb{R}^d, \quad N(\mathbf{x}) = W^* \mathbf{x} + \mathbf{b}^*.$$

Pourquoi les fonctions σ_k doivent-elles être non-linéaires ?

9.1.2 Entraînement d'un réseau de neurones, et rétro-propagation

Le but des réseaux de neurones est d'approcher une fonction h de $\mathbb{R}^{d_1}$ dans $\mathbb{R}^{d_K}$. En pratique, ces fonctions ne sont pas connues, et on ne dispose que de certaines valeurs $\mathbf{y}_s = \mathbf{h}(\mathbf{x}_s)$ avec $1 \leq s \leq S$,

où S est le nombre d'échantillons. Le but est alors de construire un réseau de neurones N tel que $N(\mathbf{x}_s) \approx \mathbf{y}_s$ pour tout $1 \leq s \leq S$. La qualité de ce réseau est mesurée par les écarts $\|N(\mathbf{x}_s) - \mathbf{y}_s\|$. On pourra par exemple chercher à minimiser une fonction de type

$$F[W_1, \mathbf{b}_1, \cdots W_K, \mathbf{b}_K] := \sum_{s=1}^{S} \frac{1}{2} \|N[W_1, \mathbf{b}_1, \cdots W_K, \mathbf{b}_K](\mathbf{x}_s) - \mathbf{y}_s\|^2.$$

La phase d'optimisation s'appelle *l'entrainement du réseau de neurones*, et les données $(\mathbf{x}_s, \mathbf{y}_s)_{1 \leq s \leq S}$ est l'ensemble d'entrainement. Par exemple, si on veut entraîner notre réseaux à reconnaître des chiens sur des photos, on suppose qu'on possède S photos $\mathbf{x}_s$ où on sait si oui ou non il y a un chien. Dans ce cas, on peut prendre $\mathbf{y}_s$ un vecteur de $\mathbb{R}^2$, de la forme $(y_s, 0)^T$ s'il y a chien (avec $y_s \geq 0$ grand si ce chien est très reconnaissable), et de la forme $(0, y_s,)^T$ sinon (avec $y_s \geq 0$ grand si c'est évident qu'il n'y aucun chien).

Les paramètres à optimiser sont les matrices W_k et les vecteurs $\mathbf{b}_k$. Afin d'appliquer des algorithme de descente, il faut calculer le gradient de F par rapport à tous les coefficients de W_k et $\mathbf{b}_k$. La forme très spécifiques des réseaux de neurones permet de calculer efficacement ces gradients, et c'est ce que nous présentons maintenant.

9.1.3 Calcul des gradients

Afin de ne pas alourdir cette courte présentation, nous prenons un réseau à deux couches (la généralisation est évidente une fois qu'on comprend $K = 2$). Notre réseau peut alors être vu comme la succession des opérations suivantes :

$$\mathbf{x}_1 = \mathbf{x}, \quad \mathbf{z}_1 = W_1\mathbf{x}_1 + \mathbf{b}_1, \quad \mathbf{x}_2 = \sigma_1 \odot \mathbf{z}_1, \quad \mathbf{z}_2 = W_2\mathbf{x}_2 + \mathbf{b}_2, \quad \mathbf{x}_3 = \sigma_2 \odot \mathbf{z}_2,$$

et, enfin, si on veut calculer la qualité de notre réseau,

$$E = \frac{1}{2}\|\mathbf{x}_3 - \mathbf{y}\|^2.$$

Ici, on voit toutes ces variables comme des fonctions, par exemple $\mathbf{z}_1 = \mathbf{z}_1(W_1, \mathbf{x}_1, \mathbf{b}_1)$. Autrement dit, on utilise la même notation pour les variables et pour les fonctions. On peut alors calculer les différentielles de ces variables/fonctions par rapport à ses variables. On commence avec $\mathbf{z}_1 := \mathbf{z}_1(W_1, \mathbf{x}_1, \mathbf{b}_1)$, et on remarque que $\mathbf{z}_1$ est linéaire dans chacune de ses variables. On a donc

$$\mathrm{d}_{\mathbf{x}_1}\mathbf{z}_1 : \widetilde{\mathbf{x}_1} \mapsto W_1\widetilde{\mathbf{x}}_1, \quad \mathrm{d}_{W_1}\mathbf{z}_1 : \widetilde{W_1} \mapsto \widetilde{W_1}\mathbf{x}_1, \quad \text{et} \quad \mathrm{d}_{\mathbf{b}_1}\mathbf{z}_1 : \widetilde{\mathbf{b}_1} \mapsto \widetilde{\mathbf{b}_1},$$

où on a encore noté pour simplifier $\mathrm{d}_{\mathbf{x}_1}\mathbf{z}_1 = \mathrm{d}_{\mathbf{x}_1}\mathbf{z}_1(W_1, \mathbf{x}_1, \mathbf{b}_1)$, etc. Le calcul pour $\mathbf{z}_2 = \mathbf{z}_2(W_2, \mathbf{x}_2, \mathbf{b}_2)$ est similaire. Pour $\mathbf{x}_2$, on a composante par composante $x_{1,i} = \sigma(z_{1,i})$, donc en dérivant $x'_{1,i} = \sigma'(z_{1,i})$. Autrement dit,

$$\mathrm{d}_{\mathbf{z}_1}\mathbf{x}_2 : \widetilde{\mathbf{z}_1} \mapsto \big(\sigma' \odot \mathbf{z}_1\big) \odot \widetilde{\mathbf{z}_1}.$$

Dans cette expression, le premier $\odot$ est la fonction σ' appliquée à chaque composante de $\mathbf{z}_1$, et le deuxième $\odot$ est la multiplication termes à termes de deux vecteurs (produit de Hadamard). En Python, on pourrait écrire `dsigma(z1)*z1tilde`. Enfin, pour la dérivée de E par rapport à $\mathbf{x}_3$, on a comme d'habitude

$$\mathrm{d}_{\mathbf{x}_3}E : \widetilde{\mathbf{x}_3} \mapsto (\mathbf{x}_3 - \mathbf{y})^T\widetilde{\mathbf{x}_3}.$$

On peut maintenant propager ces dérivées grâce à la règle de la chaîne. Par exemple, pour calculer la différentielle de E par rapport à $\mathbf{b}_2$, on écrit

$$\mathrm{d}_{\mathbf{b}_2}E = \mathrm{d}_{\mathbf{x}_3}E \cdot \mathrm{d}_{\mathbf{b}_2}\mathbf{x}_3 = \mathrm{d}_{\mathbf{x}_3}E \cdot \mathrm{d}_{\mathbf{z}_2}\mathbf{x}_3 \cdot \mathrm{d}_{\mathbf{b}_2}\mathbf{z}_2,$$

où on a composé toutes les applications linéaires qui apparaissent. Ainsi, avec nos calculs précédents,

$$\begin{aligned}\mathrm{d}_{\mathbf{b}_2}E(\widetilde{\mathbf{b}_2}) &= \mathrm{d}_{\mathbf{x}_3}E\left(\mathrm{d}_{\mathbf{z}_2}\mathbf{x}_3\left(\mathrm{d}_{\mathbf{b}_2}\mathbf{z}_2(\widetilde{\mathbf{b}_2})\right)\right) = \mathrm{d}_{\mathbf{x}_3}E\left(\mathrm{d}_{\mathbf{z}_2}\mathbf{x}_3\left(\widetilde{\mathbf{b}_2}\right)\right) = \mathrm{d}_{\mathbf{x}_3}E\left((\sigma_2'\odot\mathbf{z}_2)\odot\widetilde{\mathbf{b}_2}\right)\\ &= (\mathbf{x}_3-\mathbf{y})^T\left[(\sigma_2'\odot\mathbf{z}_2)\odot\widetilde{\mathbf{b}_2}\right].\end{aligned}$$

On peut simplifier cette expression en remarquant que si $\mathbf{a},\mathbf{b},\mathbf{c},\in\mathbb{R}^d$ sont trois vecteurs de même taille, alors

$$\mathbf{a}^T(\mathbf{b}\odot\mathbf{c}) = \sum_{i=1}^{d} a_ib_ic_i, \quad \text{et donc} \quad \mathbf{a}^T(\mathbf{b}\odot\mathbf{c}) = (\mathbf{a}\odot\mathbf{b})^T\mathbf{c} = \langle\mathbf{a}\odot\mathbf{b},\mathbf{c}\rangle_{\mathbb{R}^d}.$$

On en déduit enfin que

$$\boxed{\nabla_{\mathbf{b}_2}E = (\sigma_2'\odot\mathbf{z}_2)\odot(\mathbf{x}_3-\mathbf{y}).} \tag{9.1}$$

Calculons de même la dérivée de E par rapport à W_2. On obtient cette fois

$$\mathrm{d}_{W_2}E = \mathrm{d}_{\mathbf{x}_3}E\cdot\mathrm{d}_{\mathbf{z}_2}\mathbf{x}_3\cdot\mathrm{d}_{W_2}\mathbf{z}_2,$$

et donc

$$\begin{aligned}\mathrm{d}_{W_2}E\left(\widetilde{W_2}\right) &= \mathrm{d}_{\mathbf{x}_3}E\left(\mathrm{d}_{\mathbf{z}_2}\mathbf{x}_3\left(\mathrm{d}_{\mathbf{b}_2}\mathbf{z}_2(\widetilde{W_2})\right)\right) = \mathrm{d}_{\mathbf{x}_3}E\left(\mathrm{d}_{\mathbf{z}_2}\mathbf{x}_3\left(\widetilde{W_2}\mathbf{x}_2\right)\right) = \mathrm{d}_{\mathbf{x}_3}E\left((\sigma'\odot\mathbf{x}_2)\odot\widetilde{W_2}\mathbf{x}_2\right)\\ &= (\mathbf{x}_3-\mathbf{y})^T\left[(\sigma'\odot\mathbf{x}_2)\odot\widetilde{W_2}\mathbf{x}_2\right] = \left[(\mathbf{x}_3-\mathbf{y})\odot(\sigma'\odot\mathbf{x}_2)\right]^T\widetilde{W_2}\mathbf{x}_2.\end{aligned}$$

On utilise maintenant le fait que pour $\mathbf{a}\in\mathbb{R}^n$, $\mathbf{b}\in\mathbb{R}^m$ et $W\in\mathcal{M}_{n,m}$, on a

$$\mathbf{a}^TW\mathbf{b} = \mathrm{Tr}\ \left(\mathbf{a}^TW\mathbf{b}\right) = \mathrm{Tr}\ \left(\mathbf{b}\mathbf{a}^TW\right) = \mathrm{Tr}\ \left(\left(\mathbf{a}\mathbf{b}^T\right)^TW\right) = \langle\mathbf{a}\mathbf{b}^T,W\rangle_{\mathcal{M}_{m,n}(\mathbb{R})},$$

où le dernier produit scalaire est celui de Frobenius pour les matrices (voir l'Exercice 3.32). On en déduit que

$$\boxed{\nabla_{W_2}E = \left[(\sigma_2'\odot\mathbf{x}_2)\odot(\mathbf{x}_3-\mathbf{y})\right]\mathbf{x}_2^T.} \tag{9.2}$$

On calcule de même le gradient de E par rapport à $\mathbf{b}_1$ et à W_1, et on trouve

$$\boxed{\nabla_{\mathbf{b}_1}E = (\sigma_1'\odot\mathbf{z}_1)\odot W_2^T\left[(\sigma_2'\odot\mathbf{z}_2)\odot(\mathbf{x}_3-\mathbf{y})\right].} \tag{9.3}$$

$$\boxed{\nabla_{W_1}E = (\sigma_1'\odot\mathbf{z}_1)\odot W_2^T\left[(\sigma_2'\odot\mathbf{z}_2)\odot(\mathbf{x}_3-\mathbf{y})\right]\mathbf{x}_1^T.} \tag{9.4}$$

Exercice 9.5

Montrer ces deux formules.

Le point important est que le calcul tous ces gradients ne nécessite que des opérations (très) rapides et parallélisables du point de vue numérique ! C'est ce qui fait la force et le succès de ces fonctions.

On peut maintenant optimiser nos paramètres avec une méthode de gradient, comme par exemple une méthode de gradient à pas constant. Dans ce dernier cas, le pas τ est appelé *taux d'apprentissage*.

En regardant les formules (9.1)-(9.2)-(9.3)-(9.4), on voit que les mêmes vecteurs apparaissent plusieurs fois. Il est donc intéressant de calculer ces vecteurs une fois pour toute. On peut par exemple poser

$$\begin{gathered}\mathrm{dx3E} := (\mathbf{x}_3-\mathbf{y}), \quad \mathrm{db2E} := (\sigma_2'\odot\mathbf{z}_2)\odot\mathrm{dx3E}, \quad \mathrm{dW2E} := (\mathrm{db2E})\mathbf{x}_2^T\\ \mathrm{db1E} := (\sigma_1'\odot\mathbf{z}_1)\odot W_2^T(\mathrm{db2E}), \quad \text{and} \quad \mathrm{dW1E} := (\mathrm{db1E})\mathbf{x}_1^T.\end{gathered}$$

Avec ces formules, on voit que l'information est propagée à l'envers : on calcule d'abord les dérivées de E par rapport aux paramètres $(W_2, \mathbf{b}_2)$, puis ensuite celles par rapport aux paramètres $(W_1, \mathbf{b}_1)$. On parle de rétro-propagation des dérivées. Ce phénomène est très général, et est à la base des librairies de *différentiations automatiques*.

Exercice 9.6

Soit $g : \mathbb{R} \to \mathbb{R}$ une fonction difficile à calculer (dans le sens où g et g' prennent du temps à évaluer numériquement).
a/ Comment implémenter efficacement les fonctions suivantes :

$$f_1(x) = g(x) + \frac{1}{g(x)} \quad \text{et} \quad f_2(x) = g(x) + g(g(x)).$$

b/ Calculer f_1' et f_2'. Comment implémenter efficacement ces fonctions ?

9.2 «Bonnes pratiques» numériques.

Dans cette section, nous commençons par présenter quelques techniques de code pour gagner du temps, et nous présentons aussi quelques outils de *profilage* de code, qui permettent d'améliorer la vitesse d'un code.

9.2.1 Astuces de code

Un exemple important Commençons par comparer ces codes :

```
1 N = int(1e8) # un très grand nombre entier
2 xx = linspace(0, 1, N) # environ 0.6s.
3 A = [x**2 for x in xx] # environ 35s.
4 B = [x*x for x in xx] # environ 17s.
5 C = xx**2 # environ 1.23s.
6 D = xx*xx # environ 1.24s.
```

Dans tous les cas, on calcule la liste $[1^2, 2^2, \cdots, N^2]$, mais les temps de calcul sont différents, et peuvent varier d'un facteur 30 (!) C'est un facteur important (on préfère un calcul qui dure un jour qu'un mois). Il est important de comprendre d'où vient cette différence pour améliorer la vitesse d'un code.

La différence principale entre A et B est la fonction x^2 qui est remplacée par $x \times x$. Dans le premier cas, Python appelle une fonction *software* de calcul de puissance (une fonction codée quelque part), alors que dans le deuxième cas, il utilise la multiplication, qui est codée en *hardware* : il y a une puce électronique physique dédiée à la multiplication. Une multiplication est donc quasiment instantanée dans un ordinateur. Cela explique pourquoi on gagne un facteur 2 entre A et B.

La différence entre A et C est que dans A, il y a une boucle `for`. Les éléments de la liste sont donc traités un par un, les uns après les autres. En revanche, pour C, on applique la fonction x^2 à un `array`, et Python peut traiter plusieurs éléments à la fois (calcul parallèle). Le gain de temps est énorme ! On retiendra donc qu'il faut éviter au maximum éviter les boucles `for` pour des vecteurs de grandes tailles : il faut traiter des matrices comme des matrices, et non comme une suite d'éléments.

Enfin, il y a peu de différence de temps de calcul entre C et D. Pour expliquer le temps de calcul de D, on commence par remarquer qu'il faut environ 0.7 secondes pour créer le vecteur `xx`. Ce temps est du principalement à l'allocation de l'espace mémoire (10^8 float). Dans le calcul de D, certes la fonction $x \times x$ est plus rapide que x^2, mais le vecteur `xx` doit être copié (+0.6s), et le nouveau vecteur D doit aussi être alloué en mémoire (+0.6s), d'où le temps de calcul de $1.23s$. On voit que le calcul de $x \times x$ est devenu négligeable devant le temps l'allocation mémoire.

Éviter les boucles for. L'exemple précédent montre qu'un important gain de temps de calcul peut-être fait en évitant les boucles `for`. Nous expliquons ici les principales astuces pour s'en passer (au risque souvent d'un code moins lisible).

ANNEXE A

FORMULAIRES

Nous rappelons dans cet annexe quelques formules à connaître.

A.1 Rappel d'algèbre linéaire

Soit $A \in \mathcal{S}_d(\mathbb{R})$, et soit $\lambda_1 \leq \cdots \leq \lambda_d$ ses valeurs propres rangées en ordre croissant. La matrice A est diagonalisable. Soit $U \in O(d)$ vérifiant $U^{-1} = U^T$ et $D = \mathrm{diag}(\lambda_1, \cdots, \lambda_d)$ tel que $A = U^{-1}DU$. Pour tout $\mathbf{x} \in \mathbb{S}^{d-1}$, on pose $\mathbf{y} := U\mathbf{x}$. On a

$$\|\mathbf{y}\|^2 = \langle \mathbf{y}, \mathbf{y} \rangle = \langle U\mathbf{x}, U\mathbf{x} \rangle = \langle \mathbf{x}, U^T U\mathbf{x} \rangle = \langle \mathbf{x}, \mathbf{x} \rangle = \|\mathbf{x}\|^2,$$

donc $\mathbf{y} \in \mathbb{S}^{d-1}$. De plus, on a

$$\langle \mathbf{x}, A\mathbf{x} \rangle = \langle \mathbf{x}, U^T DU\mathbf{x} \rangle = \langle U\mathbf{x}, DU\mathbf{x} \rangle = \langle \mathbf{y}, D\mathbf{y} \rangle = \sum_i \lambda_i y_i^2 \quad \begin{cases} \geq \lambda_1 \sum_i y_i^2 = \lambda_1; \\ \leq \lambda_d \sum_i y_i^2 = \lambda_d. \end{cases}$$

ce qui prouve que $\lambda_1 \leq \langle \mathbf{x}, A\mathbf{x} \rangle \leq \lambda_d$ pour tout $\mathbf{x} \in \mathbb{S}^{d-1}$, avec égalité pour $\mathbf{x} = U^T \mathbf{e}_1$ et $\mathbf{x} = U^T \mathbf{e}_d$. Cela prouve

$$\boxed{\lambda_1 := \min \left\{ \langle \mathbf{x}, A\mathbf{x} \rangle,\ \mathbf{x} \in \mathbb{S}^{d-1} \right\} \quad \text{et} \quad \lambda_d := \max \left\{ \langle \mathbf{x}, A\mathbf{x} \rangle,\ \mathbf{x} \in \mathbb{S}^{d-1} \right\}.} \tag{A.1}$$

Si $\mathbf{x} \neq \mathbf{0}$ n'est pas de norme 1, alors $\mathbf{x}/\|\mathbf{x}\| \in \mathbb{S}^{d-1}$, et donc $\lambda_1 \leq \langle \mathbf{x}, A\mathbf{x} \rangle / \|\mathbf{x}\|^2 \leq \lambda_d$, ou encore

$$\boxed{\forall \mathbf{x} \in \mathbb{R}^d, \quad \lambda_1 \|\mathbf{x}\|^2 \leq \langle \mathbf{x}, A\mathbf{x} \rangle \leq \lambda_d \|\mathbf{x}\|^2.} \tag{A.2}$$

Par ailleurs, on a par définition

$$\|A\|_{\mathrm{op}}^2 = \max\{\|A\mathbf{x}\|^2,\ \mathbf{x} \in \mathbb{S}^{d-1}\} = \max\{\langle A\mathbf{x}, A\mathbf{x} \rangle,\ \mathbf{x} \in \mathbb{S}^{d-1}\} = \max\{\langle \mathbf{x}, A^2\mathbf{x} \rangle,\ \mathbf{x} \in \mathbb{S}^{d-1}\} = \lambda_d(A^2).$$

Les valeurs propres de A^2 sont les λ_i^2. En prenant la racine carrée, on obtient $\|A\|_{\mathrm{op}} = \max_{1 \leq i \leq d} |\lambda_i|$, et, si les λ_i sont rangés par ordre croissant,

$$\boxed{\|A\|_{\mathrm{op}} = \max\{|\lambda_1|, |\lambda_d|\}.} \tag{A.3}$$

A.2 Formules de Taylor

Pour commencer, la **formule de Taylor** avec reste intégrale affirme que si $F : \mathbb{R} \to \mathbb{R}$ est de classe C^∞, alors pour tout $k \in \mathbb{N}$, tout $x \in \mathbb{R}$ et tout $h \in \mathbb{R}$, on a

$$F(x+h) = F(x) + F'(x)h + \frac{F''(x)}{2}h^2 + \cdots + \frac{F^{(k)}}{k!}h^k + h^{k+1}\left(\frac{1}{k!}\int_0^1 (1-t)^k F^{(k+1)}(x+ht)\mathrm{d}t\right).$$

Les cas les plus importants sont pour $k = 1$ et $k = 2$, où on obtient respectivement

$$F(x+h) = F(x) + F'(x)h + h^2\int_0^1 (1-t)F''(x+ht)\mathrm{d}t,$$

et

$$F(x+h) = F(x) + F'(x)h + F''(x)h^2 + \frac{h^3}{2}\int_0^1 (1-t)^2 F'''(x+ht)\mathrm{d}t.$$

A.3 Calcul spectral

Si $A \in \mathcal{S}_d(\mathbb{R})$ est une matrice symétrique réelle, A est diagonalisable. Il existe une matrice orthonormale $U \in O(d)$ et une matrice $D = \mathrm{diag}(\lambda_1, \cdots, \lambda_d)$ telle que $A = U^T D U$. Si $F : \mathbb{R} \to \mathbb{R}$ est une fonction continue, on définit la matrice $F(A) \in \mathcal{M}_d(\mathbb{R})$ par

$$F(A) := U^T \begin{pmatrix} F(\lambda_1) & 0 & 0 & \cdots & 0 \\ 0 & F(\lambda_2) & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & 0 \\ 0 & 0 & 0 & \cdots & F(\lambda_d) \end{pmatrix} U.$$

C'est la matrice qu'on obtient en remplaçant λ_k par $F(\lambda_k)$ dans la diagonalisation de A. On admettra que $F(A)$ ne dépend pas du choix de la diagonalisation de A, de sorte que $F : \mathcal{S}_d \to \mathcal{S}_d$ est bien définie. On étend ainsi les fonctions continues de $\mathbb{R}$ dans $\mathbb{R}$ en des fonctions de $\mathcal{S}_d$ dans $\mathcal{S}_d$. Cette extension s'appelle **le calcul spectral**.