

# **Projet - Minimisation des fonctionnelles quadratiques par des méthodes à direction de descente**

## **Aéro 3 - Tronc commun**

Julien Fresnel and 3 others

# Table des matières

<b>Table des matières</b>	<b>2</b>
<b>Introduction</b>	<b>3</b>
<b>Formulation et analyse mathématique</b>	<b>4</b>
I. Identification au sens des moindres carrés linéaires	4
A. Ajustement linéaire	4
II. Le problème est-il bien posé ?	9
A. Autour de la fonction quadratique	9
B. Unicité de la solution Q	18
C. Conditionnement du problème	20
<b>Méthodes à directions de descente : une étude numérique</b>	<b>22</b>
I. Principes généraux	22
A. Direction de descente	22
B. Pas de descente	23
C. Critère d'arrêt	25
II. Méthode de descente du gradient ou de plus forte descente	27
A. Direction de plus forte descente	27
B. L'algorithme de descente du gradient à pas fixe	27
C. L'algorithme de descente du gradient à pas optimal	32
III. L'algorithme des gradients conjugués	35
A. Direction du gradient conjugué	35
B. Pseudo-code et test numérique	37
IV. Analyse des résultats numériques	41
A. Les résultats	41
B. Comportements des méthodes	49

# Introduction

Ce projet (accompagné d'une partie numérique<sup>1</sup> traitée en Travaux Pratiques) poursuit deux objectifs. Le premier consiste à étudier (au sens des mathématiques) un problème d'analyse de données (plus précisément un problème de calibrage ou identification de paramètres d'un modèle) posé au sens des moindres carrés. Cette étude fait l'objet de la section 2. Afin de résoudre numériquement le problème, nous allons considérer trois algorithmes à direction de descente : deux du type à direction de plus forte pente (l'un à pas fixe et l'autre à pas optimal), et le troisième basé sur les directions dites des gradients conjugués (algorithme des gradients conjugués). Le second objectif consiste à présenter, étudier puis comparer théoriquement et numériquement ces trois algorithmes à directions de descentes. La présentation et l'étude numérique de ces trois méthodes feront l'objet de la section 3, tandis que la sections 4 sera consacrée à l'étude théorique (mathématiques) des méthodes. On pourra ainsi mettre en relation les résultats numériques avec les développements théoriques.

---

<sup>1</sup> Tous les codes seront fournies en fichier .py avec le rapport

# Formulation et analyse mathématique

## I. Identification au sens des moindres carrés linéaires

La finalité de cette sous-section est d'expliciter le problème d'optimisation quadratique sans contrainte découlant de l'application de l'approche aux moindres carrés dans l'identification de paramètres d'un modèle (linéaire vis-à-vis de ces paramètres à déterminer)

### A. Ajustement linéaire

Nous allons traiter dans ce projet l'approche simple (la plus simple) de l'ajustement d'une loi expérimentale par régression linéaire : l'ajustement affine. Pour illustrer nos propos, on considère l'archive "ex1tp1.zip" des données des hauteurs d'un échantillon de  $m = 50$  enfants entre 2 et 8 ans. L'archive contient :

- "dataP.dat" : fichier des âges des enfants ;
- "dataQ.dat" : fichier des hauteurs des enfants.

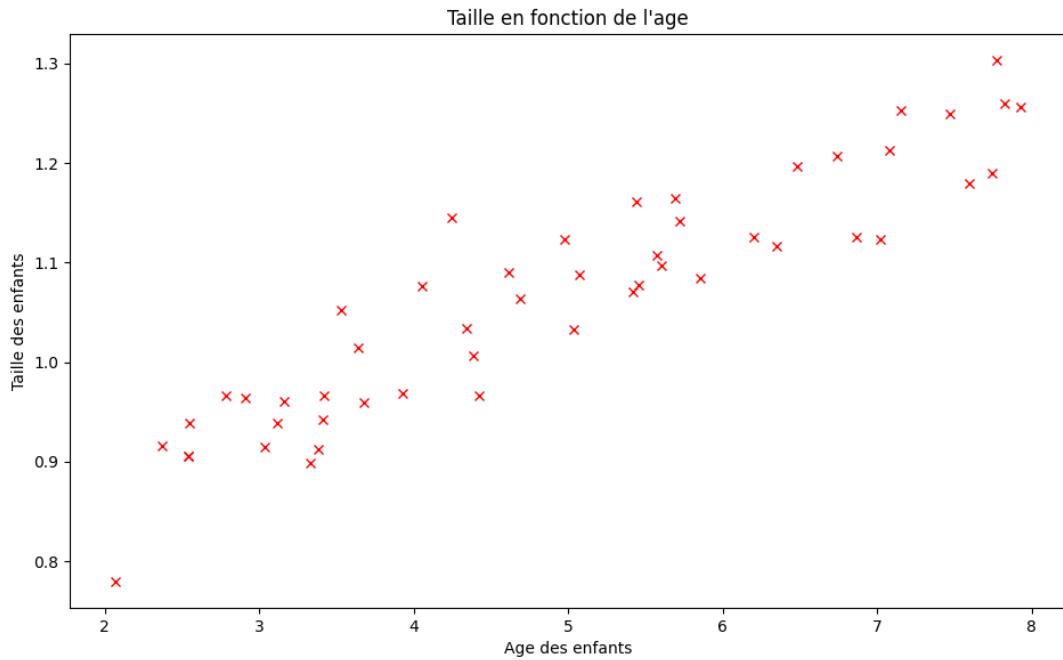
1. Lire les fichiers des données et sauvegarder les âges dans le vecteur  $p$  et les hauteurs dans le vecteur  $q$ . **Fonction de la librairie numpy** : `numpy.loadtxt load`. Puis, tracer dans un graphique les couples  $(p_i, q_i)$   $i = 1, \dots, m$ .

Dans cette question nous allons effectuer un code python dans lequel nous extrairons les données des fichiers "dataP.dat" et "dataQ.dat" dans des listes  $x$  et  $y$ . Nous tracerons<sup>2</sup> ensuite le nuage des points de ces données qui représentent la corrélation entre l'âge et la taille d'enfants.

```
x = np.loadtxt("dataP.dat")
y = np.loadtxt("dataQ.dat")

def display_data(x,y, xtry):
    plt.figure("2.1.2 Ajustement linéaire")
    plt.plot(x,y, color ='red', label="Taille en fonction de l'age", marker ='x', linestyle='none')
    plt.legend()
    plt.show()
```

<sup>2</sup> Nous précisons que tous les tracés sont réalisés grâce à la library `matplotlib` de python



2. Donner une formulation du problème dans laquelle on fait référence à la famille de fonctions affines indexée par les paramètres  $c_1$  et  $c_2$ . C'est de cette formulation du problème que l'on tire l'appellation : ajustement linéaire.

Le but de ce problème est de réaliser une approximation d'un nuage de point à l'aide d'une droite affine. Ce projet aura pour but de déterminer une approche efficace en faisant appel à l'algèbre linéaire. On comparera ces méthodes pour déterminer quelle est la plus efficace. Dans un premier temps et d'après ce que nous avons vu en cours nous pouvons poser la droite linéaire suivante :  $y = c_1 + c_2 \cdot p$  où  $p$  est l'âge des enfants et  $(c_1, c_2)$  le couple de coefficient de la droite de régression linéaire on a d'après le cours :

$$\frac{\sum (q_i - (c_1 + c_2 \cdot p))^2}{\sum (c_1 + c_2 \cdot p + r_i - (c_1 + c_2 \cdot p))^2}$$

Cela revient donc à minimiser :

$$\sum (r_i)^2$$

### 3. Vérifier que le système d'équation se réécrit sous la forme matricielle $q = Xc + r$

On cherche alors à retrouver le système précédent (4) en partant de la forme matricielle. Nous procérons de la façon suivante :

$$\begin{aligned} q = Xc + r &\Leftrightarrow \begin{pmatrix} q_1 \\ \vdots \\ q_m \end{pmatrix} = \begin{pmatrix} 1 & p_1 \\ \vdots & \vdots \\ 1 & p_m \end{pmatrix} \cdot \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} + \begin{pmatrix} r_1 \\ \vdots \\ r_m \end{pmatrix} \\ &\Leftrightarrow \begin{cases} c_1 + c_2 \cdot p_1 + r_1 &= q_1 \\ \vdots & \vdots \\ c_1 + c_2 \cdot p_m + r_m &= q_m \end{cases} \\ &\Leftrightarrow q_i = c_1 + c_2 \cdot p_i + r_i \end{aligned}$$

C'est bien ce qu'il faut que l'on obtienne.

### 4. Montrer que poser au sens des moindres carrés le problème d'identification des paramètres $c_1$ et $c_2$ du modèle consiste à résoudre le problème de minimisation sans contrainte :

$$\begin{cases} \text{Minimiser } F(c) \\ c \in \mathbb{R}^2 \end{cases}$$

On doit alors résoudre ce système. On pose ce qu'on a déjà énoncé précédemment :

$$\begin{aligned} \min_{c_1, c_2} \sum (r_i)^2 &= \min_c \|r\|^2 \\ \Leftrightarrow \min_c \frac{1}{2} \|q - X \cdot c\|^2 \end{aligned}$$

On pose et on développe :

$$\begin{aligned} F(c) &= \frac{1}{2} \|q - X \cdot c\|^2 \\ &\Leftrightarrow \frac{1}{2} (q - X \cdot c)^T (q - X \cdot c) \\ &\Leftrightarrow \frac{1}{2} (q^T - c^T \cdot X^T) (q - X \cdot c) \\ &\Leftrightarrow \frac{1}{2} c^T X^T X c - \frac{1}{2} q^T X c - \frac{1}{2} c^T X^T q + \frac{1}{2} q^T q \end{aligned}$$

Avec les règles matricielles sur les transpositions de matrices et sur les multiplications de transposée on obtient le résultats suivant :

$$F(c) = \frac{1}{2}c^T X^T X c - (X^T q)^T c + \frac{1}{2}||q||^2$$

5. Vérifier que :

$$\begin{aligned} X^T X &= \begin{pmatrix} 1 & \dots & 1 \\ p_1 & \dots & p_m \end{pmatrix} \cdot \begin{pmatrix} 1 & p_1 \\ \vdots & \vdots \\ 1 & p_m \end{pmatrix} \\ &= \begin{pmatrix} m & (1\dots 1) \begin{pmatrix} p_1 \\ \vdots \\ p_m \end{pmatrix} \\ (p_1 \dots p_m) \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} & (p_1 \dots p_m) \begin{pmatrix} p_1 \\ \vdots \\ p_m \end{pmatrix} \end{pmatrix} \end{aligned}$$

Ainsi d'après l'énoncé et les règles sur les multiplications de matrices on peut conclure que :

$$X^T X = \begin{pmatrix} m & p^T 1 \\ p^T 1 & ||p||^2 \end{pmatrix}$$

Et de plus :

$$\begin{aligned} X^T q &= \begin{pmatrix} 1 & \dots & 1 \\ p_1 & \dots & p_m \end{pmatrix} \cdot \begin{pmatrix} q_1 \\ \vdots \\ q_m \end{pmatrix} \\ &= \begin{pmatrix} (1\dots 1) \begin{pmatrix} q_1 \\ \vdots \\ q_m \end{pmatrix} \\ (p_1 \dots p_m) \begin{pmatrix} q_1 \\ \vdots \\ q_m \end{pmatrix} \end{pmatrix} \\ &= \begin{pmatrix} (q_1 \dots q_m) \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \\ (p_1 \dots p_m) \begin{pmatrix} q_1 \\ \vdots \\ q_m \end{pmatrix} \end{pmatrix} \end{aligned} \quad \text{D'après les propriétés de la transposée}$$

$$X^T q = \begin{pmatrix} q^T 1 \\ p^T q \end{pmatrix}$$

Voici ci-après le code qui a permis la constructeur et le calcul des deux matrices précédentes. Le résultat obtenu est aussi présenté.

```

def X_build(x,y):
    n = len(y)
    q = np.zeros((n,1))
    X = np.ones((n,2))

    for i in range(0,n):
        q[i,:] = y[i]
        X[i,1] = x[i]

    XTX = (np.transpose(X))@X
    XTq = (np.transpose(X))@q

    return X,q,XTX,XTq

```

```

XTX = [[ 50.          246.1785986 ]
       [ 246.1785986  1358.30473017]]
XTq = [[ 53.23430273]
       [271.44405179]]

```

## 6. La fonction F est-elle quadratique ? (justifier votre réponse)

La fonction F est une fonction polynôme du second degré à deux inconnues, elle possède donc l'écriture matricielle suivante :

$$\forall c = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \in M_{2,1}(\mathbb{R})$$

$$F(c) = \frac{1}{2}c^T X^T X c - (X^T q)^T c + \frac{1}{2} \|q\|^2$$

Ainsi la fonction est bel et bien quadratique car  $X^T X$  est une matrice réelle définie positive et  $X^T q$  et  $\|q\|^2$  sont bien des réelles également.

## II. Le problème est-il bien posé ?

### A. Autour de la fonction quadratique

Couples propres et conditionnement de  $X^T X$  :

7. A l'aide de la librairie *numpy*, calculer numériquement les couples propres ( $\lambda_1, v_1$ ) et ( $\lambda_2, v_2$ )  
(Utiliser le lien :[WikiMath-Eléments propres](#))

Nous allons utiliser la library numpy pour calculer les couples propres du vecteurs  $X^T X$ . Voici ci-après le code et son résultat :

```
lbd, v = np.linalg.eig(XTX)
print("Les couples propres sont les suivants", lbd[0], ", ", v[0]), (", lbd[1], ", ", v[1], ")")
```

Les couples propres sont les suivants 5.21086599162868 , [-0.98384923 -0.17899913] ), ( 1403.093864179935 , [ 0.17899913 -0.98384923] )

8. Déduire de la question 1.(a) le conditionnement de la matrice  $X^T X$  (Utiliser le lien :[Norme matricielle et conditionnement](#)). Comparer ce résultat à celui obtenu en calculant à l'aide de la librairie *numpy* le conditionnement de  $X^T X$  (Fonction : *numpy.linalg.cond*).

D'après le lien du sujet et des cours du semestre précédent on sait que l'on peut trouver le conditionnement d'une matrice grâce à ses vecteurs propres par la relation suivante :

$$cond(X^T X) = \left| \frac{\lambda_n}{\lambda_1} \right|$$

Ainsi en utilisant le résultat de la question précédente on a alors :

$$cond(X^T X) = \left| \frac{1403.1}{5.21} \right| = 269.3$$

Avec le code python que nous avons fait nous obtenons le résultat suivant :

```
cond = np.linalg.cond(XTX)
print("le conditionnement est cond(XTX) = ", cond)
```

le conditionnement est cond(XTX) = 269.263087255362

On remarque le par le calcul manuel et numérique les résultats sont similaires. De plus, on peut conclure que la matrice est très mal conditionnée puisque son conditionnement est éloigné de 1.

Quelques propriétés élémentaires de  $F$  :

### 9. Montrer que la fonction quadratique $F$ est définie positive

Nous avons la fonction  $F$  définie de la manière suivante :

$$F(c) = \frac{1}{2}c^T X^T X c - (X^T q)^T c + \frac{1}{2} \|q\|^2$$

Cette fonction est dite définie positive si et seulement si :  $0 < \lambda_1 < \lambda_2 < \dots < \lambda_n$

Or dans notre cas les valeurs propres nous donne :  $0 < \lambda_1 = 5.21 < \lambda_2 = 1403.1$ .

On peut alors conclure que la fonction  $F$  est bien définie positive.

### 10. Que peut-on dire de la coercivité et de la convexité de $F$ sur $\mathbb{R}^2$ ?

Comme nous avons démontré à la question précédente la fonction  $F$  est quadratique définie positive. De plus, grâce aux propriété du cours on peut montrer que  $f$  est coercive sur  $\mathbb{R}$ , ainsi que strictement convexe sur  $\mathbb{R}$  et qu'elle admet un minimiser global stricte sur  $\mathbb{R}$ .

### 11. Montrer que la fonction quadratique $F$ admet un unique point critique, notée $c^*$ , qui est l'unique solution des équations normales.

On pose :  $A = X^T X$  et  $b = X^T q$ .

Soit  $F$  la fonction quadratique précédente définie positive voir question 9. D'après la proposition "Unicité du point critique d'une fonction quadratique définie positive" du cours : comme  $F$  est définie positive alors elle admet un **unique** point critique noté  $c^*$ . Ce dernier est aussi l'unique solution du système :  $Ac = b$ .

C'est à dire ici :

$$(X^T X)c = X^T q$$

Fonctions partielles de  $F$  :

### 12. Donner l'expression analytique de la fonction partielle, notée $F_{a,d}$ , de $F$ en $a$ suivant $d$ .

Soit  $a$  un point de  $\mathbb{R}^n$  et  $d$  un vecteur de  $\mathbb{R}^n$ . La fonction partielle de  $F$  en  $a$  suivant  $d$  correspond à  $F(a+td)$ .

Ainsi on a :  $F(a + td) = \frac{1}{2}(a + td)^T A(a + td) - b^T(a + td) = c$

$$\Leftrightarrow = \frac{1}{2}[a^T A a + a^T a t d + d^T A a t + d^T A d t^2] - b^T a - b^T t d + c$$

$$\Leftrightarrow = \frac{1}{2}a^T A a - b^T a + c + \frac{1}{2}d^T A d t^2 + a^T A d t - b^T d t$$

$$\Leftrightarrow = \frac{1}{2}d^T A d t^2 - (b^T d - a^T A d)t + f(a)$$

$$\Leftrightarrow = \frac{1}{2}d^T A d t^2 - (b - A a)^T d t + f(a)$$

### 13. Montrer que $F_{a,d}$ est une fonction quadratique définie positive d'une variable.

Posons  $a = d^T \cdot A \cdot d$ ,  $b = (b - A \cdot a)^T \cdot d$  et  $c = f(a)$ .

Ainsi on peut écrire la fonction précédente sous la forme :

$$F_{a,d} = \frac{1}{2}at^2 + bt + c$$

De plus, la matrice  $A = X^T X$  étant définie positive et  $a \in \mathbb{R}^2$ ,  $(b, c) \in \mathbb{R}^2$  on peut ainsi en déduire que  $F_{a,d}$  est une fonction quadratique définie positive à une seule variable.

### 14. Que peut-on dire de la coercivité et de la convexité de $F_{a,d}$ sur $\mathbb{R}$ ?

On sait d'après la question précédente que  $F_{a,d}$  est une fonction quadratique définie positive. Donc d'après les propriétés du cours on peut déduire que  $F$  est coercive, strictement convexe et que la fonction admet un unique point critique  $t^*$  qui est son minimiseur global strict.

### 15. Expliciter l'unique minimiseur globale strict de $F_{a,d}$ . On le notera $t^*_{a,d}$ .

On sait grâce à la question 14. Que la fonction admet un minimiseur.

Ici donc :

$$\begin{aligned} t^*_{a,d} &= \frac{b}{a} \\ \Leftrightarrow &= \frac{(b - A \cdot a)^T \cdot d}{d^T \cdot A \cdot d} \end{aligned}$$

### 16. Préciser la nature géométrique de la courbe représentative de $F_{a,d}$ .

La fonction partielle  $F_{a,d}$  est une fonction quadratique définie positive. La courbe représentative de cette fonction partielle est une parabole sur  $\mathbb{R}$  strictement convexe de sommet  $(t^*_{a,d}, F_{a,d}(t^*_{a,d}))$  et d'axe de symétrie la droite d'équation  $t = t^*_{a,d}$ .

### 17. Tracer, sur un intervalle $[-10, 10]$ les fonctions partielles $F_{c^*,d}$ pour $d = e1, e2, v1$ et $v2$ .

On commence d'abord par trouver l'expression du point critique  $c^*$  de la fonction  $F$ .

On pose :  $Ac^* = b$  avec  $A = X^T X$  et  $b = X^T q$

On obtient les valeurs de  $A$  et  $b$  grâce à la question 5 et ainsi :

$$\Leftrightarrow = \begin{pmatrix} 50 & 246,17885986 \\ 246,1788598 & 1358,30473017 \end{pmatrix} \begin{pmatrix} c_1^* \\ c_2^* \end{pmatrix} = \begin{pmatrix} 53,23430273 \\ 271,44405179 \end{pmatrix}$$

$$\Leftrightarrow = \begin{cases} 50.c_1^* + 246,17885986.c_2^* = 53,23430273 \\ 246,17885986.c_1^* + 1358,30473017.c_2^* = 271,44405179 \end{cases}$$

Ainsi on trouve :

$$\Leftrightarrow c^* = \begin{pmatrix} 0,75016254 \\ 0,06388117 \end{pmatrix}$$

Ainsi nous pouvons à présent tracer les courbes des fonctions partielles à l'aide du code python suivant :

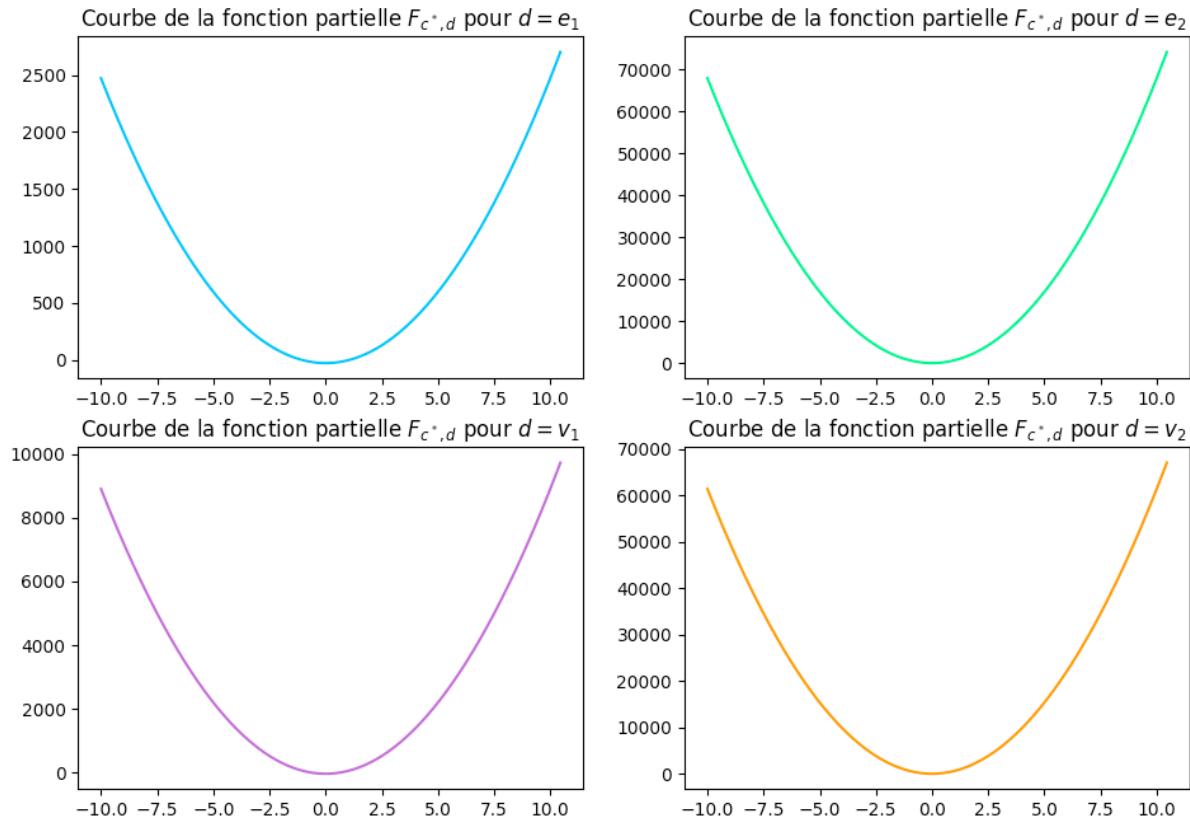
```

d1= np.array([[1],[0]])
d2 = np.array([[0],[1]])

def partial_function (d, cstar, A, b):
    t = np.arange(-10,10.5, 0.05)
    F_cstar = list()
    for i in t:
        f_cstar = (1/2)* (np.transpose(d)@A@d*(i**2))-np.transpose((b - A@cstar))@d*i + (1/2)*(np.transpose(cstar)@A@cstar)-np.transpose(b)@cstar
        F_cstar.append(f_cstar[0])
    return(F_cstar ,t)

F_cstar1, t= partial_function(d1, cstar,A,b)
F_cstar2, t2= partial_function(d2, cstar,A,b)
F_cstar3, t3= partial_function(v[0], cstar,A,b)
F_cstar4, t4= partial_function(v[1], cstar,A,b)
plt.figure("Partials functions")
plt.subplot(2,2,1)
plt.plot(t,F_cstar1, color ='deepskyblue')
plt.title("Courbe de la fonction partielle $F_{c^*,d}$ pour $d=e_1$")
plt.subplot(2,2,2)
plt.plot(t2,F_cstar2, color ='springgreen')
plt.title("Courbe de la fonction partielle $F_{c^*,d}$ pour $d=e_2$")
plt.subplot(2,2,3)
plt.plot(t3,F_cstar3, color ='mediumorchid')
plt.title("Courbe de la fonction partielle $F_{c^*,d}$ pour $d=v_1$")
plt.subplot(2,2,4)
plt.plot(t4,F_cstar4, color ='darkorange')
plt.title("Courbe de la fonction partielle $F_{c^*,d}$ pour $d=v_2$")
plt.show()

```



Carte de niveau :

18. Construire la matrice  $Z = X^T X = (Z_{ij})_{1 \leq i,j \leq 2} \in \mathbb{R}^{2 \times 2}$ , le vecteur  $w = X^T q = (w_1, w_2)^T \in \mathbb{R}^2$  et le scalaire  $s = q^T q \in \mathbb{R}$ .

Dans un premier temps on part de l'expression (6), en sachant que  $\|q\|^2 = q^T q = s$  et que  $w = X^T q$ .

On a alors :

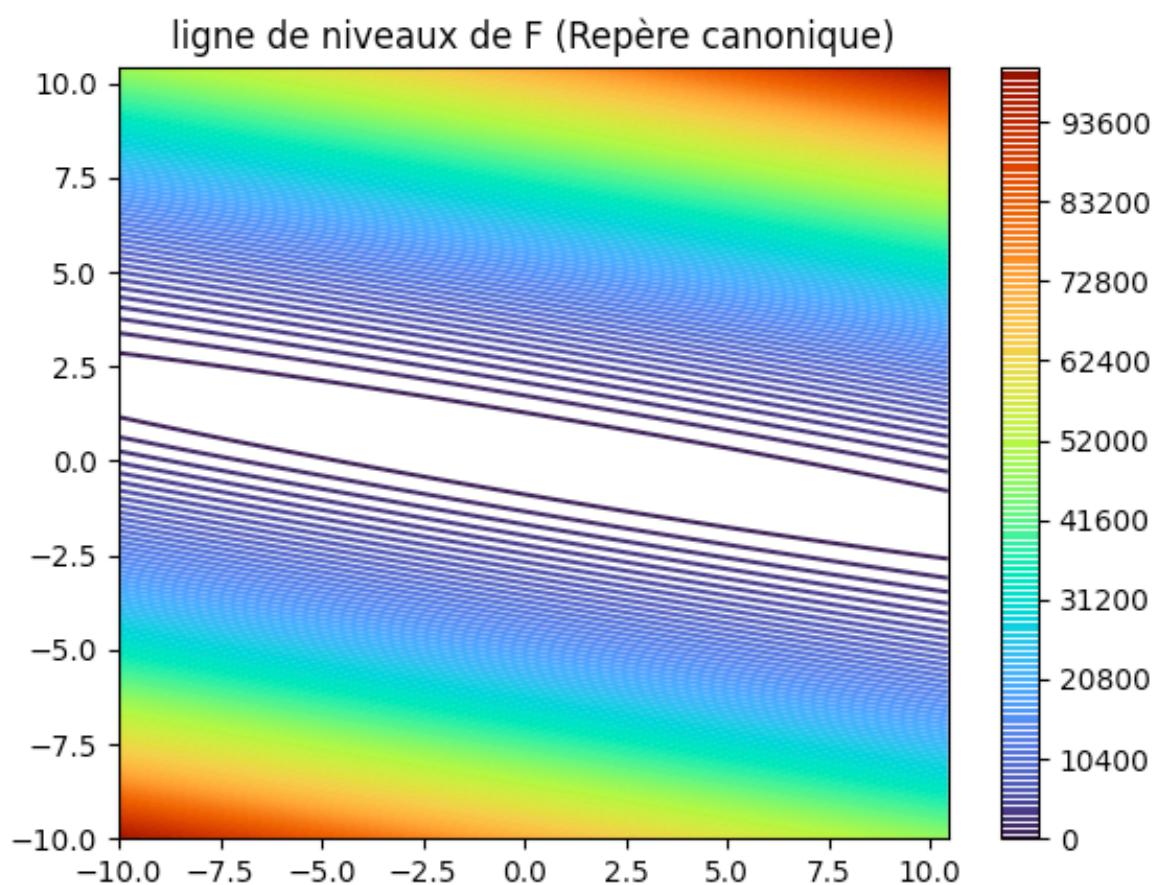
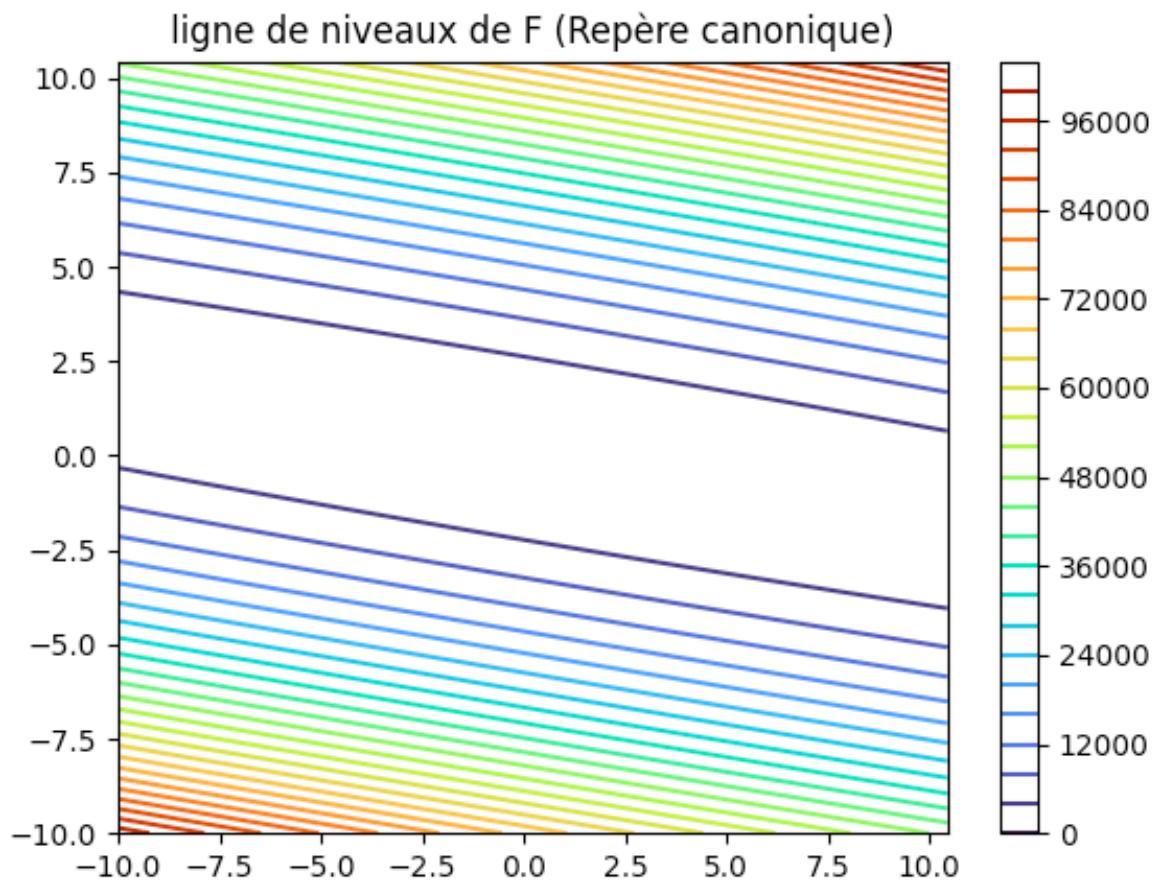
$$\begin{aligned} F(c) &= \frac{1}{2} c^T X^T X c - (X^T q)^T c + \frac{1}{2} \|q\|^2 \\ F(c) &= \frac{1}{2} c^T Z c - w^T c + \frac{1}{2} s \\ F(c) &= \frac{1}{2} (c_1 \ c_2) Z \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} - w^T \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} + \frac{1}{2} s \\ F(c) &= \frac{1}{2} (Z_{1,1} \cdot c_1^2 + 2 \cdot Z_{1,2} \cdot c_1 \cdot c_2 + Z_{2,2} \cdot c_2^2 - 2(w_1 \cdot c_1 + w_2 \cdot c_2) + s) \end{aligned}$$

On retrouve bien la formule que l'on souhaitait. Voici ci-après le code python permettant de construire les vecteurs demandés. On s'est servi de notre fonction général `X_build` pour construire ces vecteurs :

```
X,q,Z,w = X_build(x,y)
s = np.transpose(q)@q
```

19. On considère un pas de discrétilisations  $\delta = 0.5$ . Définir le pavé  $[-10, 10] \times [-10, 10]$  avec un pas  $\delta$  dans chaque direction. Puis, afficher les courbes de niveau de  $F$  dans le pavé  $[-10, 10] \times [-10, 10]$ .

```
x_axis = np.arange(-10,10.5,0.05)
y_axis = np.arange(-10,10.5,0.05)
xx_axis, yy_axis = np.meshgrid(x_axis,y_axis)
F = list()
f = (1/2)*(Z[0,0]*xx_axis**2 + 2*Z[0,1]*xx_axis*yy_axis+Z[1,1]*yy_axis**2 - 2*(w[0]*xx_axis+w[1]*yy_axis)+s)
plt.figure("Level map")
LM = plt.contour(xx_axis,yy_axis,f,30, cmap = "rainbow")
plt.title("lignes de niveaux de F (Repère canonique)")
plt.colorbar(LM)
plt.legend()
```



20. Donner la forme réduite (ou canonique), notée  $F$ , de la fonction quadratique  $F$  en précisant le repère de réduction, noté  $V^*$ .

Considérons le repère de réduction  $V^* = (\Omega, V)$  où le représentant matriciel de l'origine est :

$\begin{pmatrix} c_1^* \\ c_2^* \\ 0 \end{pmatrix} = \Omega$ . On note aussi  $(v_1, v_2)$  les vecteurs propres de  $Z$  associé aux deux valeurs propres  $(\lambda_1, \lambda_2)$  dans le repère canonique. L'expression canonique matricielle s'écrit donc de la forme suivante :

$$F(c^* + Py) = F_{V_{c^*}}(y) = \frac{1}{2}y^T Dy + F(c^*)$$

21. Expliciter la carte de niveaux de  $F$ .

Expliciter la carte de niveaux revient à expliciter les équations cartésienne de ses lignes de niveaux. Pour expliciter celles de notre fonction nous allons appliquer le théorème 4.1 du cours : Soit la fonction  $F$  précédente quadratique définie positive d'expression matricielle :

$$f(x) = \frac{1}{2}x^T Ax - b^T x + c$$

On désigne aussi le repère orthonormé  $V_{c^*} = (\Omega, V)$  tel que :

- $\Omega$  est le point de représentant matriciel  $c^* := A^{-1}b$  dans le repère canonique de  $\mathbb{R}^N$ ;
- $V := (v_1, \dots, v_n)$  la base orthonormé des vecteurs propres de  $A = X^T X$  tel que chaque vecteur propre est associé à une valeur propre  $\lambda$ .

Ainsi la carte de niveaux de  $f$  est donné par :

- (1) Si  $\beta < f(c^*)$ , alors  $L_\beta(f) = \emptyset$ .
- (2) Si  $\beta = f(c^*)$ , alors  $L_\beta(f) = x^*$
- (3) Si  $\beta > f(c^*)$ , alors  $L_\beta(f)$  est la ligne de niveau d'équation cartésienne dans le repère  $V_{c^*}$  :

$$\begin{aligned} \sum_{i=1}^N \frac{y_i^2}{(\sqrt{\frac{2(\beta-f(c^*))}{\lambda_i}})^2} &= 1 \\ \Leftrightarrow \sum_{i=1}^2 \frac{y_i^2}{(\sqrt{\frac{2(\beta-f(c^*))^2}{\lambda_i}})} &= 1 \\ \Leftrightarrow \frac{y_1^2}{(\sqrt{\frac{2(\beta-f(c^*))}{\lambda_1}})^2} + \frac{y_2^2}{(\sqrt{\frac{2(\beta-f(c^*))}{\lambda_2}})^2} &= 1 \\ \Leftrightarrow \frac{y_1^2}{(\sqrt{\frac{2(\beta+28.63727104)}{5.21}})^2} + \frac{y_2^2}{(\sqrt{\frac{2(\beta+28.63727104)}{1403.1}})^2} &= 1 \\ \Leftrightarrow \frac{y_1^2}{(\sqrt{2(\beta+5.49)})^2} + \frac{y_2^2}{(\sqrt{2(\beta+0.02)})^2} &= 1 \end{aligned}$$

Ainsi on peut voir que les courbes de niveau sont des ellipses centrées sur le point  $\Omega$ , dont les demi-axes sont les droites passant par ce point telles-que :

$(\sqrt{2(\beta + 5.49)})^2$  est le demi grand axe et  $(\sqrt{2(\beta + 0.02)})^2$  est le demi petit axe associés respectivement aux vecteurs propres  $v_1$  et  $v_2$ .

22. Les résultats obtenues par le calcul de la question 4).(d) coïncident-ils avec le tracé des courbes de niveau de la question 4.(b).

En observant le résultat des lignes de niveaux calculées précédemment et les courbes obtenues via python, on remarque bien les ellipses explicitées. La vision de ces ellipses sur les graphiques dépendent essentiellement du repère choisi. En effet, le demi grand axe étant très nettement supérieur au demi petit axe il faudrait changer de repère pour visualiser correctement les ellipses des lignes de niveaux.

23. Donner une caractérisation géométrique (à partir des lignes de niveau de  $F$ ) du conditionnement de la matrice  $X^T X$ .

Comme nous l'avons annoncé à la question 8 le conditionnement de la matrice  $Z = X^T X$  est très très mauvais. De plus, on remarque par les lignes de niveaux sur les graphiques et celles calculées que les ellipses sont très aplatis. Néanmoins, on sait qu'une matrice correctement conditionnée possède des lignes de niveaux circulaires.

De cela, nous pouvons conclure que plus le conditionnement est mauvais plus les lignes de niveaux sont aplatis et ainsi on en conjecture bien notre première conclusion : la matrice  $Z$  est très mal conditionnée.

*Surface représentative :*

24. Tracer la fonction  $F$  dans le pave  $[-10,10] \times [-10, 10]$ .

Nous avons tracé cette surface représentative de la même manière que pour tracer les lignes de niveaux. Nous avons intégrer les deux tracés dans la même fonction. Voici le code ainsi que le résultat obtenu :

```

def level_map(xit_, xit2_, xit3_):
    X,q,Z,w = X_build(x,y)
    s = np.transpose(q)@q

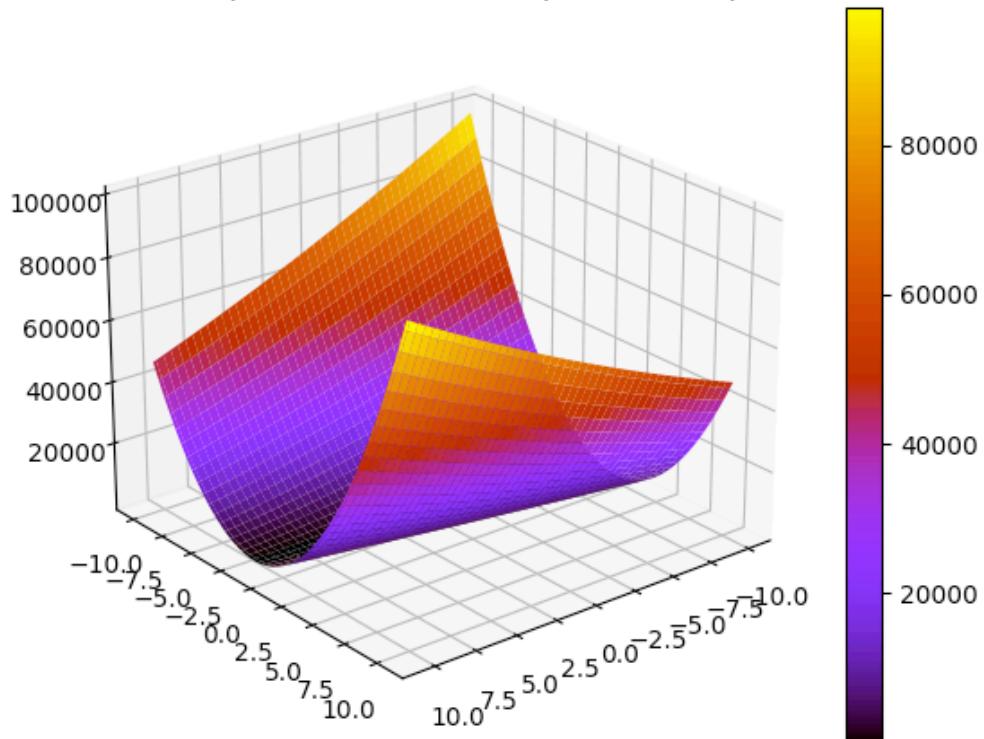
    x_axis = np.arange(-10,10.5,0.05)
    y_axis = np.arange(-10,10.5,0.05)
    xx_axis, yy_axis = np.meshgrid(x_axis,y_axis)
    F = list()
    f = (1/2)*(Z[0,0]*xx_axis**2 + 2*Z[0,1]*xx_axis*yy_axis+Z[1,1]*yy_axis**2 - 2*(w[0]*xx_axis+w[1]*yy_axis)+s)
    plt.figure("Level map")

    LM = plt.contour(x_axis,y_axis,f,30, cmap = "turbo")
    plt.title("ligne de niveaux de f (Repère canonique)")
    plt.colorbar(LM)
    #plt.legend()

    fig = plt.figure("Level map 2")
    ax3d = fig.gca(projection = "3d")
    LM_2 = ax3d.plot_surface(xx_axis, yy_axis,f, cmap = "gnuplot", edgecolor ='none')
    plt.title("Surface représentative de f (Repère canonique)")
    plt.colorbar(LM_2)
    plt.show()

```

Surface représentative de  $f$  (Repère canonique)



## 25. Précisez les caractéristiques géométrique de la surface représentative, notée $S_F$ , de $F$ .

Les caractéristiques géométriques de la surface représentative  $S_F$  de  $F$  sont les suivantes :

- Des tranches horizontales : la carte de niveau est formée d'ellipses centrées au point critique  $c^*$  de  $F$ .
- Des tranches verticales : les paraboles strictement convexes d'axe de symétrie la droite,  $D_{\Omega^+, e_3}$ , et dirigé par le vecteur unitaire  $e_3$ .
- Un centre noté  $\Omega^+$ .
- Un axe de symétrie noté  $D_{\Omega^+, e_3}$ .

## 26. Pourquoi dit-on que $S_F$ est un paraboloïde elliptique ?

On dit que la surface représentative  $S_F$  est un paraboloïde elliptique car ses tranches verticales sont des paraboles et ses tranches horizontales des ellipses comme nous l'avons montré précédemment.

## B. Unicité de la solution $Q$

### 27. Montrer que l'unique solution du problème d'optimisation est l'unique point critique $c^*$ (son représentant relativement à le repère canonique $E_0$ ) de $F$ .

Soient les vecteurs  $c^*$  et  $y^*$  respectivement les représentant du repère canonique et du repère  $V_{c^*}$  de l'unique point critique de  $F$ . Admettons :

$$f(x^*) = f_{V_{c^*}}(y^*) = f_{V_{c^*}}(0)$$

De la même manière : soient  $x$  et  $y$  les représentant matricielle dans les deux repère respectivement. On a :  $c = c^* + Py$  avec  $P$  la matrice de passage entre les deux bases liées aux repères. On a alors :  $f(x) = f_{V_{c^*}}(y) = \sum_{i=1}^N \lambda_i y_i^2 + f(c^*)$

Donc :

$$\begin{aligned} \forall c \neq c^*; f(c) - f(c^*) &= f_{V_{c^*}}(y) = \sum_{i=1}^N \lambda_i y_i^2 \\ \Leftrightarrow f(c) - f(c^*) &= f_{V_{c^*}}(y) = \frac{1}{2} \sum_{i=1}^N \lambda_i y_i^2 = \frac{1}{2}(\lambda_1 y_1^2 + \lambda_2 y_2^2) > 0 \end{aligned}$$

Cette expression nous permet de dire que :  $f(c) > f(c^*)$  et donc dire que  $c^*$  est le minimiseur global stricte et unique de  $F$ .

### 28. En utilisant la question précédente, préciser le problème équivalent à notre problème de minimisation quadratique définie positive ( $Q$ )

Notre problème consiste à la détermination du minimiseur globale de notre fonction  $F$ . Ce minimiseur étant un point critique de notre fonction on a :  $f'(c^*) = 0$

Alors:

$$\begin{aligned}
f'(c^*) &= 0 \\
\Leftrightarrow (X^T X)c^* - X^T q &= 0 \\
\Leftrightarrow (X^T X)c &= X^T q \\
\Leftrightarrow Ac &= b
\end{aligned}$$

On a donc une équation de la forme  $Ax = b$ . C'est à dire que déterminer le minimiseur global de notre fonction revient à résoudre un système de la forme  $Ax = b$ .

Dans cette question nous allons expliciter  $c^*$ , le représentant dans le repère canonique  $E_0$  de l'unique solution de  $(Q)$

### 29. Expliciter $c^*$ en fonction de $q, p$ et $1$

De l'équation précédente on peut dire que :  $c^* = (X^T X)^{-1} X^T q$

Cela nous permet de dire :  $q = Xc^*$

Et ainsi :

$$\begin{aligned}
&\Leftrightarrow \begin{pmatrix} q_1 \\ \vdots \\ q_m \end{pmatrix} = \begin{pmatrix} 1 & p_1 \\ \vdots & \vdots \\ 1 & p_m \end{pmatrix} \begin{pmatrix} c_1^* \\ c_2^* \end{pmatrix} \\
&\Leftrightarrow \begin{cases} c_1 + c_2 p_1 &= q_1 \\ \vdots &=: \\ c_1 + p_m c_2 &= q_m \end{cases}
\end{aligned}$$

On a donc finalement la relation suivante :  $q_i = c_1 + p_i c_2$

### 30. Calculer numériquement $c^*$ à partir de son expression établie à la question précédente

À l'aide de python nous avons calculer grâce à la library numpy la solution du système  $Ac = b$  comme ci-après :

```
X,q,XTX,XTq = X_build(x,y)
A = XTX
b = XTq

cstar = np.linalg.solve(A,b)
print(cstar)
```

```
[[0.75016254]
 [0.06388117]]
```

## C. Conditionnement du problème

31. Montrer que l'on a :  $\|\delta c^*\| \leq \|(X^T X)^{-1}\| \cdot \|\delta(X^T q)\|$

Posons  $A = X^T X$  et  $b = X^T q$  et explicitons le système suivant :  $Ac^* = b$ . Ce style de problème est récurrent. On cherche à étudier les variations de  $A$  ou de  $b$  par rapport à la solution du système. Supposons la variation  $b'$  de  $b$  défini par :  $b' = b + \delta(b)$ . Où  $\delta(b)$  est la différence entre  $b'$  et  $b$  et est de norme faible. On a alors le système :  $Ac' = b'$  ayant une uniquement solution.

Supposons maintenant le système :  $\begin{cases} Ac^* = b & (1) \\ Ac' = b' & (2) \end{cases}$ .

Nous faisons maintenant la soustraction de 2 avec 1 et on obtient :

$$\left\{ \begin{array}{l} A(c' - c^*) = b' - b \\ \Leftrightarrow (c' - c^*) = A^{-1}(b' - b) \\ \Leftrightarrow (c' - c^*) = (X^T X)^{-1}(b' - b) \end{array} \right.$$

D'après les expressions précédentes on a :  $\delta(b) = b' - b$  et posons  $\delta(c^*) = c' - c$ .

On obtient donc :  $\delta(c^*) = (X^T X)^{-1}\delta(b)$

En appliquant les règles de l'inégalité triangulaire on a alors :

$$\left\{ \begin{array}{l} \|\delta(c^*)\| \leq \|(X^T X)^{-1}\| \cdot \|\delta(b)\| \\ \Leftrightarrow \|\delta(c^*)\| \leq \|(X^T X)^{-1}\| \cdot \|\delta(X^T q)\| \end{array} \right.$$

32. En utilisant la majoration précédente établir la majoration  $\frac{\|\delta c^*\|}{\|c^*\|} \leq \text{cond}_2(X^T X) \frac{\|\delta(X^T q)\|}{\|X^T q\|}$

Avec l'expression du système de la question précédente on peut exprimer le système suivant :

$$\left\{ \begin{array}{l} Ac^* = b \\ \|\delta(c^*)\| \leq \|(X^T X)^{-1}\| \cdot \|\delta(X^T q)\| \end{array} \right.$$

En appliquant comme précédemment l'inégalité triangulaire on a :

$$\left\{ \begin{array}{l} \|b\| \leq \|A\| \cdot \|c^*\| \\ \|\delta(c^*)\| \leq \|(X^T X)^{-1}\| \cdot \|\delta(X^T q)\| \end{array} \right.$$

En faisant la division des deux expressions nous obtenons la formule suivante :

$$\frac{\|\delta c^*\|}{\|c^*\|} \leq \|(X^T X)^{-1}\| \cdot \|X^T X\| \cdot \frac{\|\delta(X^T q)\|}{\|X^T q\|}$$

Or rappelons la formule du conditionnement d'une matrice  $A$  :  $\text{cond}_2(A) = \|(A)^{-1}\| \cdot \|A\|$

Nous avons alors l'expression suivante :

$$\frac{\|\delta c^*\|}{\|c^*\|} \leq \text{cond}_2(X^T X) \cdot \frac{\|\delta(X^T q)\|}{\|X^T q\|}$$

33. Que pouvez-vous conclure sur la sensibilité (conditionnement) du problème Q.

Sachant que  $\frac{||\delta c^*||}{||c^*||}$ , l'erreur relative de la solution  $c^*$ , est majoré par le conditionnement de la matrice  $X^T X$  multiplié par l'erreur relative de la matrice  $X^T q$ , alors plus la matrice  $X^T X$  est mal conditionné moins la valeur approchée obtenue sera exacte et fiable . Il en va de même concernant  $X^T q$ .

# Méthodes à directions de descente : une étude numérique

## I. Principes généraux

### A. Direction de descente

34. Rappeler la définition d'une direction de descente de la fonction  $f$  au point  $x_k$ .

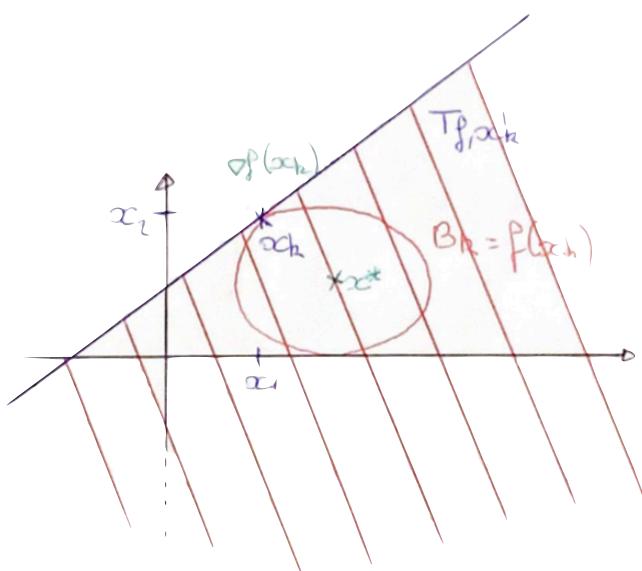
Soit  $f : \mathbb{R}^N \rightarrow \mathbb{R}$  une fonction différentiable. Soient  $x, d \in \mathbb{R}^N$ .

La direction de descente  $d$  est une direction de descente au point  $x_k$  si :

$$d^T \nabla f(x_k) < 0$$

35. Faire une figure (même à main levée) dans le plan de représentation des variables  $x_1$  et  $x_2$  : représentant les courbes de niveaux passant par le point  $x_k$  respective de la fonction  $f$  et de son approximation tangentielle linéaire  $T_f(x_k)$  au point  $x_k$ ; représenter le vecteur  $\nabla f(x_k)$ ; colorier ou hachurer en rouge le demi-espace contenant les directions de descente de  $f$  au point  $x_k$ .

L'équation de notre problème est une équation quadratique définie positive. D'après les propriétés du cours on peut ainsi affirmer que  $f$  admet un seul et unique minimiseur global strict  $x^*$  sur  $\mathbb{R}$ . Nous avons alors une seule courbe de niveau passant par le point  $x_k$  qui est une ellipse ayant pour centre le minimiseur  $x^*$ . On notera cette courbe  $\beta_k = f(x_k)$  :



## B. Pas de descente

36. Dans cette question nous allons montrer le lemme qui justifie l'existence d'une infinité de pas de descentes pour une direction de descente calculée. L'objectif est de réaliser deux figures qui permettent d'interpréter le résultat du lemme 3.1.2. Dans ce contexte faites les deux figures suivantes :

- **Figure 1** Dans le plan de représentation des variables  $x_1$  et  $x_2$  : 1) tracer les courbes de niveaux passant par le point  $x_k$  respective de la fonction  $f$  et de son approximation tangentielle linéaire  $T_f, x_k$  au point  $x_k$ ; 2) représenter le vecteur  $\nabla f(x_k)$  et la direction de descente  $d_k$  de  $f$  en  $x_k$ ; 3) placer le point  $x_k + \tilde{\alpha}_k d_k$ ; 4) Le point  $x_k + \alpha_k d_k$  où  $\alpha_k$  est un pas de descentes de  $f$  en  $x_k$  suivant  $d_k$ .
- **Figure 2** Dans le plan de représentation d'une fonction d'une variable : 1) tracer la courbe représentative de la fonction partielle  $f_{x_k, d_k}$ ; 2) placer le point de  $f$  d'abscisse  $\tilde{\alpha}_k$ ; 3) placer le point de  $f$  d'abscisse  $\alpha_k$  correspondant à un pas de descentes de  $f$  en  $x_k$  suivant  $d_k$ .

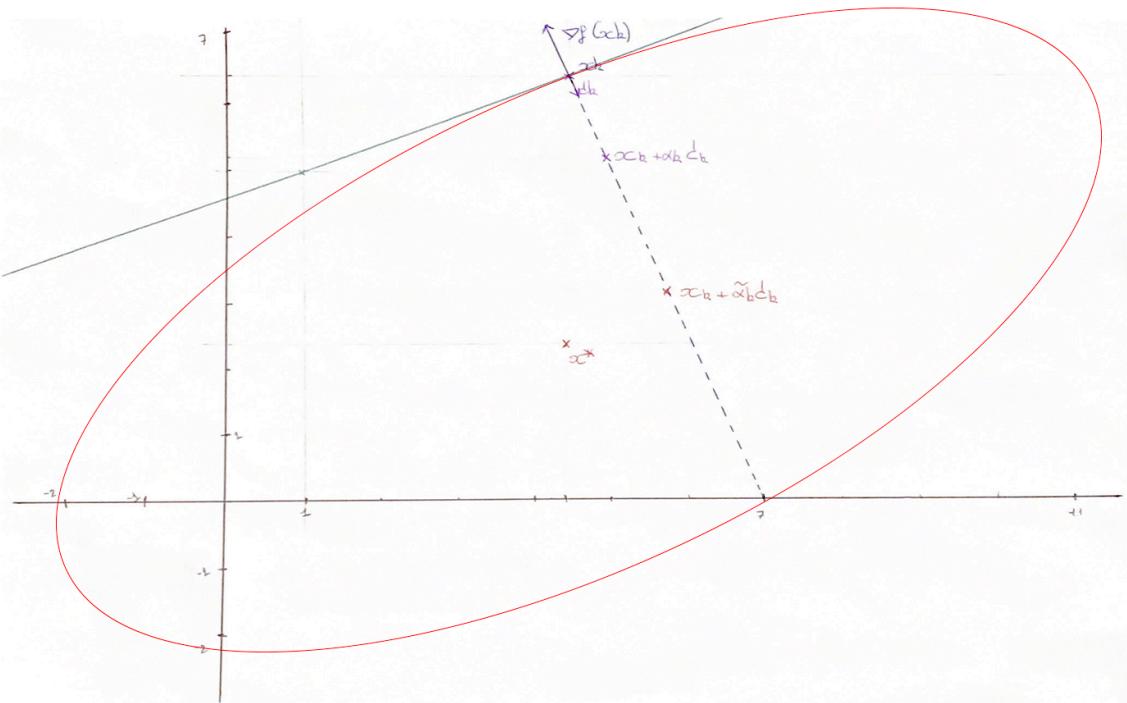


Figure 1

37. Rappeler (sans démonstration) l'expression de la fonction partielle  $f_{x_k, d_k}$  de  $f$  au point  $x_k$  dans la direction  $d_k$ .

On rappelle l'expression de la fonction partielle  $f_{x_k, d_k}$  de  $f$  au point  $x_k$  dans la direction  $d_k$  :

$$\forall t \in \mathbb{R}^N, f_{x_k, d_k}(t) = f(x_k + d_k t) = \frac{1}{2} d_k^T A d_k t^2 - (A x_k - b)^T \cdot d_k t + f(x_k)$$

38. Déterminer le réel  $\alpha_k > 0$  tel que  $f_{x_k, d_k}(\alpha_k) = f_{x_k, d_k}(0)$ .

Posons alors :

$$f_{x_k, d_k}(0) = \frac{1}{2} d_k^T A d_k \times 0^2 - (A x_k - b)^T \cdot d_k \times 0 + f(x_k) = f_{x_k, d_k}(\tilde{\alpha}) = \frac{1}{2} d_k^T A d_k \times \tilde{\alpha}^2 - (A x_k - b)^T \cdot d_k \times \tilde{\alpha} + f(x_k)$$

On a alors :

$$\left\{ \begin{array}{l} \Leftrightarrow \frac{1}{2} d_k^T A d_k \tilde{\alpha}^2 - (A x_k - b)^T d_k \tilde{\alpha} + f(x_k) = f(x_k) \\ \Leftrightarrow \frac{1}{2} d_k^T A d_k \tilde{\alpha}^2 = (A x_k - b)^T d_k \tilde{\alpha} \\ \Leftrightarrow \frac{1}{2} d_k^T A d_k \tilde{\alpha} = (A x_k - b)^T d_k \\ \Leftrightarrow \tilde{\alpha} = 2(A x_k - b)^T \cdot d_k \cdot (d_k^T \cdot A \cdot d_k)^{-1} \end{array} \right.$$

39. Déduire de la question précédente la proposition 3.1.

D'après la question précédente on a :  $\tilde{\alpha} = -2(A x_k - b)^T \cdot d_k \cdot (d_k^T \cdot A \cdot d_k)^{-1}$   
C'est à dire aussi :

$$\tilde{\alpha} = -\frac{2(A x_k - b)^T \cdot d_k}{(d_k^T \cdot A \cdot d_k)}$$

C'est bel et bien ce qu'on devait obtenir.

40. Montrer que  $\alpha_k^*$  défini par l'expression (10) est le minimiseur global strict de  $f_{x_k, d_k}$ .

Dans un premier temps on rappelle que la fonction  $f_{x_k, d_k}$  est une fonction quadratique définie positive ayant pour expression :

$$x \mapsto \frac{1}{2} a x^2 - b x + c$$

De plus d'après les propriétés du cours sur les fonctions quadratiques, on sait que si cette fonction est définie positive elle admet un minimiseur global strict et unique, noté  $x^*$ , solution de l'équation :  $A x = b$ . Ce minimiseur admet pour expression :

$$x^* := \frac{b}{a}$$

En reprenant l'équation de  $f$  nous avons. :

$$f_{x_k, d_k}(t) = \frac{1}{2} d_k^T A d_k t^2 - (A x_k - b)^T \cdot d_k t + f(x_k)$$

On pose à présent :  $a = \frac{1}{2}d_k^T A d_k$  et  $b = -(A x_k - b)^T \cdot d_k$

En remplaçant dans l'expression du minimiseur exposée plus tôt on obtient pour un minimiseur  $\tilde{\alpha}_k^*$  l'expression suivante :

$$\alpha_k^* = -\frac{(A x_k - b)^T \cdot d_k}{d_k^T A d_k}$$

41. Montrer que  $\alpha_k^* > 0$ , et conclure.

Nous cherchons à démontrer que le minimiseur  $\alpha_k^*$  est positif strictement. On sait d'abord grâce au cours que :  $-(A x_k - b)^T d_k = -\nabla f(x_k)^T d_k = -D_{d_k} f(x_k) > 0$

De plus, la direction de descente est strictement négative. Cela signifie que le signe moins l'a rend strictement positive.

On rappelle aussi que la matrice A est une matrice symétrique définie positive c'est à dire donc :  $A > 0 \Rightarrow d_k^T A d_k > 0$ . On conclut alors que  $\alpha_k^*$  est strictement positif. En effet, tout quotient d'entités positive est une entité positive.

42. Quel est la position du  $(k+1)$ -ième itéré  $x_{k+1} = x_k + \alpha_k^* d_k$  relativement au segment  $[x_k, x_k + \tilde{\alpha}_k d_k]$  ou  $\alpha_k^*$  est défini par (7) (justifier rigoureusement vos propos en considérant les propriétés de la courbe représentative de la fonction partielle  $f(x_k, d_k)$ ).

43. Réaliser les figures de la question 3 item (a) dans le cas où  $\alpha_k$  désigne le pas optimal de f en  $x_k$  suivant  $d_k$ .

## C. Critère d'arrêt

44. Rappeler la définition du critère du résidu.

Les résidus ou "erreurs observées" sont définis comme étant les différences entre les valeurs observées et les valeurs estimées par un modèle de régression, ils ont la particularité de représenter la partie non expliquée par l'équation de regression.

45. Donner les deux interprétations de ce critère : l'une liée à la résolution du problème de minimisation quadratique (G), l'autre associée au problème équivalent de résolution du système linéaire inversible  $Ax = b$ .

Lors de la résolution du problème de minimisation quadratique ce critère correspond à l'erreur constaté entre la du minimiseur,  $c^*$ , de f calculé et le minimiseur théorique.

Pour ce qui est de la résolution du système linéaire inversible  $Ax = b$ , il nous permet d'avoir une information sur la différence (l'erreur) entre le  $x^*$  calculé et le théorique.

46. Soit  $x_k$  le  $k$ -ième itéré générée par l'algorithme itératif dont la  $k$ -ième itération est défini par (8). En réinterprétant la majoration établit dans le cas particulier du problème d'optimisation quadratique sans contrainte (Q), montrer que :  $\forall k \in \mathbb{N}, \frac{\|x_k - x^*\|}{\|x^*\|} \leq \text{cond}_2(A) \frac{\|r_k\|}{\|b\|}$ , où  $r_k$  désigne le résidu associé au système linéaire  $Ax = b$  de l'itération  $x_k$ .

Reprendons la démonstration de l'erreur relative énumérée à la question 32 :

$$\frac{\|\delta x^*\|}{\|x^*\|} \leq \text{cond}_2(A) \cdot \frac{\|\delta(b)\|}{\|Xb\|}$$

On pose dans un premier temps :  $\|\delta x^*\| = \|x_k - x^*\|$  et en introduisant le résidu on note la nouvelle égalité :  $Ax_k = b + r_k$ .

On obtient alors :

$$\begin{cases} Ax^* = b & (1) \\ Ax_k = b + r_k & (2) \end{cases}$$

En soustrayant 1 à 2 on a :

$$\begin{cases} A(x_k - x^*) = b + r_k - b \\ \Leftrightarrow (x_k - x^*) = A^{-1}(r_k) \\ \Leftrightarrow (\delta(x^*)) = (A)^{-1}(r_k) \end{cases}$$

En utilisant les mêmes méthodes qu'aux questions 32 et 33, ainsi que l'inégalité triangulaire on obtient alors :

$$\frac{\|x_k - x^*\|}{\|x^*\|} \leq \text{cond}_2(A) \frac{\|r_k\|}{\|b\|}$$

47. À partir de l'inégalité (20), que peut-on dire sur la qualité du critère dans le contexte de la résolution du problème (G), et par conséquent sur celle de l'approximation calculée.

Aux regards des majorations établies dans le cas du problème d'optimisation quadratique, on pose alors que  $A = X^T X$ . En voyant le conditionnement que l'on a calculé lors des phases préliminaires de ce projet, le critère et, par conséquent, l'approximation sont de très mauvaises qualités. L'écart relatif est majoré par une valeur de conditionnement élevée ce qui ne permet en aucun cas de garantir un résultat précis et proche de celui théorique.

## II. Méthode de descente du gradient ou de plus forte descente

### A. Direction de plus forte descente

48. Montrer que  $d_k := -\nabla f(x_k)$  est bien une direction de descente de  $f$  au point  $x_k$ .

Soit  $d_k = -\nabla f(x_k) \neq 0$ . On cherche donc à montrer qu'il existe  $\alpha_0 > 0$  tel que si :  $\alpha \in ]0, \alpha_0[$  on a :  $f(x + \alpha d_k) < f(x)$ .

Posons

$$\varphi(\alpha) = f(x + \alpha d_k)$$

Notre problème revient alors à montrer :  $\varphi(\alpha) < \varphi(0)$

On cherche alors :

$$\varphi'(\alpha) = \nabla f(x_k) \times d_k = -\|\nabla f(x_k)\|^2 < 0$$

Comme  $\varphi'$  est continue et décroissante, d'après le théorème des valeurs intermédiaires on a alors :

$$\alpha_0 > 0 \mid \alpha \in [0, \alpha_0] \mid \varphi'(\alpha) < 0$$

On a alors :

$$\varphi(\alpha) - \varphi(0) = \int_0^\alpha \varphi'(t) dt < 0$$

On a donc bien :  $\varphi(\alpha) < \varphi(0)$  ce qui nous prouve bien que  $d_k$  est une direction de descente stricte en  $x_k$

49. Montrer que :  $\forall d \in \mathbb{R}^2; \|d\| = \|\nabla f(x_k)\|, d^T \nabla f(x_k) \leq \|\nabla f(x_k)\|^2$

Soit  $d = -\nabla f(x_k) \neq 0$  on a aussi  $\varphi'(\alpha) = \nabla f(x_k) \times d_k = -\|\nabla f(x_k)\|^2 < 0$ . On en déduit alors que :  $\forall d \in \mathbb{R}^2; \|d\| = \|\nabla f(x_k)\|, d^T \nabla f(x_k) \leq \|\nabla f(x_k)\|^2$

50. Déduire de (24) l'inégalité (23)

On a  $\|d\| = \|d_k\|$  c'est à dire également  $\|d\| = \|\nabla f(x_k)\|$ . On en déduit alors que :

$$\forall d \in \mathbb{R}^2; \|d\| = \|d_k\|, d_k^T \nabla f(x_k) \leq d^T \nabla f(x_k)$$

### B. L'algorithme de descente du gradient à pas fixe

51. Justifier les différentes étapes de l'algorithme 3.2

Étape 1 : On fixe la condition d'arrêt avec le critère de résidu

Étape 2 : On explicite la direction de descente  $d_k$

Étape 3 : On explicite le pas, fixe,  $\alpha_k$

Étape 4 : on effectue le calcul du k+1 itéré selon l'équation :  $x_{k+1} = x_k - \alpha r_k$

52. Montrer que si le réel  $\alpha$  vérifie la condition  $\forall k \in \mathbb{N}, 0 < \alpha < \tilde{\alpha}_k := \frac{2||r_k||^2}{r_k^T X^T X r_k}$ , alors l'algorithme à pas fixe  $\alpha$  est bien une méthode à direction de descente.

Soit  $d_k = -\nabla f(x_k) \neq 0$ . On a pour la cas du pas fixe :

$$\begin{aligned} \forall k \in \mathbb{R}, \alpha_k &= \alpha \\ f(x_k + \alpha d_k) &< f(x_k) \\ 0 < \alpha &< \frac{2||r_k||^2}{r_k^T X^T X r_k} \end{aligned}$$

53. Montrer que  $\forall k \in \mathbb{N}, \tilde{\alpha}_k := \frac{2||r_k||^2}{r_k^T X^T X r_k} \geq \frac{2}{\lambda_2}$

Comme nous l'avons depuis le début de ce projet, notre matrice  $A$  est symétrique définie positive et admet 2 valeurs propres :  $\lambda_1$  et  $\lambda_2$ . De plus cette matrice est alors diagonalisable de la forme :  $A = PDP^{-1}$ .

On peut alors expliciter  $D$ , de la forme :  $D = P^T A P = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$

On définit à présent une matrice  $C$  telle que :  $C = PY$

Et on réécrit la forme du terme quadratique en intégrant  $C$ , on obtient alors l'égalité suivante :

$$\begin{aligned} C^T AC &= Y^T P^T A P Y \\ &= Y^T D Y \\ &= \lambda_1 y_1^2 + \lambda_2 y_2^2 \end{aligned}$$

On a alors que  $\forall f \in \mathbb{N}$  et ainsi on obtient :

$$\begin{aligned} \lambda_1 ||r_k||^2 &\leq r_k^T A r_k \leq \lambda_2 ||r_k||^2 \\ \Rightarrow \frac{2||r_k||^2}{r_k^T A r_k} &\geq \frac{2}{\lambda_2} \\ \Rightarrow \frac{2||r_k||^2}{r_k^T X^T X r_k} &\geq \frac{2}{\lambda_2} \end{aligned}$$

54. Déduire de la question précédente la conclusion de la proposition d'existence du pas fixe.

En combinant les résultats des deux questions précédemment traitée on obtient l'inégalité suivante :

$0 < \alpha < \frac{2||r_k||^2}{r_k^T X^T X r_k} \leq \frac{2}{\lambda_2}$ , Ainsi d'après les théorèmes mathématiques de comparaison on peut affirmer trivialement que :

$$0 < \alpha < \frac{2}{\lambda_2}$$

55. Programmer une fonction Python  $sol,xit,nit = gradientPasFixe(A,b,x0,rho,tol)$  pour résoudre le problème  $Ax = b$  par la méthode du gradient à pas fixe . La fonction reçoit comme entrée la matrice A, le vecteur b, un point initial  $x_0$ , un pas  $\rho$  et la tolérance  $tol$ . En sortie, on a la solution finale  $sol$ , la donnée des  $x^{(i)}$  calculés à chaque itération  $xit$  et le nombre d'itérations  $nit$ . Fournir les fichiers Python qui vous à permis de faire les tests numériques. Expliquer votre code le plus rigoureusement possible.

```
def fixed_step_gradient(x,y,x0,rho,epsilon,itmax):
    X,q,A,b = X_build(x,y)
    x = x0
    i = 0
    xit = [x]
    r = A@x - b
    while(np.linalg.norm(r) > epsilon and i < itmax):
        d = -r
        alpha = rho
        x = x + alpha*d
        xit.append(x)
        r = A@x - b
        i += 1
```

La fonction a été construite un peu différemment de ce qui nous a été demandé, en effet nous avons appelé notre extraction des données "dataP.dat" et "dataQ.dat" dans l'expression de la fonction afin d'avoir une formule plus générale et cela quelque soit les données fournies.

Dans un premier temps nous construisons nos matrice  $X^T X$  et  $X^T q$  avec la fonction  $X\_build$  ensuite nous initialisons nos itération, notre  $x_0$  ainsi que notre première expression du critère d'arrêt  $r_k$ .

Ensuite dans notre boucle **while** nous fixons les conditions d'arrêt en fonction de l'erreur relative désirée et du nombre d'itérations maximal souhaité.

Ensuite nous appliquons l'algorithme du pas fixe avec l'expressions de notre direction de descente  $d = -r_k$  et de notre pas  $\alpha$ .

56. On fixe  $x_0 := c_0 = (-9, -7)^T$  et un pas  $\rho = 10^{-3}$ . Résoudre le système linéaire  $(X^T X)c = X^T q$  par la méthode du gradient à pas fixe considérée. Combien d'itérations sont nécessaires avec une tolérance fixée à  $10^{-6}$

Pour  $\rho = 10^{-3}$  :

```
c = np.array([[ -9, -7]])
x0 = np.transpose(c)

startFSG = time.time()
xtry,i,xit = fixed_step_gradient(x,y,x0,(10**(-3)),(10**(-6)),(5*10**(4)))
endFSG = time.time()
paceFSG = endFSG - startFSG
print("\n\n\t-----With fixed step gradient-----")
xit_ = np.array(xit)
print("\tfixed step gradient speed (s) =", paceFSG)
print("\tfixed step gradient result =\n", xtry)
print("\tfixed step gradient iteration = ",i)
```

```
-----With fixed step gradient-----
fixed step gradient speed (s) = 0.06012415885925293
fixed step gradient result =
[[0.75016235]
 [0.0638812 ]]
fixed step gradient iteration = 3367
```

On remarque que nous obtenons le vecteur déjà obtenu lors de la résolution avec `np.linalg.solve` en question 17.

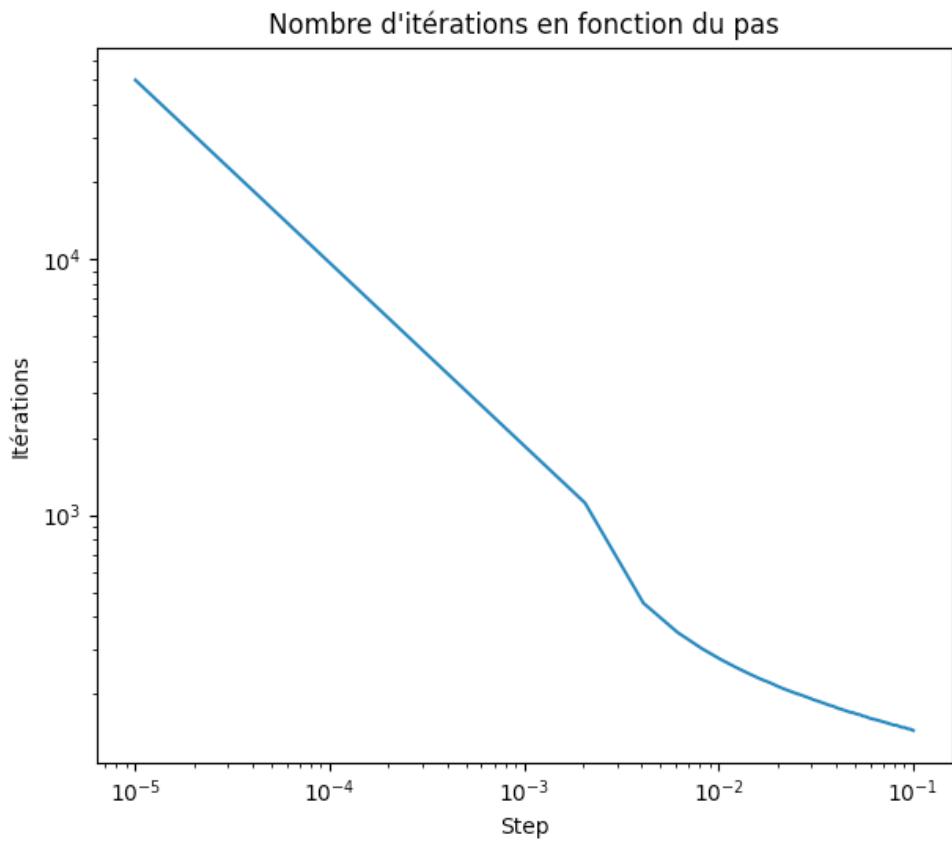
On obtient 3367 itérations. Cela nous montre tout de même déjà que le gradient à pas fixe est pas forcément la solution la plus optimale.

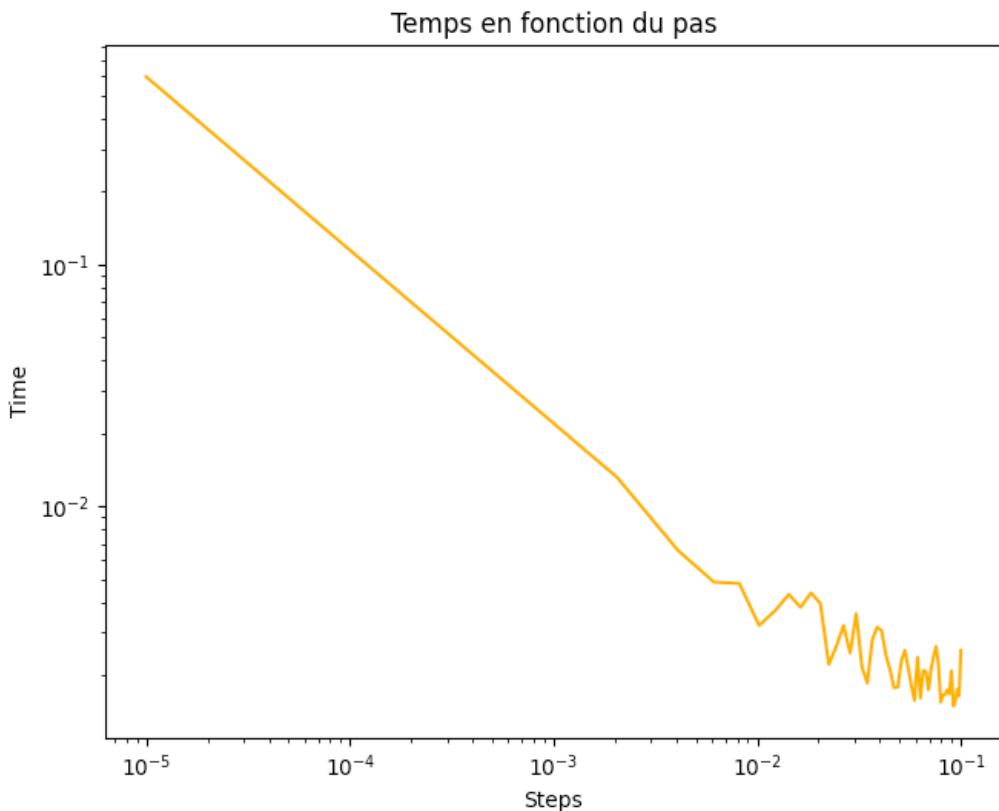
57. Qu'est-ce qu'on observe si on choisit  $\rho = 10^{-1}$  ? Et si on prend  $\rho = 10^{-5}$  ?

```
Steps = np.linspace(10**(-1), 10**(-5), 50)
STEPS_I = list()
TIME = list()
for i in range(len(Steps)) :
    timeC_start = time.time()
    xtryS,iS,xitS = fixed_step_gradient(x,y,x0,Steps[i],(10**(-6)),(5*10**(-4)))
    timeC_end = time.time()
    timeC_final = timeC_end - timeC_start
    STEPS_I.append(iS)
    TIME.append(timeC_final)

plt.figure("Nombre d'itérations en fonction du pas")
plt.title("Nombre d'itérations en fonction du pas")
plt.loglog(Steps, STEPS_I)
plt.ylabel("Itérations")
plt.xlabel("Step")

plt.figure("Temps en fonction du pas")
plt.title("Temps en fonction du pas")
plt.loglog(Steps, TIME, color='orange')
plt.ylabel("Time")
plt.xlabel("Steps")
plt.show()
```





Ces deux courbes précédentes nous montre :

Pour la première l'évolution des itérations en fonction du pas choisi<sup>3</sup>. On remarque alors grâce à cette échelle logarithmique que plus le pas est faible plus le temps est long et les itérations sont nombreuses. En effet pour certains pas nous atteignons même des durées de 30 à 40 secondes. Sachant que nos test ont été effectués sur un appareil très performant sur des machines un peu moins performantes cela peut prendre plusieurs minutes. On se rend bien compte que cette méthode n'est pas la plus optimale pour réaliser ce genre de résolution numérique.

### C. L'algorithme de descente du gradient à pas optimal

58. En utilisant la question 4 de la sous-section 3.1.2, justifier les différentes étapes de l'algorithme 3.3. Maintenant nous allons implémenter et tester l'algorithme 3.3 .

La méthode de descente est donnée par :  $x_{k+1} = x_k + \alpha_k d_k = x_k - \alpha_k r_k$ , où  $d_k$  est une direction de descente et on dit que c'est à pas optimal si le pas, noté  $\alpha_k$ , minimise la fonction réelle  $t \mapsto f(x_k + td_k)$ .

Lorsque le pas est optimal nous avons avant tout une relation d'orthogonalité et dans ce cas précis nous avons alors. :  $d_k = -\nabla F(x_k)$ . Ainsi si  $\alpha_k$  est un pas optimal alors  $d_k$  et  $\nabla F(x_{k+1})$  sont orthogonaux.

Analysons l'algorithme :

Étape 1 : Nous fixons le critère d'arrêt qui correspond au critère de résidu. On remarque ce que nous vous dit aux questions précédentes

Étape 2 : On fixe la direction de descente selon la relation d'orthogonalité exprimé ci-dessus

Étape 3 : On exprime le pas explicité comme en question 53

<sup>3</sup> Nous avons choisi des pas entre  $10^{-5}$  et  $10^{-1}$

Étape 4 : Comme pour la méthode du gradient à pas optimal nous explicitons l'itéré suivant.

59. Programmer une fonction Python  $sol,xit,nit = gradientPasOptimal(A,b,x0,tol)$  pour résoudre le problème  $Ax = b$  par la méthode du gradient à pas optimal (Algorithme ??). La fonction reçoit comme entrée la matrice A, le vecteur b, le point initial  $x_0$  et la tolerance  $tol$ . En sortie, on a la solution finale  $sol$ , la solution à chaque itération  $xit$  et le nombre d'itérations  $nit$ . Fournir les fichiers Python qui vous à permis de faire les tests numériques. Expliquer votre code le plus rigoureusement possible.

```
def optimal_step_gradient(x,y,x0,epsilon,itmax):
    X,q,A,b = X_build(x,y)
    x = x0
    i = 0
    xit = [x]
    r = A@x - b
    while(np.linalg.norm(r)>epsilon and i<itmax):
        d = -r
        rho = (np.transpose(r)@r)/(np.transpose(r)@A@d)
        x = x+ rho*d
        xit.append(x)
        r = A@x - b
        i += 1
    return(x,i,xit)
```

De la même façon que pour le pas fixe, notre fonction a été construite un peu différemment que ce qui a été demandé. En effet, nous nous servons encore une fois de notre fonction  $X\_build$  afin d'extraire les matrices nécessaire à la réalisation de l'algorithme.

Ensuite nous fixons  $x_0$  ainsi que notre première itération du critère de résidu. Cette première itération nous permet en fait de rentrer dans notre boucle `while`. Ce premier calcul constitue notre point de départ des itérations à venir. Nous rentrons dans notre boucle d'itération ensuite dans laquelle nous allons définir la direction de descente  $d_k$  ainsi que notre pas  $\alpha_k = \frac{\|r_k\|^2}{r_k^T(X^T X)r_k}$ .

Puis nous calculons les solutions  $x_k$  grâce à la relation :

$$x_{k+1} = x_k - \alpha_k \cdot r_k$$

Ensuite chaque  $x_k$  calculé est stocké et permet de calculer le  $x_{k+1}$  d'après.

60. On fixe  $x_0 := c_0 = (-9, -7)^T$ . Résoudre le système linéaire  $(X^T X)c = X^T q$  par la méthode du gradient à pas optimal, avec la tolérance  $tol := 10^{-6}$ .

```
startOSG = time.time()
xtry2,i2,xit2 = optimal_step_gradient(x,y,x0,(10**(-6)),(5*10**(4)))
endOSG = time.time()
paceOSG = endOSG - startOSG
print("\n\n\t-----With optimal step gradient-----")
xit2_ = np.array(xit2)
print("\toptimal step gradient speed (s) =", paceOSG)
print("\toptimal step gradient result =\n",xtry2)
print("\toptimal step gradient iterations =",i2)
```

```
-----With optimal step gradient-----
optimal step gradient speed (s) = 0.0008146762847900391
optimal step gradient result =
[[0.75016254]
[0.06388117]]
optimal step gradient iterations = 9
```

61. Qu'est-ce qu'on observe par rapport au nombre d'itérations ?

Dans un premier temps on remarque que la méthode du gradient à pas optimal atteint la solution avec environ 374 fois moins d'itération que le pas fixe. Le pas de cette méthode n'étant pas fixe (variable), le pas est recalculé à chaque nouvelle itération, cela permet d'avoir en réalité un pas qui diminue au fil des itérations et qui est également beaucoup plus faible que pour une méthode de pas fixe. Cette méthode permet, comparé à celle du pas fixe, d'arriver en moins d'itération à la solution.

De plus, comme on le remarque grâce au script python effectué ci-dessus, cette méthode est aussi beaucoup plus rapide que celle du pas fixe. Evidemment cela est fortement lié au nombre d'itération nécessaire à la résolution du système par cet algorithme.

### III. L'algorithme des gradients conjugués

#### A. Direction du gradient conjugué

62. Montrer que  $d_0$  défini par (28) est une direction de descente de  $f$  au point  $x_0$

On a :

$$d_k = \begin{cases} -\nabla f(x_0) & \text{si } k = 0 \\ -\nabla f(x_k) + \beta_{k-1} d_{k-1} & \text{si } k \geq 1 \end{cases}$$

On rappelle alors que  $d_k$  est une direction de descente si et seulement si il existe  $\delta > 0$  tel que :

$$f(x_k + \lambda d_k) < f(x_k) : \forall \lambda \in ]0, \delta[$$

On a alors : pour  $k = 0$

$$d_0 = -\nabla f(x_0)$$

Pour que  $d_0$  soit une direction de descente au point  $x_0$  il faut que la condition suivante soit respectée :

$$f'(x_0, d_0) = \nabla f(x_0)^T \cdot d_0 < 0$$

On admet que la fonction est différentiable au point  $x_0$  et on note l'équation suivante :

$$\begin{aligned} f(x_0 + \lambda d_0) &= f(x_0) + \lambda \nabla f(x_0)^T \cdot d \\ \Leftrightarrow f(x_0 + \lambda d_0) - f(x_0) &= \lambda \nabla f(x_0)^T \cdot d \\ \Leftrightarrow \nabla f(x_0)^T \cdot d &= \lim_{\lambda \rightarrow 0} \frac{f(x_0 + \lambda d_0) - f(x_0)}{\lambda} < 0 \end{aligned}$$

La limite étant négative strictement il existe un voisinage de 0 tel que :

$$\frac{f(x_0 + \lambda d_0) - f(x_0)}{\lambda} < 0$$

C'est à dire que :  $f(x_0 + \lambda d_0) < f(x_0)$

On a alors bien  $d_0$  qui est une direction de descente.

63. Soit  $k \in \mathbb{N}^*$ . Montrer que le pas de descente optimal  $\alpha_{k-1}$  de  $f$  au point courant  $x_{k-1}$

$$\text{suivant direction } d_{k-1} \text{ est donné par : } \rho_{k-1} = -\frac{r_{k-1}^T d_{k-1}}{d_{k-1}^T A d_{k-1}}$$

Nous avons montré à la question 40 que le pas optimal est défini par:

$$\alpha_k^* = -\frac{(Ax_k - b)^T \cdot d_k}{d_k^T A d_k}$$

Le résidu relatif à l'équation s'exprime :  $r_k = Ax_k - b$

En remplaçant dans l'équation du pas optimal on se retrouve donc que pour une direction de descente  $d_k$  on a :

$$\alpha_k^* = -\frac{r_k^T \cdot d_k}{d_k^T A d_k}$$

Et donc de la même façon pour une direction de descente  $d_{k-1}$  on a :

$$\alpha_{k-1}^* = -\frac{r_{k-1}^T \cdot d_{k-1}}{d_{k-1}^T A d_{k-1}}$$

64. Montrer que pour tout  $k \in \mathbb{N}^*$  on a:  $r_k = r_{k-1} + \alpha_{k-1} A d_{k-1}$

65. Montrer que pour tout  $k \in \mathbb{N}^*$  on a :  $r_k^T d_{k-1} = 0$

On a :

$$\begin{aligned} d_k &= -r_k \\ \Leftrightarrow d_k^T &= -r_k^T \\ \Leftrightarrow r_k^T &= -d_k^T \\ \Leftrightarrow r_k^T d_{k-1} &= -d_k^T d_{k-1} \end{aligned}$$

Les directions de descente étant perpendiculaire les unes aux autres on se retrouve alors avec :

$$r_k^T d_{k-1} = 0$$

## 66. Donner la définition d'une direction de descente de $f$ au point $x_k$

Une direction de descente de  $f$  au point  $x_k$  est défini par :  $f(x_k + \lambda d_k) < f(x_k)$

On note également que :  $x_{k+1} = x_k + \lambda d_k$

## 67. En utilisant la propriété (31), montrer que $d_k$ est bien une direction de descente de $q_h$ au point $x_k$ .

On sait comme nous l'avons dit précédemment que  $r_k^T d_{k-1} = 0$ , et comme :

$d_k = -r_k$  on a alors aussi  $d_{k-1} = -r_{k-1}$  ainsi on retrouve :

Et les résidus  $r_i$  sont orthogonaux, c'est à dire que  $r_i^T r_j = 0$  si  $i \neq j$ . Ainsi on peut conclure que :

$$r_k^T r_{k-1} = 0$$

Ainsi cette orthogonalité des résidus générés par les directions de descente nous laisse la conclusion que  $d_k$  est bien une direction de descente.

## 68. En utilisant les questions précédentes, montrer que la méthodes du gradient conjugué est bien une méthode à directions de descente.

Nous avons montré que le gradient a un pas  $\alpha_{k-1}$  qui suit une direction  $d_{k-1}$  et qui est strictement négatif. De plus nous avons aussi démontrer que les directions de descente sont perpendiculaires les unes des autres et que les résidus générés sont orthogonaux entre eux. Enfin nous avons montré la relation suivante :

$$r_k = r_{k-1} + \alpha_{k-1} d_{k-1}$$

Tout ceci fait donc du gradient conjugué une méthode à directions de descentes.

## B. Pseudo-code et test numérique

### 69. Montrer que le coefficient de conjugaison $\beta_{k-1}$ entre les directions $d_{k-1}$ et $d_k$ est défini

$$\text{par : } \beta_{k-1} = \frac{r_k^T A_h d_{k-1}}{d_k^T A_h d_{k-1}}$$

Les directions  $d_k$  sont des combinaisons linéaires entre la direction de la plus forte pente  $-\nabla f(x_k) = -r_k$  et la direction de  $d_{k-1}$  :

$$d_k = -r_k = \beta_k d_{k-1}$$

Où  $\beta_{k-1}$  est choisi tel que la conjugaison entre  $d_k$  et  $d_{k-1}$  soit vérifiée. On multiplie ensuite les deux coté, pour garder l'homogénéité du système, par  $d_{k-1}^T A$  :

$$d_k d_{k-1}^T A = -r_k d_{k-1}^T A + \beta_{k-1} d_{k-1} d_{k-1}^T A$$

Or d'après les règles d'orthogonalité nous avons :  $d_k A^T d_{k-1} = 0$  et donc  $d_k d_{k-1} A = d_k A^T d_{k-1}$  et ainsi nous avons :

$$\begin{aligned} -r_k d_{k-1}^T A + \beta_{k-1} d_{k-1} d_{k-1}^T A &= 0 \Leftrightarrow \beta_{k-1} d_{k-1} d_{k-1}^T A = r_k d_{k-1}^T A \\ &\Leftrightarrow \beta_{k-1} d_{k-1} A^T d_{k-1} = r_k A^T d_{k-1} \\ &\Leftrightarrow \beta_{k-1} = \frac{r_k A^T d_{k-1}}{d_{k-1}^T A d_{k-1}} = \frac{r_k^T A d_{k-1}}{d_{k-1}^T A d_{k-1}} \\ &\Leftrightarrow \beta_{k-1} = \frac{r_k^T (X^T X) d_{k-1}}{d_{k-1}^T (X^T X) d_{k-1}} \end{aligned}$$

On obtient le coefficient appelé coefficient de conjugaison entre les direction  $d_{k-1}$  et  $d_k$ .

70. En combinant la question 4 de la sous-section 3.1.2 et la question précédente, justifier les différentes étapes de algorithme.

Nous rappelons qu'à la question 40 nous avons défini ce qu'était le pas lors d'un gradient optimal. De plus, à la question précédente nous avons démontrer la formule du coefficient de conjugaison. Avec les connaissances que nous avons grâce à ce projet nous pouvons à présenter construire et comprendre l'algorithme du gradient conjugué.

Étape 1 : On commence par calculer la première itération du critère d'arrêt

Étape 2 : On a ici le calcul du pas de descente associé à la direction de descente  $d_{k-1}$ , on remarque que lorsque  $i = 0$  on a à faire au pas de descente du gradient optimal. En effet, la méthode du gradient conjugué est en fait une amélioration de la méthode du gradient à pas optimal.

Étape 3 : on calcul la nouvelle approximation  $x_k$  selon la formule vu précédemment :

$$x_k = x_{k-1} + \alpha_{k-1} d_{k-1}$$

Étape 4 : On calcule le coefficient  $\beta_{k-1}$  avec la formule ci-dessus/

Étape 5 : Ce coefficient de conjugaison nous permet de calculer une nouvelle direction de descente (les directions de descente sont variables)  $d_k$  tel que :  $d_k = r_k + \beta_{k-1} d_{k-1}$ . De cette formule on peut exposé aisément la propriété que l'on nous a exposé au début de cette partie c'est à dire que les directions de descente  $d_k$  et  $d_{k-1}$  soient A-conjuguées c'est à dire :  $d_k^T A d_{k-1} = 0$ .

71. Programmer une fonction Python `sol,xit,nit = gradientConjugue(A,b,x0,tol)` pour résoudre le problème  $Ax = b$  par la méthode du gradient conjugué (Algorithme). La fonction reçoit comme entrée la matrice A, le vecteur b, le point initial  $x_0$  et la tolérance tol. En sortie, on a la solution finale sol, la solution à chaque itération xit et le nombre d'itérations nit. Fournir les fichiers Python qui vous a permis de faire les tests numériques. Expliquer votre code le plus rigoureusement possible

```
def conjugued_gradient(x,y,x0, epsilon, itmax):
    X,q,A,b = X_build(x,y)
    x = x0
    i = 1
    xit = [x]
    r = A@x - b
    while(np.linalg.norm(r)>epsilon and i<itmax):
        if (i ==1):
            d = -r
            d_1 = d
        else:
            beta = ((np.transpose(r)@r))/((np.transpose(r_1)@r_1))
            d = -r + beta*d_1
            d_1 = d
        rho = (np.transpose(r)@r)/(np.transpose(d)@A@d)
        x = x+ rho*d
        xit.append(x)
        r_1 = r
        r = A@x - b
        i+=1
    return(x,i,xit)
```

De la même manière que les deux algorithme de gradient précédent nous avons construit notre fonction un peu différemment e, construisant directement A et b dans la fonction grâce à `X_build`, nous avons aussi initialiser notre premier  $x_k$  pour  $k=0$ . Comme pour tous les autres méthodes de gradient nous avons calculé la première itération du critère d'arrêt afin de rentrer dans la boucle `while`. Et à cet endroit nous avons une disjonction de cas c'est à dire que :

Pour  $k = 1$  :

Nous avons effectuer l'algorithme de la même façon que pour le gradient à pas optimal. Nous avons juste calculé défini ensuite  $d_k$  comme  $d_{k-1}$  pour calculer l'itération suivante ( on rappelle que les directions de descente sont variables)

Pour  $k > 1$  :

Nous avons alors appliqué la méthode détaillée précédemment du gradient conjugué. En calculant le coefficient de conjugaison ainsi que les nouvelles directions  $d_k$ .

Ensuite quelque soit le cas nous avons défini grâce aux formules précédente le pas du gradient conjugué et calculer les  $x_{k+1}$  grâce à la formule :  $x_{k+1} = x_k + \alpha_k d_k$

Notons  $r_{k-1} = r_k$  pour passer à l'itération de  $\beta_{k+1}$  grâce au  $r_{k-1}$  précédemment obtenu.

On fini par redéfinir à chaque boucle le nouveau critère d'arrêt  $r_k$  afin de vérifier l'entrée dans le `while`.

72. On fixe  $x_0 := c_0 = (-9, -7)^T$ . Résoudre le système linéaire  $(X^T X)c = X^T q$  par la méthode du gradient conjugué. (On fixera la tolérance  $\text{tol} := 10^{-6}$ ).

```
startCG = time.time()
xtry3,i3,xit3 = conjugued_gradient(x,y,x0,(10**(-6)),(5*10**(4)))
endCG = time.time()
paceCG = endCG - startCG
print("\n\n\t-----With conjugued gradient-----")
xit3_ = np.array(xit3)
print("\tconjugued gradient speed (s) =", paceCG)
print("\t conjugued gradient result =\n", xtry3)
print("\tconjugued gradient iteration =",i3)
```

```
-----With conjugued gradient-----
conjugued gradient speed (s) = 0.0002608299255371094
conjugued gradient result =
[[0.75016254]
 [0.06388117]]
conjugued gradient iteration = 3
```

Comme pour les deux codes de gradient précédents nous avons aussi calculer la vitesse d'exécution du gradient conjugué afin de comparer au mieux les méthodes

### 73. Qu'est-ce qu'on observe par rapport au nombre d'itérations ?

En comparant cette méthode à celle du gradient optimal on se rend compte que l'algorithme réalise trois fois moins d'itérations. En effet nous en avons que 3 ce qui montre l'optimisation parfaite du code.

Si on compare ces itérations par rapport à celle du gradient à pas fixe on réalise encore plus à quel point il n'est pas judicieux de réaliser les résolutions numériques par cette méthode.

La méthode la plus adéquate est donc la méthode du gradient conjugué.

Cependant, et bien qu'il soit faible, on se rend compte que la vitesse d'exécution du gradient conjugué n'est pas significativement plus rapide que celle du gradient à pas conjugué. Cela s'explique sûrement par le nombre de calcul nécessaire à la réalisation d'une seule itération.

Précisons tout de même que cet algorithme reste le plus rapide et le plus optimisé.

## IV. Analyse des résultats numériques

### A. Les résultats

74. On considère un pas de discréttisation  $\delta = 0.5$ . Définir le pavé  $[-10, 10] \times [-10, 10]$  avec un pas  $\delta$  dans chaque direction. Fonction : `numpy.meshgrid`.

```
x_axis = np.arange(-10,10.5,0.5)
y_axis = np.arange(-10,10.5,0.5)
xx_axis, yy_axis = np.meshgrid(x_axis,y_axis)
```

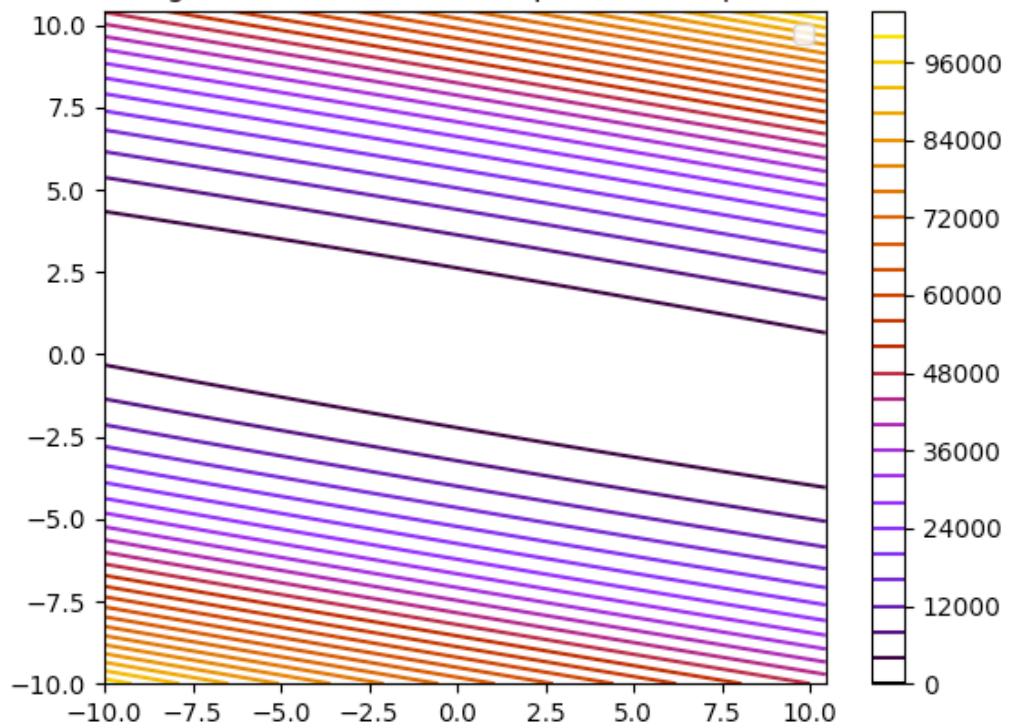
75. Afficher les courbes de niveau de  $F(c_1, c_2)$ , et son gradient dans le pavé  $[-10,10] \times [-10,10]$ .

Avec Matplotlib utiliser : `contour`, `quiver`.

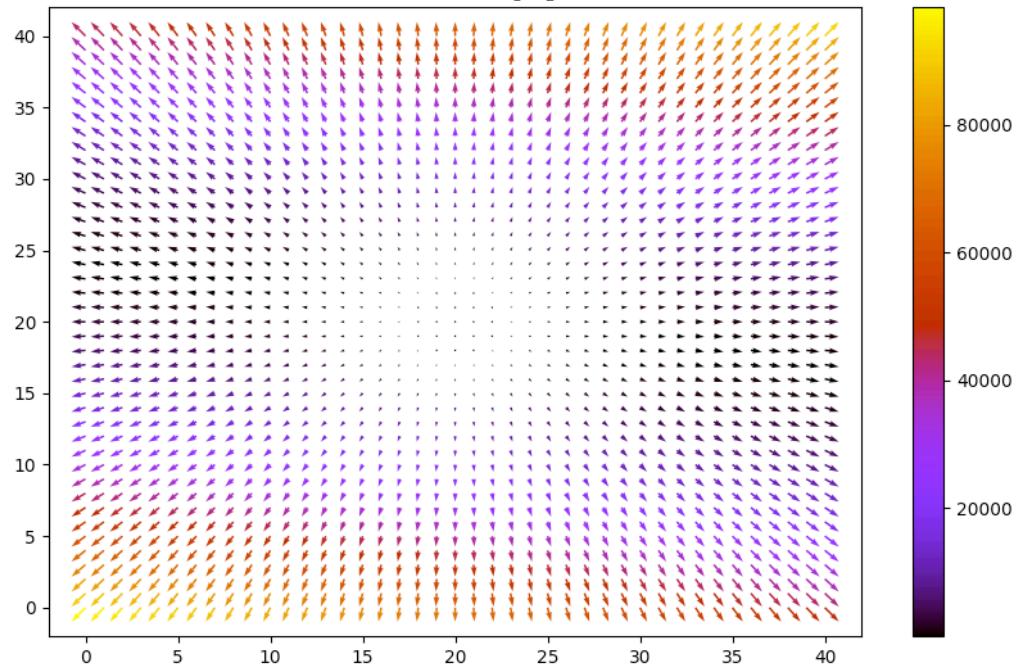
```
LM = plt.contour(xx_axis,yy_axis,f,30, cmap = "gnuplot")
plt.title("ligne de niveaux de F (Repère canonique)")
plt.colorbar(LM)
fig = plt.figure("gradient")
quiv = plt.quiver(xx_axis, yy_axis, f, cmap = "gnuplot")
plt.colorbar(quiv)
plt.title("Gradient de F($c_1, c_2$)")
```

Nous avons donc d'abord tracé les courbes de niveaux de la fonction  $F$  grâce à `contour` de Matplotlib et ensuite nous avons tracé le gradient dans le pavé  $[-10,10] \times [-10,10]$  grâce à la fonction `quiver`. Nous avons utilisé- un pas de discréttisation de 0.5.

ligne de niveaux de  $F$  (Repère canonique)



Gradient de  $F(c_1, c_2)$

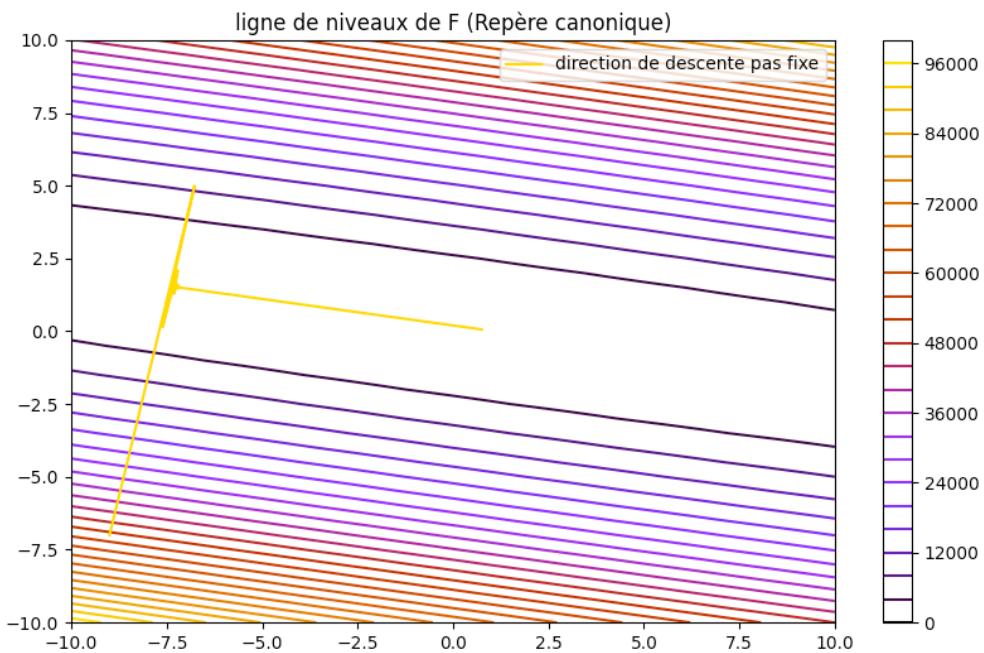


76. Sur les figures obtenues à la question précédente, à partir du vecteur initial  $c_0$  tracer pour chaque méthode qui converge les trajectoires reliant la solution à chaque itération (variable  $xit$  en sortie des algorithmes). Avec Matplotlib utiliser : `plot`.

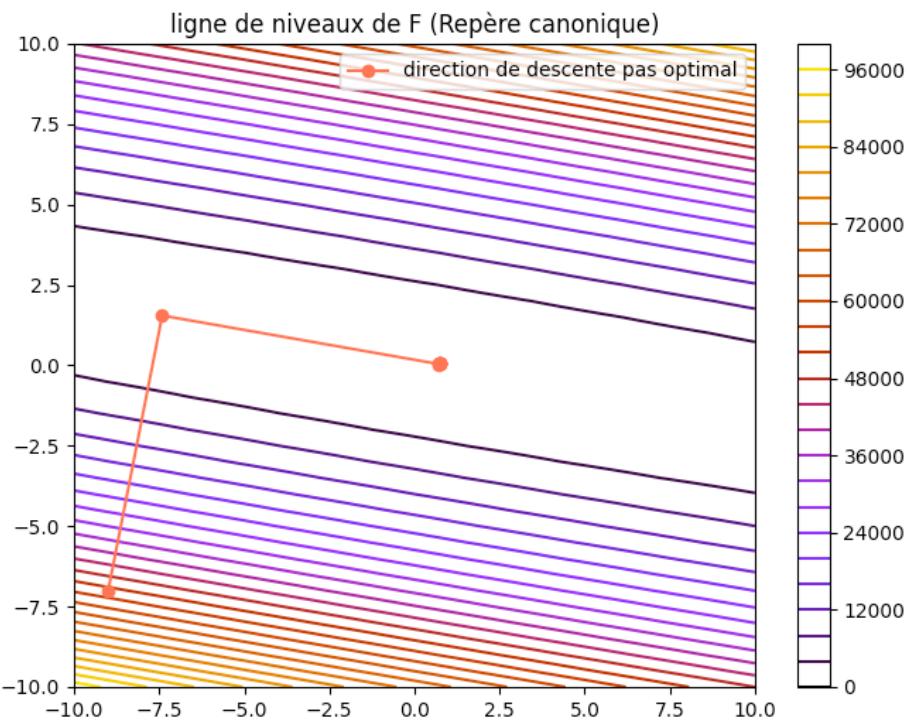
```
F = list()
f = (1/2)*(Z[0,0]*xx_axis**2 + 2*Z[0,1]*xx_axis*yy_axis+Z[1,1]*yy_axis**2 - 2*(w[0]*xx_axis+w[1]*yy_axis)+s)
plt.figure("Level map")

plt.plot(xit_[:,0,0], xit_[:,1,0], color='gold', label='direction de descente pas fixe')
"""
plt.plot(xit2_[:,0,0], xit2_[:,1,0], color='tomato', marker = "o", label='direction de descente pas optimal')
plt.plot(xit3_[:,0,0], xit3_[:,1,0], color='maroon', marker = "v", label='direction de descente gradient conjugué')
"""

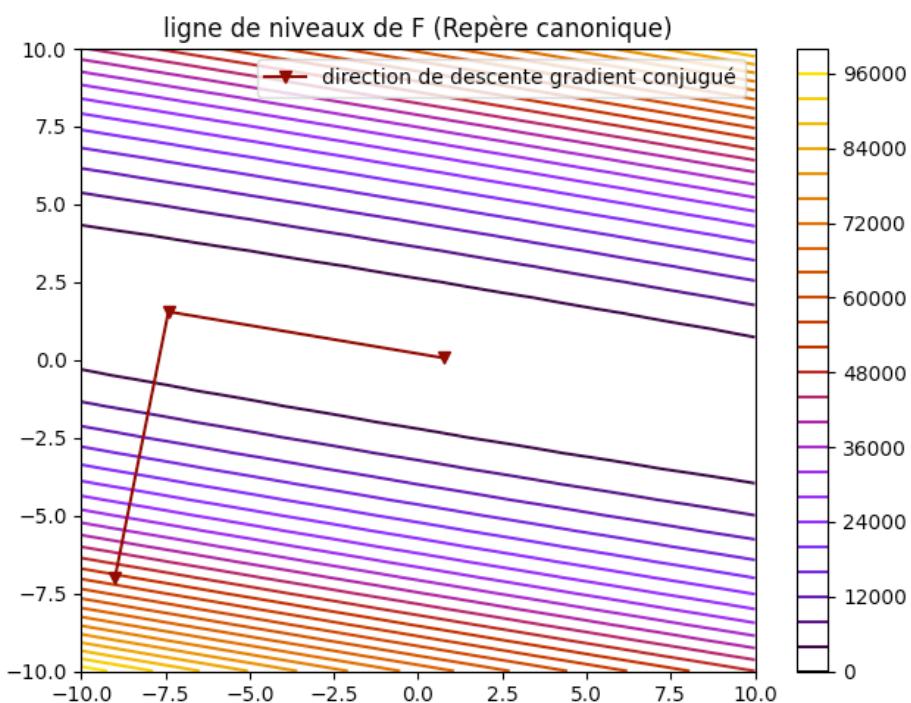
LM = plt.contour(x_axis,y_axis,f,30, cmap = "gnuplot")
plt.title("ligne de niveaux de F (Repère canonique)")
plt.colorbar(LM)
plt.legend()
```



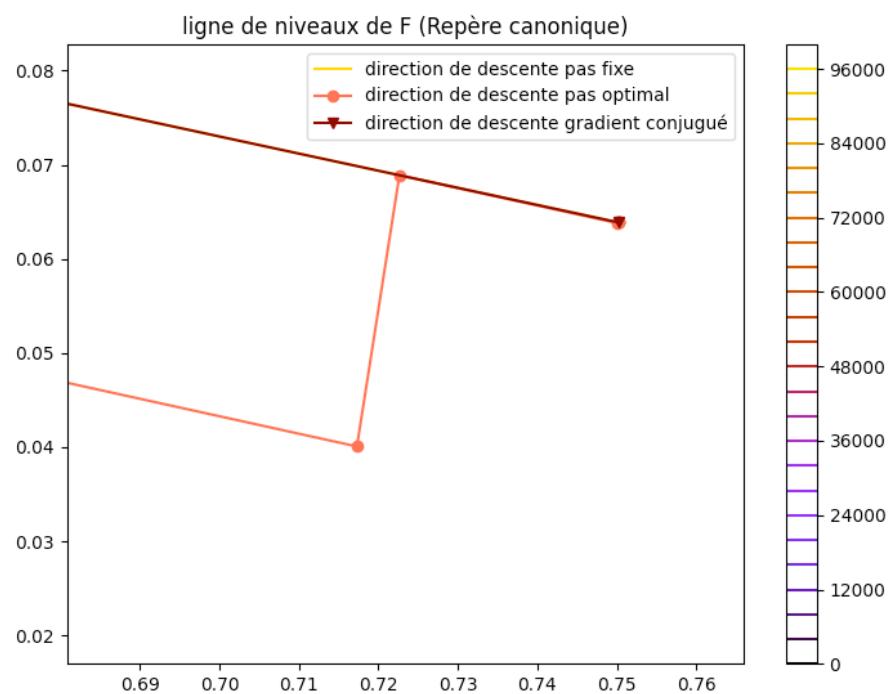
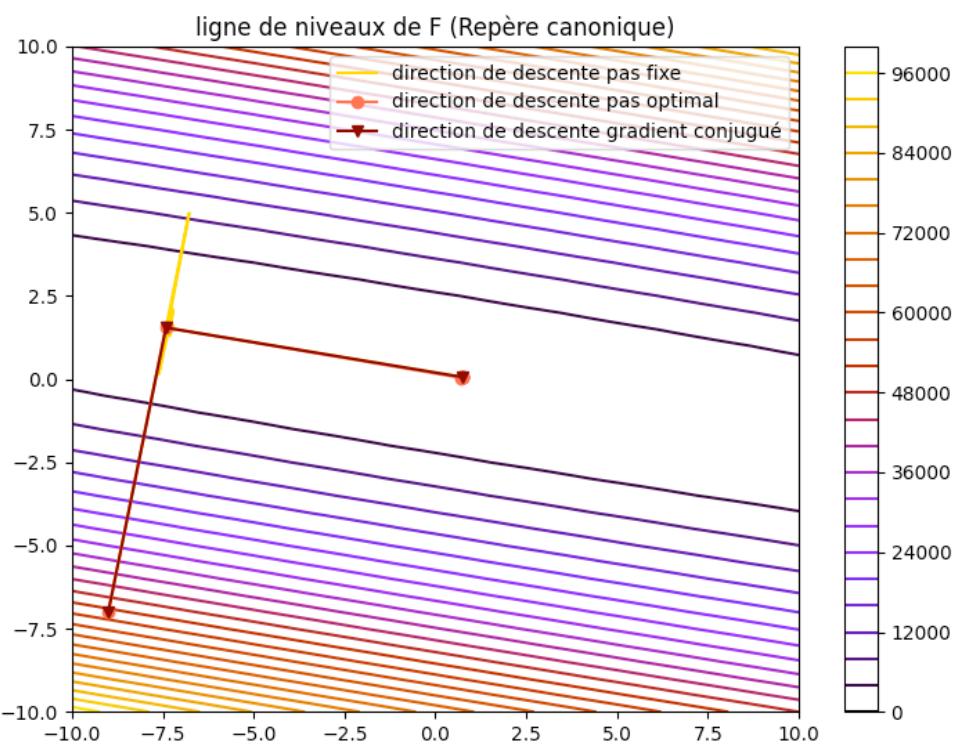
*Méthode du gradient à pas fixe*



*Méthode du gradient à pas optimal*



*Méthode du gradient conjugué*



77. Tracer dans un graphique les couples ( $p_i, q_i$ )  $i = 1, \dots, m$  (voir point (b)). Sur la même figure, afficher les trois modèles linéaires  $q = c_1 + c_2 p$  construits à partir des solutions avec les trois méthodes de gradient.

```
def display_data(x,y, xtry1, xtry2, xtry3):
    plt.figure("2.1.2 Ajustement linéaire")

    c1 = xtry1[0,:]
    c2 = xtry1[1,:]
    c1_2 = xtry2[0,:]
    c2_2 = xtry2[1,:]
    c1_3 = xtry3[0,:]
    c2_3 = xtry3[1,:]
    Q1 = list()
    Q2 = list()
    Q3 = list()
    for i in range (len(x)):
        q1 = c1 + c2*x[i]
        q2 = c1_2 + c2_2*x[i]
        q3 = c1_3 + c2_3*x[i]
        Q1.append(q1)
        Q2.append(q2)
        Q3.append(q3)

    plt.subplot(2,2,1)
    plt.plot(x,Q1, color = 'green', label='approximation linéaire gradient à pas fixe')
    plt.plot(x,y, color = 'black', marker ='x', linestyle='none')
    plt.xlabel("Age des enfants")
    plt.ylabel("Taille des enfants")
    plt.title("Taille en fonction de l'age")
    plt.legend()

    plt.subplot(2,2,2)
    plt.plot(x,Q2, color = 'orange', label='approximation linéaire gradient à pas optimal')
    plt.plot(x,y, color = 'black', marker ='x', linestyle='none')
    plt.xlabel("Age des enfants")
    plt.ylabel("Taille des enfants")
    plt.title("Taille en fonction de l'age")
    plt.legend()

    plt.subplot(2,2,3)
    plt.plot(x,Q3, color = 'red', label='approximation linéaire gradient conjugué')
    plt.plot(x,y, color = 'black', marker ='x', linestyle='none')
    plt.xlabel("Age des enfants")
    plt.ylabel("Taille des enfants")
    plt.title("Taille en fonction de l'age")
    plt.legend()

    plt.subplot(2,2,4)
    plt.plot(x,Q1, color = 'green', label='approximation linéaire gradient à pas fixe')
    plt.plot(x,Q2, color = 'orange', label='approximation linéaire gradient à pas optimal')
    plt.plot(x,Q3, color = 'red', label='approximation linéaire gradient conjugué')
    plt.plot(x,y, color = 'black', marker ='x', linestyle='none')
    plt.xlabel("Age des enfants")
    plt.ylabel("Taille des enfants")
    plt.title("Taille en fonction de l'age")
    plt.legend()
    plt.show()
```

```

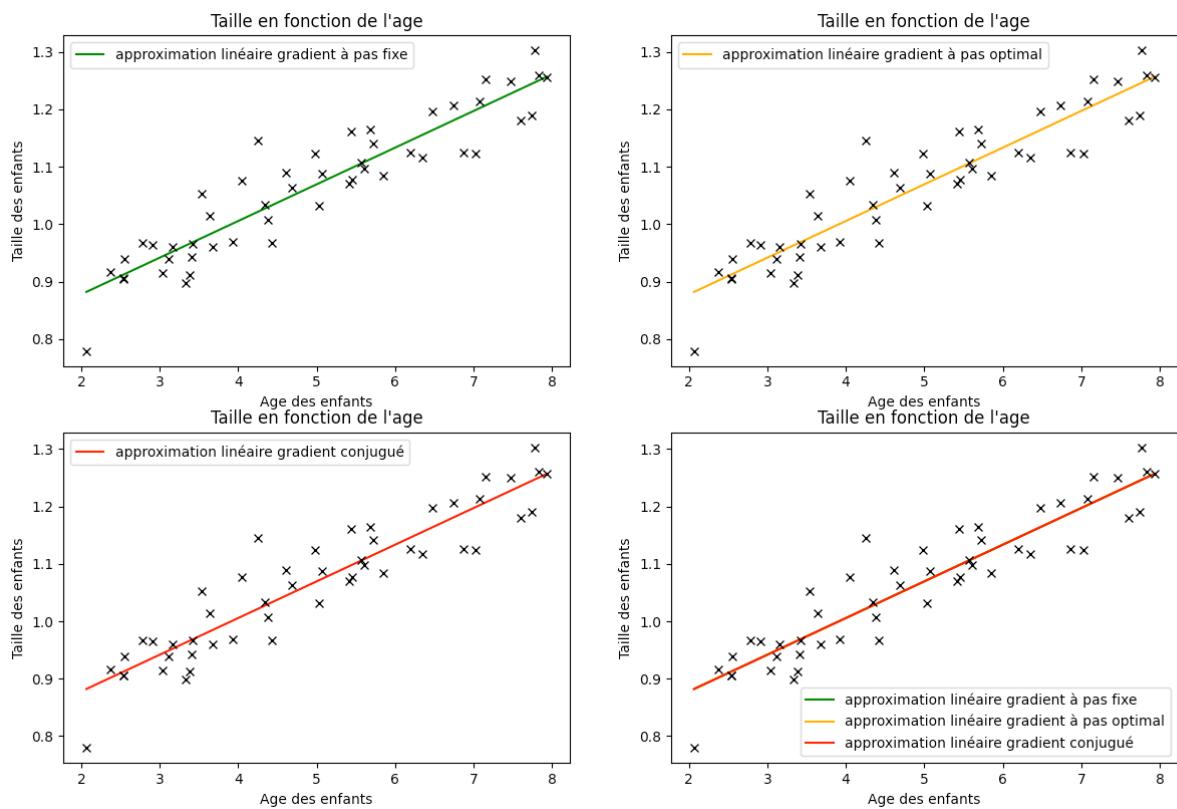
startFSG = time.time()
xtry,i,xit = fixed_step_gradient(x,y,x0,(10**(-3)),(10**(-6)),(5*10**(-4)))
endFSG = time.time()
paceFSG = endFSG - startFSG
print("\n\n\t-----With fixed step gradient-----")
xit_ = np.array(xit)
print("\tfixed step gradient speed (s) =", paceFSG)
print("\tfixed step gradient result =\n",xtry)
print("\tfixed step gradient iteration =",i)

startOSG = time.time()
xtry2,i2,xit2 = optimal_step_gradient(x,y,x0,(10**(-6)),(5*10**(-4)))
endOSG = time.time()
paceOSG = endOSG - startOSG
print("\n\n\t-----With optimal step gradient-----")
xit2_ = np.array(xit2)
print("\toptimal step gradient speed (s) =", paceOSG)
print("\toptimal step gradient result =\n",xtry2)
print("\toptimal step gradient iterations =",i2)

startCG = time.time()
xtry3,i3,xit3 = conjugued_gradient(x,y,x0,(10**(-6)),(5*10**(-4)))
endCG = time.time()
paceCG = endCG - startCG
print("\n\n\t-----With conjugued gradient-----")
xit3_ = np.array(xit3)
print("\tconjugued gradient speed (s) =", paceCG)
print("\t conjugued gradient result =\n",xtry3)
print("\tconjugued gradient iteration =",i3)

level_map(xit_, xit2_, xit3_)
display_data(x,y, xtry, xtry2,xtry3)

```



78. Récapituler, pour chaque méthode, si l'algorithme converge et le nombre d'itérations.

```
-----With fixed step gradient-----
fixed step gradient speed (s) = 0.033082008361816406
fixed step gradient result =
[[0.75016235]
[0.0638812 ]]
fixed step gradient iteration = 3367

-----With optimal step gradient-----
optimal step gradient speed (s) = 0.00039577484130859375
optimal step gradient result =
[[0.75016254]
[0.06388117]]
optimal step gradient iterations = 9

-----With conjugued gradient-----
conjugued gradient speed (s) = 0.0002307891845703125
conjugued gradient result =
[[0.75016254]
[0.06388117]]
conjugued gradient iteration = 3
```

Récapitulons :

### Avec la méthode du gradient à pas fixe:

Pour une tolérance de  $10^{-6}$  et un pas de  $10^{-3}$  :

La méthode converge

Nous avons 3367 itérations et une vitesse de 0.033 secondes.

Nous précisons concernant la vitesse d'exécution qu'elle dépend aussi du nombre d'application utilisé en même temps sur l'ordinateur. Si c'est pas une donnée fixe comme les itérations.

On remarque plus le pas est petit plus le nombre d'itérations est grand.

### Avec la méthode du gradient à pas optimal :

Pour une tolérance de  $10^{-6}$  :

La méthode converge

Nous avons 9 itérations (374 fois moins que le gradient à pas optimal) et une vitesse de 0.0004 secondes.

### Avec la méthode du gradient conjugué :

Pour une tolérance de  $10^{-6}$  :

La méthode converge

Nous avons 3 itérations (3 fois moins que le gradient à pas optimal et 1122 fois moins que le gradient à pas fixe) et une vitesse de 0.00023.

Remarquons qu'avec moins de logiciel ouvert les vitesse du pas optimal et du gradient conjugué sont presque équivalentes.

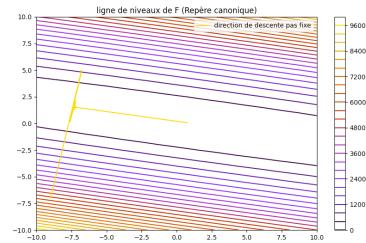
## B. Comportements des méthodes

79. Commenter le comportement des trois méthodes. Bien évidemment, si vous le désirez vous pouvez ajouter des critères de comparaison. Vous y êtes même encouragé !

Observons les figures en page 45 ainsi que les résultats de la page précédente.

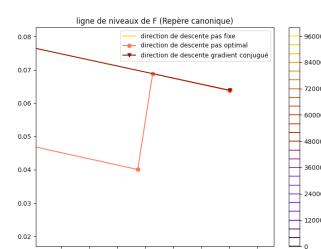
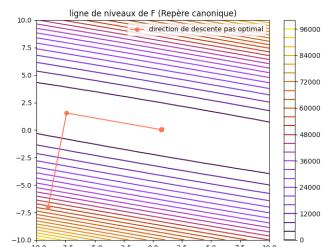
### Gradient à pas fixe :

Nous voyons tout d'abord par son graphique qu'il lui faut un nombre d'itération assez important pour arriver à la solution de l'équation  $Ax = b$ . De plus sa trajectoire de descente n'est pas directe comme on le remarque ci-contre qu'avant d'arriver à la solution finale la méthode passe par énormément de points dont certain implique une direction de "montée" cela paraît donc assez curieux étant donné la nature de direction de descente de la méthode. On se rend bien compte ici que cette méthode n'est pas très précise et surtout très gourmande en ressource numérique. Il paraît impensable de l'utiliser pour des pas très faible, comme nous vous déjà pu conclure précédemment. Néanmoins, tout cela n'empêche en aucun cas la méthode d'être précise puisque nous retrouvons la solution de l'équation  $Ax = b$  avec une très bonne précision de  $10^{-6}$



### Gradient à pas optimal :

Nous voyons grâce aux deux graphiques ci-contre quelle gradient à pas optimal est une assez bonne méthode de résolution numérique. En effet, elle possède peu d'itération et une vitesse correcte. Néanmoins, elle ne reste pas la plus directe et cela pourrait poser quelques problèmes lors de la résolution de système plus important. En effet, le fait qu'elle ne soit pas précise, à l'instar du gradient à pas fixe, pourrait générer sur des systèmes plus conséquent des erreurs relatives plus importantes. Cependant, nous retiendrons que dans notre cas cette méthode est déjà largement suffisante et très précise puisqu'on obtient le résultat correct à  $10^{-6}$  près.



### Gradient conjugué :

Grâce aux graphiques ci-contre et aux résultats précédents on se rend compte que cet algorithme est le plus direct. En effet, nous arrivons à la solution en très peu d'itérations et avec la même précision que pour les deux méthodes précédentes de  $10^{-6}$ . Cet algorithme est sûrement le plus fiable pour les systèmes plus important à résoudre. Notons tout de même que la complexité de cette algorithme implique une vitesse presque égale à celle du gradient à pas optimale et pourrait poser problème à des petits ordinateurs sur des systèmes plus complexe à résoudre.

