FSBM

FACULTE DES SCIENCES BEN M'SICK
UNIVERSITE HASSAN II DE CASABLANCA

Algorithmes d’optimisation

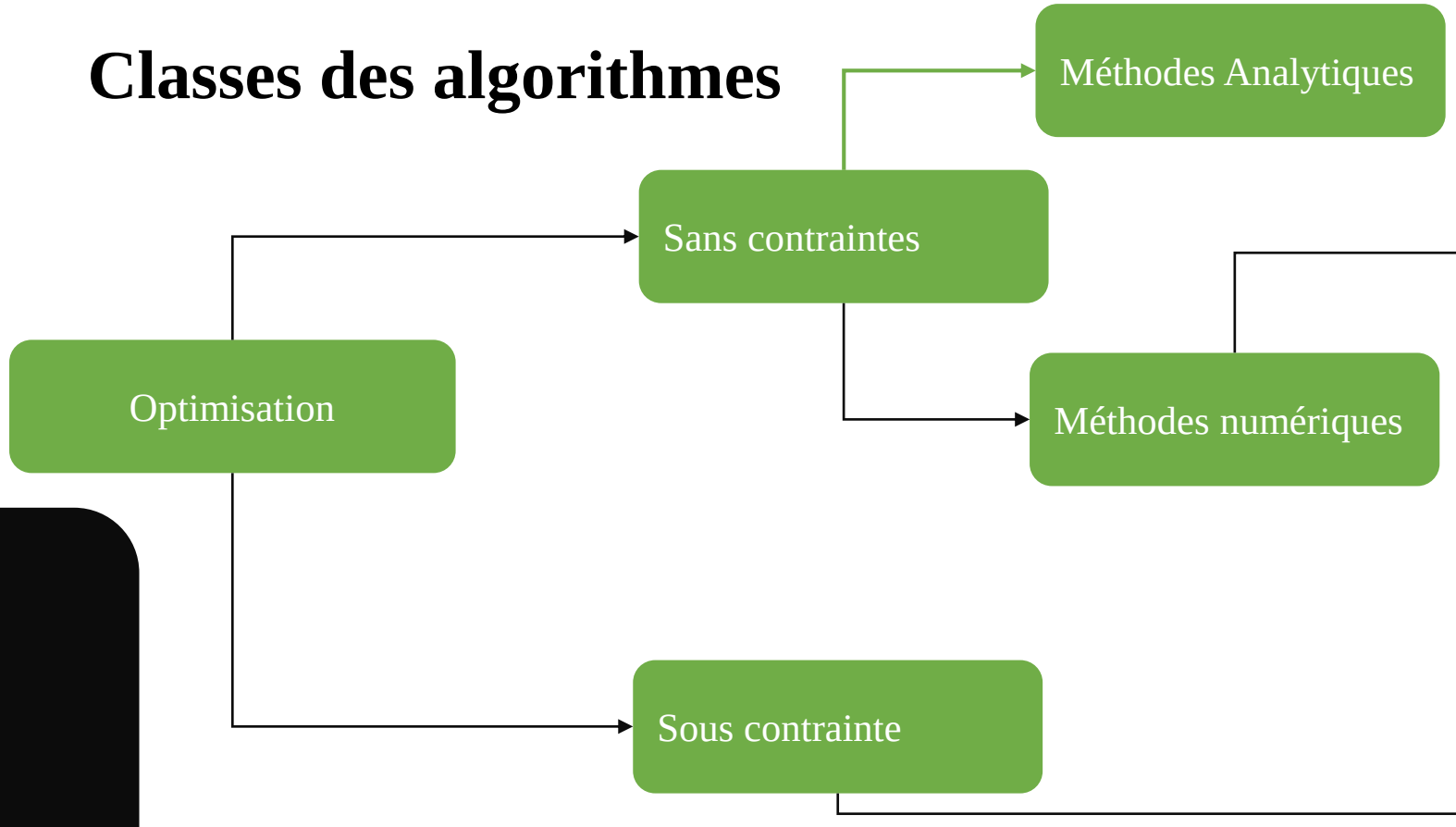
Pr. Faouzia Benabbou (faouzia.benabbou@univh2c.ma)
Département de mathématiques et Informatique
Master Data Science & Big Data
2024-2025

Plan du Module: Algorithmes d’optimisation

1. Introduction
2. Rappels mathématiques
3. Algorithmes d’Optimisation Sans Contraintes
4. Algorithmes d’Optimisation avec Contraintes
5. Programmation Linéaire et non Linéaire
6. Optimisation combinatoire et globale
7. Méthodes heuristiques et métaheuristiques

Les algorithmes d’optimisation

Classes des algorithmes



- Méthode sans gradient
- Méthode de descente de gradient:
  - ✓ Méthode de gradient à pas fixe et optimal
  - ✓ Méthode de gradient conjugué
  - ✓ Méthodes du gradient quasi-newton
  - ✓ Méthode de Newton

- Méthodes générales de résolutions de PCE, PCI
- Programmation linéaire et non linéaire
- Optimisation combinatoire et globale
- Méthodes heuristiques et métaheuristiques
- Algorithmes multi-objectifs

Méthodes d’optimisation de descente gradient

Méthode de gradient à pas fixe: limites

- L’inconvénient d’utiliser un pas de descente constant est que l’algorithme converge très lentement si le pas est trop petit.
- Dans le cas où le pas est trop grand, l’algorithme peut devenir instable et diverger.
- Le choix optimal de la valeur de α est très important pour la convergence.
- Il faudra par conséquent ajuster la valeur de α à chaque itération.

Méthodes d'optimisation de descente gradient

Méthode de gradient à pas optimal

- la méthode de gradient à pas optimal a pour idée de base de chercher α_k diminuant davantage $f(x_k)$ dans la direction d_k .

$$d_k = -\nabla f(x_k).$$

Dans la méthode de gradient à pas fixe on a: $x_{k+1}=x_k+ \alpha\, d_k$ et on cherche à minimiser $f(x_{k+1})$ par rapport à α.

on va poser: $\varphi(\alpha) = f(x_{k+1}) = f\big(x_k - \alpha\nabla f(x_k)\big)$ et on cherche le pasα **optimal** tel qu'il réalise $\min_{\alpha>0} \varphi(\alpha)$.

- Minimiser $\varphi(\alpha)$ revient à chercher α tel que $\frac{d\varphi(\alpha)}{d\alpha}=0$.

Méthodes d’optimisation de descente gradient

Méthode de gradient à pas optimal

Algorithme 4 : descente de gradient à pas optimal

1. **Initialisation :** $x_0 \in \mathbb{R}^n$, f une fonction objective de classe 1, $\alpha \in \mathbb{R}^{+*}$.

   ε : critère d’arrêt (tolérance sur le gradient), max_iter: nombre max d'itérations

2. **Répéter**
   a) Calculer la direction de la descente $d_k = -\nabla f(x_k)$.
   b) Recherche linéaire avec condition d’Armijo de $\alpha_k = \min_{\alpha>0} f(x_k - \alpha \nabla f(x_k))$
   c) Mettre à jour la solution : $x_{k+1} = x_k + \alpha\, d_k$
   d) k++

Méthodes d'optimisation de descente gradient

Méthode de gradient à pas optimal

- Pour calculer le minimum des pas, on va utiliser l'algorithme de la **recherche linéaire avec la condition de Armijo**
- L'objectif est de trouver un pas α satisfaisant la condition d'Armijo suivante :

$$f(x + \alpha\, d) \leq f(x) + \delta * \alpha * \nabla f(x)^T d.$$

À chaque itération on reduit α par $\alpha * \beta$

Avec :

- δ : paramètre d'Armijo pris dans l'intervalle [0,1] (doit être petite, ex. 10−4)
- β : facteur de réduction du pas pris dans l'intervalle [0,1] (ex. 0.7)
- α : le pas de la descente initial et d est la direction de descente

Dans la descente du gradient on prend la plus forte descente $d = -\nabla f(x)$, après simplification on obtient la condition de Armijo suivante :

$$f(x + \alpha\, d) \leq f(x) + \delta * \alpha * \|\nabla f(x)\|^2$$

Méthodes d’optimisation de descente gradient

Méthode de gradient à pas optimal

- Si la condition n'est pas satisfaite, le pas δ est réduit de manière multiplicative en utilisant un facteur β.

Algorithme de la recherche linéaire avec la condition de Armijo

1. Initialisation : $\alpha \leftarrow$valeur<1 (pas initial choisi)

2. Tant que la condition de Armijo n'est pas satisfaite et que i<max_iter

a) Si $f(x - \alpha \nabla f(x)) \leq f(x) - \delta \alpha \|\nabla f(x)\|^2$

Alors, retourner α (pas valide)

b) Sinon, $\alpha \leftarrow \alpha * \beta$

3. Retourner le dernier α trouvé

Méthodes d’optimisation de descente gradient

Méthode de gradient à pas optimal: limites

- Trouver α_k nécessite de résoudre un **sous-problème d'optimisation** unidimensionnel à chaque itération, ce qui peut être coûteux en temps de calcul.
- Lorsque le gradient varie fortement en norme, le pas optimal peut fluctuer, cela peut entraîner des zigzags et mener à une convergence lente.
- C’est une méthode **simple et efficace** pour les problèmes bien conditionnés et convexes.
- Cependant, elle devient inefficace dans les cas **mal conditionnés (si** le rapport entre la plus grande et la plus petite valeur propre est élevé), non **convexes**, ou lorsque la recherche du pas est trop **coûteuse**.

Méthodes d'optimisation de descente gradient

Méthode de gradient à pas fixe

- **Exemple. Soit** $f(x , y)=100x^2+y^2$.

```python
x, y = sp.symbols('x y', real=True)

# Définition de la fonction f(x, y)
f = 100*x**2 + y**2

# Calcul dgradient symbolique
f_x = sp.diff(f, x)  # Dérivée de f par rapport à x
f_y = sp.diff(f, y)  # Dérivée de f par rapport à y

# Affichage des dérivées partielles
print("Dérivée partielle par rapport à x:", f_x)
print("Dérivée partielle par rapport à y:", f_y)

# Conversion des dérivées partielles en fonctions numériques
grad_f_x = sp.lambdify((x, y), f_x, 'numpy')
grad_f_y = sp.lambdify((x, y), f_y, 'numpy')

# Fonction pour calculer le gradient
def grad_f(x_val, y_val):
    return np.array([grad_f_x(x_val, y_val), grad_f_y(x_val, y_val)])
```

Méthodes d'optimisation de descente gradient

Méthode de gradient à pas fixe

- **Exemple. Soit** $f(x,y)=100x^2+y^2$.

```python
#  🔹 Descente de gradient à pas optimal
def descente_optimal(x0, y0, epsilon=1e-6, max_iterations=50):
    x, y = x0, y0
    trajectory = [(x, y)]
    num_iterations = 0

    for _ in range(max_iterations):
        grad = grad_f_numeric(x, y)
        norm_grad = np.linalg.norm(grad)

        if norm_grad < epsilon:
            break  # Arrêt si la norme du gradient est trop faible

        # Recherche linéaire pour trouver le pas optimal
        t = line_search_armijo(np.array([x, y]), grad)

        # Mise à jour des variables
        x -= t * grad[0]
        y -= t * grad[1]

        trajectory.append((x, y))
        num_iterations += 1

    return trajectory, num_iterations  # Retourne la trajectoire et le nombre d'itérations
```

Méthodes d’optimisation de descente gradient

Méthode de gradient à pas fixe

- **Exemple. Soit** $f(x,y)=100x^2+y^2$

```python
# Paramètres de la descente de gradient
alpha = 0.01  # Taux d'apprentissage
epsilon = 1e-6  # Critère d'arrêt basé sur la norme du gradient
max_iterations = 100  # Nombre maximum d'itérations
x0, y0 = -1.0, 1.0  # Point de départ

# Exécution de la descente de gradient
x_vals, y_vals, num_iterations = gradient_descent(f, grad_f, x0, y0, alpha, epsilon, max_iterations)

# Coordonnées du minimum atteint
min_x, min_y = x_vals[-1], y_vals[-1]

# Affichage des résultats
print(f"Le minimum est atteint en ({min_x:.6f}, {min_y:.6f}) après {num_iterations} itérations.")
```

Méthodes d'optimisation de descente gradient

Méthode de gradient à pas fixe

- **Exemple. Soit** $f(x,y)=100x^2+y^2$,

t = 0.1

✅ Pas fixe : Minimum trouvé à x = -7505162419825238762053834565649869457584632414094586444475405823883810593104402820446297722346852774202751340888057865887442534 4.0000, y = 0.0000, f = 5632746294795702855005384171848075278325783840893797673321886927282627655286467056067599979135528671727753376739647854137106034750900551310644981513511697645653011468495239310876506830431191149025078805883594199646025774426275180944519952933903329062918553 60.0000, en 100 itérations

t = 0.01

✅ Pas fixe : Minimum trouvé à x = -1.0000, y = 0.1326, f = 100.0176, en 100 itérations

Méthodes d'optimisation de descente gradient

Méthode de gradient à pas optimal

- **Exemple. Soit** $f(x,y)=100x^2+y^2$

```python
# Algorithme de recherche linéaire (Armijo)
def line_search_armijo(x, grad, t=1e-2, beta=0.7, alpha=1, max_iter=100):
    """Recherche linéaire avec la condition d'Armijo pour déterminer le pas optimal"""
    for _ in range(max_iter):
        if f_lambdified(*(x - alpha * grad)) <= f_lambdified(*x) - t * alpha * np.linalg.norm(grad)**2:
            return alpha  # Retourner le pas optimal
        alpha *= beta  # Réduire le pas si la condition n'est pas satisfaite
    return alpha  # Retourne le dernier alpha testé
```

Méthodes d'optimisation de descente gradient

Méthode de gradient à pas optimal

- **Exemple. Soit** $f(x,y)=100x^2+y^2$

```python
# Fonction de descente de gradient avec condition d'arrêt et recherche linéaire
def gradient_descent_optimal(f, grad_f_numeric, x0, y0, epsilon, max_iterations):

    x, y = x0, y0
    trajectory = [(x, y)]  # Liste pour suivre la trajectoire des points
    for iteration in range(max_iterations):
        grad = grad_f_numeric(x, y)  # Calcul du gradient
        norm_grad = np.linalg.norm(grad)  # Norme du gradient

        # Condition d'arrêt basée sur la norme du gradient
        if norm_grad < epsilon:
            return trajectory, iteration + 1

        # Recherche linéaire pour le pas optimal
        t = line_search_armijo(np.array([x, y]), grad)

        # Mise à jour des variables
        x -= t * grad[0]
        y -= t * grad[1]

        # Ajouter le point actuel à la trajectoire
        trajectory.append((x, y))

    return trajectory, max_iterations  # Retourne la trajectoire et le nombre d'itérations
```

Méthodes d’optimisation de descente gradient

Méthode de gradient à pas fixe vs optimal

- **Exemple. Soit** $f(x,y)=100x^2+y^2$

```
✅ Pas optimal : Minimum trouvé à x = -0.0027, y = 0.1316, f = 0.0180, en 100 itérations
✅ Pas fixe : Minimum trouvé à x = -75051624198252387620538345656498694575846324140945864444754058238838105931
044028204462977223468527742027513408880578658874425344.0000, y = 0.0000, f = 563274629479570285500538417184807
527832578384089379767332188692728262765528646705606759997913552867172775337673964785413710603475090055131064449
815135116976456530114684952393108765068304311911490250788058835941996460257744262751809445199529339033290629 18
55360.0000, en 100 itérations
```

✅ Pas optimal : Minimum trouvé à x = -0.0027, y = 0.1316, f = 0.0180, en 100 itérations
✅ Pas fixe : Minimum trouvé à x = -1.0000, y = 0.1326, f = 100.0176, en 100 itérations

- Bien que le nombre d'itérations soit le même dans les deux cas (100), la **qualité de la solution** obtenue avec la descente à pas optimal est bien meilleure.
- Cela montre que la recherche linéaire adapte le pas pour mieux naviguer dans la fonction, tandis que le pas fixe n'a pas réussi à s'adapter aux changements de la fonction et a conduit à un échec de convergence.

Méthodes d’optimisation de descente gradient

Méthode de gradient conjugué

- Les méthodes du gradient conjugué sont utilisées pour résoudre des systèmes d'équations **linéaires** de la forme Ax = b, où la matrice A est **symétrique** et **définie positives.**

- **Définition.** Soit A une matrice symétrique nxn, définie positive. On dit que deux vecteurs x et y de $\mathbb{R}^n$, sont A− conjugués (ou conjugués par rapport à A) s'ils vérifient

$$x^T A y = 0$$

Méthodes d’optimisation de descente gradient

- **Méthode de gradient conjugué**

Théorème. Existence et unicité du minimum pour une fonction quadratique

Soit une fonction quadratique de la forme :

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

A est une matrice symétrique définie positive ($A^T=A$, $x^T A x>0$ pour tout $x\neq 0$) , $b \in \mathbb{R}^n$ et c une constante $\in \mathbb{R}$.

Alors, cette fonction admet un **unique minimum global**, donné par:

$$x*=A^{-1}b$$

Méthodes d'optimisation de descente gradient

- **Méthode de gradient conjugué**

  Étant donnés un point initial x_0 de $\mathbb{R}^{\mathbf{n}}$ et n-directions A−conjugués.

  L'idée de la méthode est de construire itérativement des directions d_1, ..., d_k mutuellement conjuguées.

  Sachant que :

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

  car $\nabla(\frac{1}{2} x^T A x)=Ax$, $\nabla(b^T x)=b$

  on note le résidu ou erreur, qui permet d'évaluer si on est proche de la solution. : $r(x) = b - Ax = -\nabla f(x)$.

  Le pas α_k détermine de combien avancer dans la direction d_k pour minimiser f(x)

Méthodes d'optimisation de descente gradient

Algorithme 5 : descente du gradient conjugué

1. Initialisation : $x_0 \in \mathbb{R}^n$, f une fonction quadratique symétrique définie positive, ε : critère d'arrêt (tolérance sur le gradient), max_iter :nombre maximal d'itérations. k=0

2. Calculer le **résidu initial** : $r_0 = b - Ax_0$,

3. Poser la **direction initiale** :$d_0 = r_0$

4. Répéter

a) Calculer le **pas optimal** minimisant $\alpha_k = \frac{r_k^T r_k}{d_k^T A d_k}$.

b) Mettre à jour la **solution** $x_{k+1} = x_k + \alpha_k d_k$

c) Calculer le **nouveau résidu :** $r_{k+1} = r_k - \alpha_k A d_k$

d) Calculer le **coefficient de conjugaison :** $\beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$.

e) Mettre à jour la **nouvelle direction :** $d_{k+1} = r_{k+1} + \beta_k d_k$

f) k++

5. jusqu'à ce que $\|r_k\| \leq \epsilon$ ou k> max_iter

6. Retourner x_k solution approximative de Ax=b.

Méthodes d’optimisation de descente gradient

- **Méthode de gradient conjugué**
- Exemple. Trouver le minimum de $f(x,y)= 3(x^2 + y^2)$.
  1. Calculer le gradient de f
  2. Calculer sa hessienne et déduire sa forme quadratique : $f(x)= \frac{1}{2} x^T A x - b^T x + c$
  3. Montrer que f est SDP.
  4. en prenant comme point de départ (x0; y0) = (1; 1):, appliquer la méthode des gradients conjuguées pour trouver le minimum.

Méthodes d’optimisation de descente gradient

- **Méthode de gradient conjugué**
- Exemple. Trouver le minimum de $f(x,y)= 3(x^2 + y^2)$.

  1) calculons le gradient de f : $\nabla f(x,y) =(6x,6y)$

Méthodes d'optimisation de descente gradient

- **Méthode de gradient conjugué**
- **Exemple.** Trouver le minimum de $f(x,y)= 3(x^2 + y^2)$.

  1) Calculons le gradient de f : $\nabla f(x,y) =(6x,6y)$

  2) Calculons sa hessienne : $Hf(x,y)=\begin{pmatrix}6 & 0\\ 0 & 6\end{pmatrix}=A$

  on vérifie que $f(x,y)= \frac{1}{2} x^T A x= \frac{1}{2} (x,y) \begin{pmatrix}6 & 0\\ 0 & 6\end{pmatrix}\begin{pmatrix}x\\ y\end{pmatrix}$, et b=c=0.

  3) $f(x,y)>=0$, donc $x^T A x>=0$ (forme quadratique), donc définie positive, en plus d'être symétrique (SDP).

  4) Recherche du minimum

Méthodes d'optimisation de descente gradient

- **Méthode de gradient conjugué**
- Exemple. Trouver le minimum de f(x,y)= $3(x^2 + y^2)$.

4) Recherche du minimum

a)initialisation : $(x_0,y_0)=(1,1)$, ε=1e-4

Calcul du **résidu initial** : $r_0=-Ax_0= -\begin{pmatrix}6 & 0\\ 0 & 6\end{pmatrix}\begin{pmatrix}1\\1\end{pmatrix}=\begin{pmatrix}-6\\-6\end{pmatrix}$

Calcule de la **direction initiale** :$d_0= \begin{pmatrix}-6\\-6\end{pmatrix}$

b)Calculer le **pas optimal** minimisant f $\alpha_0=\frac{r_0^T\, r_0}{d_0^T A d_0}$.

$r_0^T\, r_0 = (-6,-6)\begin{pmatrix}-6\\-6\end{pmatrix} = 72, d_0^T A d_0 = (-6,-6)\begin{pmatrix}6 & 0\\ 0 & 6\end{pmatrix}\begin{pmatrix}-6\\-6\end{pmatrix} = \mathbf{(36)\times 6 + (36)\times 6 = 216 + 216 = 432}$, $\alpha_0=\frac{1}{6}$

Méthodes d'optimisation de descente gradient

- **Méthode de gradient conjugué**
- **Exemple.** Trouver le minimum de $f(x,y)= 3(x^2 + y^2)$.

4) Recherche du minimum

c) Mettre à jour la **solution** $x_1 = x_0 + \alpha_0 d_0 = \begin{pmatrix}1\\1\end{pmatrix} + \frac{1}{6}\begin{pmatrix}-6\\-6\end{pmatrix} = \begin{pmatrix}0\\0\end{pmatrix}$

d)Calcul du **nouveau résidu :** $r_1 = r_0 - \alpha_0 A d_0 = \begin{pmatrix}-6\\-6\end{pmatrix} - \frac{1}{6}\begin{pmatrix}6 & 0\\0 & 6\end{pmatrix}\begin{pmatrix}-6\\-6\end{pmatrix} = \begin{pmatrix}0\\0\end{pmatrix}$

Sans continuer le calcul de β_0 et d_1, on voit que $\|r_0\| \leq \epsilon$? donc on arrête l'algorithme et le minimum est atteint dans la première itération (0,0).

Méthodes d'optimisation de descente gradient

- **Méthode de gradient conjugué**
- Exemple. Trouver le minimum de $f(x,y)= 5x^2+\frac{1}{2}y^2-3(x+y)$
  1. Calculer le gradient de f
  2. Calculer sa hessienne et déduire sa forme quadratique : $f(x)= \frac{1}{2} x^T A x - b^T x + c$
  3. Montrer que f est SDP.
  4. en prenant comme point de départ (x0; y0) = (1; 1):, appliquer la méthode des gradients conjuguées pour trouver le minimum.

Méthodes d'optimisation de descente gradient

- **Méthode de gradient conjugué**
- Exemple. Trouver le minimum de $f(x,y)= 5x^2+\frac{1}{2}y^2-3(x+y)$

  1) calculons le gradient de f : $\nabla f(x,y) =(10x-3, y-3)$;

  2) Calculons sa hessienne : $Hf(x,y)=\begin{pmatrix}10 & 0\\ 0 & 1\end{pmatrix}$

  3) on vérifie que $f(x,y)= \frac{1}{2} x^T A x- b^T x + c = \frac{1}{2} (x,y) \begin{pmatrix}10 & 0\\ 0 & 1\end{pmatrix}\begin{pmatrix}x\\ y\end{pmatrix}=$

  $\frac{1}{2} (x,y)\begin{pmatrix}10x\\ y\end{pmatrix} - bTx + c= 5 x^2+\frac{1}{2}. y^2- b^T x + c$, donc $b=\begin{pmatrix}3\\ 3\end{pmatrix}$ et $c=0$;

  $x^T A x=5 x^2+\frac{1}{2}. y^2\geq 0$(forme quadratique), donc f définie positive, en plus d'être symétrique.

Méthodes d'optimisation de descente gradient

- **Méthode de gradient conjugué**
- Exemple. Trouver le minimum de $f(x,y)= 5x^2+\frac{1}{2}y^2-3(x+y)$

  1) a)initialisation : $(x_0,y_0)=(1,1)$, ε=1e-4

  Calcul du **résidu initial** : $r_0= b-Ax_{0=} \begin{pmatrix}3\\3\end{pmatrix} - \begin{pmatrix}10 & 0\\0 & 1\end{pmatrix}\begin{pmatrix}1\\1\end{pmatrix}=\begin{pmatrix}3\\3\end{pmatrix}-\begin{pmatrix}10\\1\end{pmatrix}= \begin{pmatrix}-7\\2\end{pmatrix}$

  Calcule de la **direction initiale** :$d_0= \begin{pmatrix}-7\\2\end{pmatrix}$

  Implémenter en python avec epsilon =1e-6

Méthodes d'optimisation de descente gradient

- **Méthode de gradient conjugué**
- Exemple. Trouver le minimum de $f(x,y)= 5x^2+\frac{1}{2}y^2-3(x+y)$

```python
import numpy as np
#gradient conjugué
# Définir la fonction de calcul de A * x
def Ax(A, x):
    return np.dot(A, x)

# Algorithme du Gradient Conjugué avec d_k comme direction de recherche
def gradient_conjugue(A, b, x0, tol, max_iter):
    # Initialisation
    r0 = b - Ax(A, x0)  # Résidu initial
    d0 = r0  # Direction de recherche initiale (remplaçant p0 par d0)
    xk = x0  # Solution initiale
    rk = r0  # Résidu actuel
    dk = d0  # Direction de recherche actuelle
    k = 0  # Compteur d'itérations

    print(f"Initialisation : x0 = {xk}, r0 = {rk}, d0 = {dk}")
```

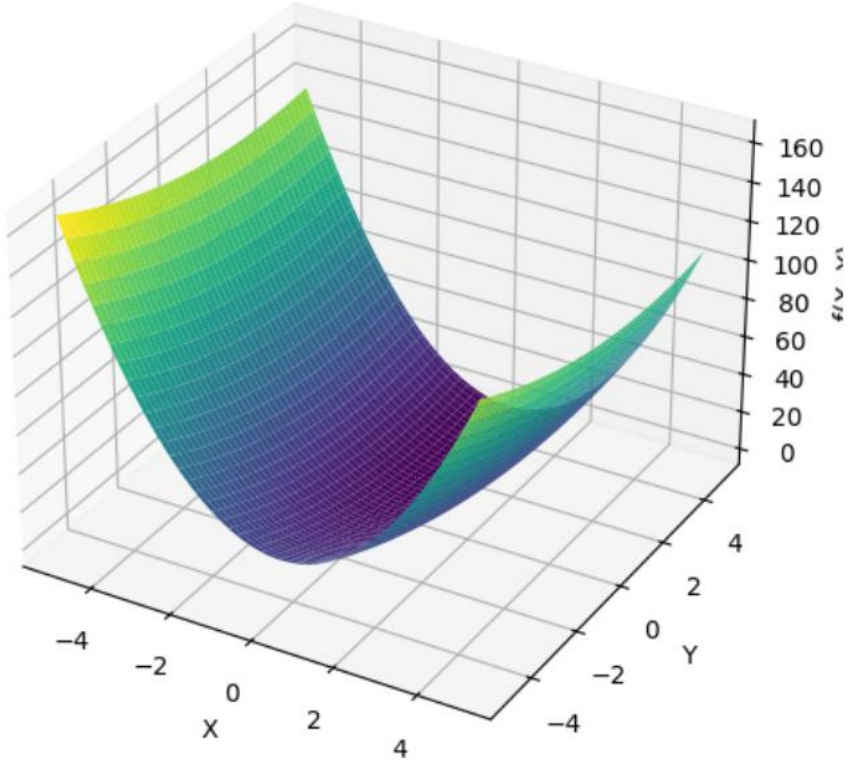
```
    while np.linalg.norm(rk) > tol and k < max_iter:
# Calcul du pas optimal alpha_k
        alpha_k = np.dot(rk, rk) / np.dot(dk, Ax(A, dk))
        print(f"Itération {k+1}:")
        print(f"  alpha_k = {alpha_k}")

# Mise à jour de la solution
        xk = xk + alpha_k * dk
        print(f"  xk+1 = {xk}")

# Mise à jour du résidu
        rk1 = rk - alpha_k * Ax(A, dk)
        print(f"  rk+1 = {rk1}")

 # Critère d'arrêt basé sur le résidu
        print(f"  ||rk+1|| = {np.linalg.norm(rk1)}")

 # Calcul du coefficient de conjugaison beta_k
        beta_k = np.dot(rk1, rk1) / np.dot(rk, rk)
        print(f"  beta_k = {beta_k}")
 # Mise à jour de la direction
        dk1 = rk1 + beta_k * dk
        print(f"  dk+1 = {dk1}")

# Passage à la prochaine itération
        rk = rk1
        dk = dk1
        k += 1
    return xk
```

```
# Exemple d'utilisation
A = np.array([[10, 0], [0, 1]])  # Matrice symétrique définie positive
b = np.array([3, 3])  # Vecteur du second membre
x0 = np.array([0, 0])  # Point de départ initial
# Appel de l'algorithme
solution = gradient_conjugue(A, b, x0, tol=1e-4, max_iter=100)
print(f"Solution approximative x = {solution}")
```

```
Initialisation : x0 = [0 0], r0 = [3 3], d0 = [3 3]
Itération 1:
  alpha_k = 0.18181818181818182
  xk+1 = [0.54545455 0.54545455]
  rk+1 = [-2.45454545  2.45454545]
  ||rk+1|| = 3.4712514712794156
  beta_k = 0.669421487603306
  dk+1 = [-0.44628099  4.46280992]
Itération 2:
  alpha_k = 0.5500000000000002
  xk+1 = [0.3 3. ]
  rk+1 = [-4.4408921e-16 -8.8817842e-16]
  ||rk+1|| = 9.930136612989092e-16
  beta_k = 8.183485042158984e-32
  dk+1 = [-4.4408921e-16 -8.8817842e-16]
Solution approximative x = [0.3 3. ]
```

Méthodes d’optimisation de descente gradient

- **Méthode de gradient conjuguéExemple.**
- Exemple. Trouver le minimum de $f(x,y)= 5x^2+\frac{1}{2}y^2-3(x+y)$



Méthodes d’optimisation de descente gradient

- **Méthode de gradient conjugué**
  - Pour les matrices symétriques définies positives, le gradient conjugué converge théoriquement en au plus **n itérations**, où n est la taille de la matrice.
  - Cette méthode est plus efficace que la **descente de gradient**, surtout pour les grands systèmes.
  - Cependant, elle est sensible aux erreurs numériques,
  - Elle est inefficace pour les matrices mal conditionnées, et exige une matrice SDP.

Méthodes d’optimisation de descente gradient

- **Méthode de Newton**
  - La méthode newton permet de résoudre des équations de type f(x) = 0, et en particulier, si f = F′ on peut calculer les points critiques de F.
  - **Bien que Newton appartient à la famille des méthodes de descente,** il ne s'agit pas d'une **descente de gradient classique**.
  - Elle appartient plus précisément aux **méthodes de second ordre**, qui exploitent la matrice **hessienne** pour ajuster la direction de mise à jour.
  - Dans ce qui suit on cherche à minimiser une fonction f : $\mathbb{R}^n \to \mathbb{R}$ de classe C^2.
  - La méthode de newton est basée sur l'approximation quadratique de la fonction $f(x)$ autour d'un point x_n, en utilisant le développement de Taylor au second ordre (modèle quadratique) .
  - Rappelons la formule de Taylor suivant au point x.

Méthodes d'optimisation de descente gradient

- **Méthode de Newton**
  - Dans ce qui suit on cherche à minimiser une fonction f : $\mathbb{R}^n \to \mathbb{R}$ de classe C^2.
  - La méthode de newton est basée sur l'approximation quadratique de la fonction $f(x)$ autour d'un point x_n, en utilisant le développement de Taylor au second ordre (modèle quadratique) .
  - Rappelons la formule de Taylor suivant au point x.

$$f(x+h) \approx f(x) + h^T \nabla f(x) + \frac{1}{2!} h^T \nabla^2 f(x)\, h.$$

Remplaçons x par x_n, on obtient :

$$f(x_n+h) \approx f(x_n) + h^T \nabla f(x_n) + \frac{1}{2!} h^T \nabla^2 f(x_n)\, h.$$

  - L'idée est de trouver un point x_{n+1} qui minimise l'**approximation quadratique** $P_{x_n}(h) : \mathbb{R}^n \to \mathbb{R}$, avec.

$$P_{x_n}(x_n+h) = f(x_n) + h^T \nabla f(x_n) + \frac{1}{2!} h^T \nabla^2 f(x_n)\, h.$$

Méthodes d'optimisation de descente gradient

- **Méthode de Newton**

Les conditions de premier ordre donnent (on dérive par rapport à h).

$$\nabla P_{x_n}(x+h) = 0 = \nabla\left(f(x_n) + h^T \nabla f(x_n) + \frac{1}{2!} h^T \nabla^2 f(x_n)\, h\right),$$

Sachant que la hessienne est une matrice symétrique ($\nabla(p^T A p) = 2Ap$), on a :

$\nabla(f(x_n) = 0$ (ne depend pas de h), et $\nabla f(x_n) + \frac{1}{2!}(2\, \nabla^2 f(x_n)\, h) = 0$

Donc on obtient :

$$h = -\frac{\nabla f(x_n)}{\nabla^2 f(x_n}$$

Posons $x_{n+1} = x_n + h$, on obtient $x_{n+1} = x_n - \frac{\nabla f(x_n)}{\nabla^2 f(x_n)}$

Méthodes d'optimisation de descente gradient

- **Méthode de Newton**

Algorithme 6 : Méthode de Newton

1. **Initialisation** $x_0, \varepsilon, \mathrm{n} = 0, \max\_\mathrm{iter}$
2. **Répéter jusqu'à convergence** :
   a) Calculer le **gradient** $\nabla \mathrm{f}(x_n)$.
   b) Calculer la **matrice Hessienne** $\nabla^2 f(x_n)$
   c) Résoudre le système linéaire : $p_n = -\frac{\nabla \mathrm{f}(x_n).}{\nabla^2 f(x_n)}$
   d) Mettre à jour la solution : $x_{n+1} = x_n + p_n$
   e) n++
3. **Vérifier la condition d'arrêt : $\|\nabla \mathrm{f}(x_n)\| < \varepsilon$ ou n>max_iter**
4. **Retourner la solution x_n**

Méthodes d’optimisation de descente gradient

- **Méthode de Newton**

Exemple.

Soit la fonction suivante : $f(x_1,x_2,x_3)= \frac{1}{4}x_3^4 + x_2^3 - \frac{1}{3}x_3^3 + 6x_1^2 + 3x_2^2 + 6x_1\,x_2$

1) Évaluer numériquement la fonction objective en utilisant l’algorithme de Newton.

Méthodes d’optimisation de descente gradient

- **Méthode de Newton**

Exemple.

Soit la fonction suivante : $f(x_1,x_2,x_3)= \frac{1}{4}x_3^4 + x_2^3 - \frac{1}{3}x_3^3 + 6x_1^2 + 3x_2^2 + 6x_1\,x_2$

1) Évaluer numériquement la fonction objective en utilisant l’algorithme de Newton.

Méthodes d'optimisation de descente gradient

- **Méthode de Newton**

Exemple.

```
# Définition des variables symboliques
x1, x2, x3 = sp.symbols('x1 x2 x3')
variables = [x1, x2, x3]

# Définition de la fonction symbolique
f_sym = (1/4) * x3**4 + x2**3 - (1/3) * x3**3 + 6 * x1**2 + 3 * x2**2 + 6 * x1 * x2

# Calcul automatique du gradient et de la hessienne
grad_f_sym = [sp.diff(f_sym, var) for var in variables]
hess_f_sym = [[sp.diff(g, var) for var in variables] for g in grad_f_sym]

# Conversion en fonctions numériques
grad_f_numeric = sp.lambdify(variables, grad_f_sym, 'numpy')
hess_f_numeric = sp.lambdify(variables, hess_f_sym, 'numpy')
```

Méthodes d'optimisation de descente gradient

- **Méthode de Newton**

Exemple.

```python
def newton_method(x0, tol, max_iter):
    x = np.array(x0, dtype=float)
    iter_count = 0

    while np.linalg.norm(np.array(grad_f_numeric(*x), dtype=float)) >= tol and iter_count < max_iter:
        grad = np.array(grad_f_numeric(*x), dtype=float)
        hess = np.array(hess_f_numeric(*x), dtype=float)

        if np.linalg.det(hess) == 0:
            print("La matrice Hessienne est singulière, arrêt de l'algorithme.")
            return x

        p = -np.linalg.inv(hess) @ grad  # Résolution du système Hessien
        x += p
        iter_count += 1

        print(f"Iteration {iter_count}: x = {x}, Gradient Norm = {np.linalg.norm(grad)}")

    return x, iter_count

# Point initial
x0 = [1.0, 1.0, 1.0]
solution, iter_count = newton_method(x0, tol=1e-6, max_iter=100)
print("Solution trouvée :", solution, "en un nombre d'itérations", iter_count)
```

Méthodes d’optimisation de descente gradient

- **Méthode de Newton**

Exemple. tol=1e-6, max_iter=100

```
Iteration 1: x = [-0.16666667  0.33333333  1.        ], Gradient Norm = 23.430749027719962
Iteration 2: x = [-0.03333333  0.06666667  1.        ], Gradient Norm = 1.3333333333333355
Iteration 3: x = [-0.00196078  0.00392157  1.        ], Gradient Norm = 0.2133333333333333
Iteration 4: x = [-7.62951095e-06  1.52590219e-05  1.00000000e+00], Gradient Norm = 0.011810841983852407
Iteration 5: x = [-1.16415322e-10  2.32830644e-10  1.00000000e+00], Gradient Norm = 4.5777764203336834e-05
Solution trouvée : [-1.16415322e-10  2.32830644e-10  1.00000000e+00] en un nombre d'itérations 5
```

```
Iteration 1: x = [-0.16666667  0.33333333  1.        ], Gradient Norm = 23.430749027719962
Iteration 2: x = [-0.03333333  0.06666667  1.        ], Gradient Norm = 1.3333333333333355
Iteration 3: x = [-0.00196078  0.00392157  1.        ], Gradient Norm = 0.2133333333333333
Iteration 4: x = [-7.62951095e-06  1.52590219e-05  1.00000000e+00], Gradient Norm = 0.011810841983852407
Iteration 5: x = [-1.16415322e-10  2.32830644e-10  1.00000000e+00], Gradient Norm = 4.5777764203336834e-05
Iteration 6: x = [-2.71050802e-20  5.42101086e-20  1.00000000e+00], Gradient Norm = 6.984919312767051e-10
Iteration 7: x = [0. 0. 1.], Gradient Norm = 1.6263017077675663e-19
Solution trouvée : [0. 0. 1.] en un nombre d'itérations 7
```

tol=1e-100, max_iter=100

Méthodes d’optimisation de descente gradient

- **Méthode de Newton**
  - Si on détecte une matrice hessienne singulière, on peut utiliser la pseudo-inverse de **Moore-Penrose** au lieu de l'inverse classique.
  - On peut aussi ajouter une régularisation en modifiant légèrement la hessienne : $H\text{mod}=H+\lambda I$, I la matrice identité et λ un petit facteur (>0).
  - Changer de méthode : Passer à une méthode plus robuste comme le gradient conjugué ou la quasi-Newton (BFGS).

Méthodes d’optimisation de descente gradient

- **Méthode de Newton**
  - Il existe d’autres versions de l’algorithme de Newton : Newton-Raphson, Newton tronquée, Gauss-Newton, .., et quasi-Newton
- **Avantages :**
  - Très performante lorsque la fonction est deux fois différentiable et que la hessienne est définie positive.
  - Atteint des solutions précises en peu d'itérations.
  - **Convergence Super-linéaire voire quadratique** : Si le point de départ est proche de la solution, la convergence est très rapide.
- **Inconvénients :**
  - Nécessite le calcul du gradient et de la hessienne, ce qui est coûteux pour des fonctions complexes ou de grande dimension.
  - Un mauvais choix peut entraîner divergence ou convergence vers un minimum local.
  - L'algorithme échoue si la matrice hessienne est singulière.
  - L'inversion de la hessienne devient impraticable pour de très grandes dimensions.