# Optimisation Continue et Programmation Linéaire

Razafinjatovo Heriniaina

IT University

# Plan

# Introduction

L'optimisation est une branche essentielle des mathématiques appliquées et de l'informatique décision. Elle consiste à trouver la meilleure solution (maximale ou minimale) à un problème donné, sous certaines contraintes. Deux grandes familles de problèmes sont abordées ici :

- **L'optimisation continue**, où les variables peuvent prendre toutes les valeurs réelles dans un certain domaine.
- **La programmation linéaire**, cas particulier de l'optimisation continue où la fonction objectif et les contraintes sont linéaires.

# Plan

# Optimisation continue

## Définition

Un problème d'optimisation continue consiste à :

$$\text{Minimiser ou maximiser } f(x)$$
$$\text{sous les contraintes : } x \in D \subset \mathbb{R}^n$$

## Cas d'utilisation

- Apprentissage automatique (descente de gradient)
- Controle optimal
- Ajustement de modèles statistiques

## Outils théoriques

- Dérivée et gradient
- Conditions de Karush-Kuhn-Tucker (KKT)
- Convexité, points selles

## Exemple illustré

**Problème :** Minimiser $f(x) = x^2 + 4x + 4$
**Solution :** $f(x) = (x+2)^2$, minimum atteint en $x = -2$

# Plan

# Programmation linéaire

## Définition

Un problème de programmation linéaire consiste à :

$$\text{Maximiser (ou minimiser) } Z = c_1x_1 + c_2x_2 + \cdots + c_nx_n$$

$$\text{sous les contraintes } \sum_{j=1}^{n} a_{ij}x_j \leq b_i \quad (i = 1, \ldots, m)$$

$$x_j \geq 0 \quad \text{(positivité)}$$

## Cas d'utilisation

- Logistique et transport
- Planification de production
- Affectation de ressources

## Méthode du simplexe (idée générale)

- D'epart d'un sommet admissible de la région des solutions
- Passage de sommet en sommet avec amélioration de la valeur de la fonction objectif
- Arret lorsqu'on atteint un optimum

## Exemple illustré

$$\begin{aligned} \text{Maximiser } & Z = 3x + 2y \\ \text{sous } & x + y \leq 4 \\ & x \leq 2 \\ & y \leq 3 \\ & x, y \geq 0 \end{aligned}$$

**Solution graphique :** Représenter les contraintes dans le plan, calculer la valeur de Z aux sommets de la région admissible . L'optimum est atteint en $(2, 2)$ avec $Z = 10$.

# Plan

# scipy.optimize.minimize

`scipy.optimize.minimize` permet de minimiser une fonction scalaire en utilisant différentes méthodes d'optimisation.

- Recherche d'un minimum local.
- Supporte plusieurs algorithmes d'optimisation.
- Gestion des contraintes et des bornes.

# Syntaxe de base

```
from scipy.optimize import minimize
result = minimize(fun, x0, method='BFGS',
constraints=(), bounds=(), options={})
```

**Paramètres** :

- `fun` : Fonction à minimiser.
- `x0` : Point initial.
- `method` : Algorithme d'optimisation.
- `constraints` : Contraintes éventuelles.
- `bounds` : Bornes sur les variables.

# Minimisation de la fonction $f(x) = (x-3)^2$

```
import numpy as np
from scipy.optimize import minimize

# Fonction à minimiser
def f(x):
return (x - 3)**2

# Point initial
x0 = np.array([0])

# Appel de minimize
result = minimize(f, x0, method="BFGS")

# Affichage des résultats
print("Solution optimale :", result.x)
print("Valeur de la fonction en minimum :", result.fun)
```

- result.x : Il s'agit de la solution optimale trouvée.
- result.fun : C'est le minimum de la fonction objective, c'est-à-dire la valeur de la fonction évaluée en result.x.

# On cherche à minimiser $f(x) = x_0^2 + x_1^2$ sous la contrainte $x_0 + x_1 = 1$

```
def objective(x):
return x[0]**2 + x[1]**2

# Définition de la contrainte x0 + x1 = 1
constraint = {'type': 'eq', 'fun':  x[0] + x[1] - 1}

# Point initial
x0 = [0.5, 0.5]

# Appel de minimize avec contrainte
result = minimize(objective, x0,
constraints=[constraint], method="SLSQP")

print("Solution optimale :", result.x)
print("Valeur de la fonction en minimum :", result.fun)
```

# Algorithmes d'optimisation courantes

- "BFGS" : Méthode de quasi-Newton, efficace pour problèmes non contraints.
- "L-BFGS-B" : Variante de BFGS qui gère des bornes sur les variables.
- "Nelder-Mead" : Approche sans gradient, utile pour fonctions non différentiables.
- "SLSQP" : Gère les contraintes d'égalité et d'inégalité.
- "COBYLA" : Optimisation sans gradient avec contraintes d'inégalité uniquement.

# scipy.optimize.linprog

La fonction `scipy.optimize.linprog` est utilisée pour résoudre des problèmes d'optimisation linéaire sous contraintes. Elle permet de minimiser une fonction objectif linéaire sous des contraintes d'égalité et d'inégalité.

# Formulation du problème

L'optimisation linéaire vise à résoudre :

$$\min \mathrm{c}^T \mathrm{x}$$

sous les contraintes :

$$A_{ub}\mathrm{x} \leq b_{ub} \quad \text{(contraintes d'inégalité)}$$

$$A_{eq}\mathrm{x} = b_{eq} \quad \text{(contraintes d'égalité)}$$

$$\mathrm{x} \geq 0 \quad \text{(contraintes de positivité, par défaut)}$$

# Syntaxe de base

```
from scipy.optimize import linprog

result = linprog(
c,              # Coefficients de la fonction objectif
A_ub=A_ub,      # Matrice des coefficients des
contraintes d'inégalité
b_ub=b_ub,␣␣␣␣␣#␣Vecteur␣des␣constantes␣des␣contraintes
d'inégalité
A_eq=A_eq,      # Matrice des coefficients des
contraintes d'égalité
b_eq=b_eq,␣␣␣␣␣#␣Vecteur␣des␣constantes␣des␣contraintes
d'égalité
bounds=bounds, # Bornes des variables(par défaut,x>=0)
method='highs' # Algorithme utilisé ('highs','highs-ds',
'highs-ipm', etc.)
                          )
```

# Exemple : Problème simple

**Problème :**

$$\min z = 3x_1 + 2x_2$$

sous :

$$x_1 + 2x_2 \leq 6, \quad 4x_1 + 2x_2 \leq 12, \quad -x_1 + x_2 \leq 1, \quad x_1, x_2 \geq 0.$$

# Code Python

```
from scipy.optimize import linprog
# Coefficients de la fonction objectif
c = [3, 2]
# Coefficients des contraintes d'inégalité
(A_ub * x <= b_ub)
A_ub = [
[1, 2],
[4, 2],
[-1, 1]  ]
b_ub = [6, 12, 1]
# Résolution du problème
result = linprog(c, A_ub=A_ub, b_ub=b_ub, method='highs')
# Affichage du résultat
if result.success:
print("Solution optimale trouvée :")
print(f"x1={result.x[0]:.2f}, x2={result.x[1]:.2f}")
print(f"Valeur optimale de la fonction objectif:
{result.fun:.2f}")
else:
print("Echec de la résolution.")
```

# Interprétation des résultats

- `result.x` : Valeurs optimales des variables $x_1, x_2$.
- `result.fun` : Valeur optimale de la fonction objectif.
- `result.success` : booléen indiquant si la solution a été trouvée.
- `result.message` : Explication du résultat.

# Méthodes disponibles

`linprog` propose plusieurs solveurs :

- `'highs'` (par défaut) : Solveur HiGHS.
- `'highs-ds'` : Simplexe dual.
- `'highs-ipm'` : Méthode des points intérieurs.