Optimisation mathématique : introduction et application à la gestion de l'eau

UCA – Master Hydroprotech

Sophie Demassey (CMA, Mines Paris – PSL)
sophie.demassey@minesparis.psl.eu http://sofdem.github.io/

Optimisation linéaire

Modèle d'optimisation mathématique

définition : programme mathématique

$$\begin{aligned} &\text{minimize } f(x) \\ &\text{subject to } g(x) \leq 0 \\ &x \in \mathbb{R}^n \end{aligned}$$

minimiser ou maximiser sur des contraintes $\leq$, $\geq$ ou $=$, mais jamais $>$ ou $<$

programme = **planification** (des opérations militaires ou logistiques)

- $x \in \mathbb{R}^n$: les n variables de décision
- $f : \mathbb{R}^n \to \mathbb{R}$: la fonction objectif — $\max f(x) \equiv -\min(-f)(x)$
- $g : \mathbb{R}^n \to \mathbb{R}^m$ définit m contraintes — $g(x) \leq 0 \equiv -g(x) \geq 0 \equiv g(x) + s = 0, s \geq 0$

solutions : $\mathbb{R}^n$
solutions réalisables : $\{x \in \mathbb{R}^n : g(x) \geq 0\}$
solution optimales : $\arg\min_{x \in \mathbb{R}^n} \{f(x) : g(x) \geq 0\}$

Modèle d'optimisation linéaire

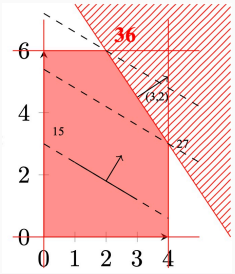
un programme mathématique $\min_{x \in \mathbb{R}^n} \{f(x) | g(x) \geq 0\}$ avec f et g linéaires/affines :
$f(x) = c^\top x$, $g(x) = Ax - b$ avec $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$.

définition : programme linéaire (PL)

$$\begin{aligned} &\min c^\top x \\ &\text{s.t. } Ax \geq b \\ &x \in \mathbb{R}^n \end{aligned} \qquad \begin{aligned} &\min \sum_{j=1}^{n} c_j x_j \\ &\text{s.t. } \sum_{j=1}^{n} a_{ij} x_j \geq b_i, && \forall i = 1, \dots, m \\ &x_j \in \mathbb{R} && \forall j = 1, \dots, n \end{aligned}$$

Résolution graphique

$$\begin{aligned} \max\ & 3x_1 + 5x_2 \\ \text{s.t. } & x_1 \le 40 \\ & x_2 \le 60 \\ & 3x_1 + 2x_2 \le 180 \\ & x_1, x_2 \ge 0 \end{aligned}$$



- contrainte linéaire = demi-plan de $\mathbb{R}^n$, solutions réalisables = polyèdre
- les solutions (x_1, x_2) de score p = le plan (ligne pointillée) $3x_1 + 5x_2 = p$
- solution optimale = sommet du polyèdre sur la ligne pointillée la plus haute : $(x_1 = 20, x_2 = 60)$ avec $p = 360$
- algorithme du simplexe : parcourt les sommets du polyèdre en suivant les arêtes *améliorantes*. Complexité théorique exponentielle, mais rapide en pratique.

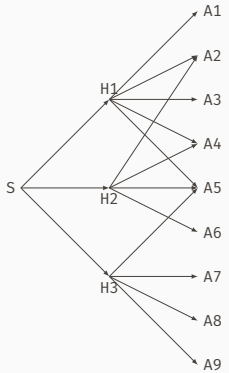
Ex : Graphes et flots

Un réseau d'irrigation branché depuis une unique source (S) dessert 9 sous-réseaux ($A_1 - A_9$) en passant par 3 points de distributions ($H_1 - H_3$)

Gérer l'irrigation pour :

- acheminer un débit (D_i) en entrée de chaque sous-réseau (A_i),
- ne pas excéder la capacité (K) à la source,
- ne pas excéder les capacités des canalisations (C_ℓ),
- minimiser les fuites : chaque canalisation ℓ perd $F_\ell\%$ du débit transporté

Ex : Flot dans un graphe



- graphe = canalisations (arcs) et jonctions (sommets)
- flot = débit eau

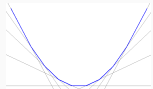
Ex : Modèle linéaire

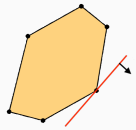
- x_ℓ débit entrant dans chaque canalisation $\ell = (i, j) \in L$
- $y_\ell = (1 - F_\ell\%)x_\ell$ débit sortant dans chaque canalisation $\ell = (i, j) \in L$
- $H = \{H_1, H_2, H_3\}$, $A = \{A_1, \ldots, A_9\}$

$$\begin{aligned} \min\ & \sum_{\ell \in L} F_\ell\% x_\ell \\ \text{s.t. } & \sum_{h \in H} x_{(S,h)} \le K \\ & \sum_{h \in H} y_{(h,a)} \ge D_a, && \forall a \in A \\ & y_{(S,h)} = \sum_{a \in A} x_{(h,a)}, && \forall h \in H \\ & y_\ell = (1 - F_\ell\%)x_\ell, && \forall \ell \in L \\ & 0 \le x_\ell \le C_\ell, && \forall \ell \in L. \end{aligned}$$

INTÉRÊT DE LA PROGRAMMATION LINÉAIRE

- **nombreuses applications :**
  modèle direct de problèmes pratiques,
  approximation de problèmes convexes,
  base de problèmes non convexes ou logiques
  (associés à des variables à valeur dans $\mathbb{Z}$)
- **facile à résoudre :**
  complexité polynomiale,
  propriétés fortes (dualité),
  algorithmes exacts efficaces,
  implémentations disponibles





OPTIMISATION COMBINATOIRE

MODÈLE D'OPTIMISATION LINÉAIRE EN NOMBRES ENTIERS

définition : programme linéaire en nombres entiers (PLNE)

$$\min \sum_{j=1}^{n} c_j x_j$$

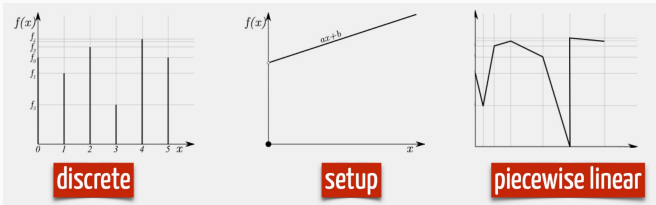
$$\text{s.t.} \sum_{j=1}^{n} a_{ij} x_j \geq b_i, \qquad \forall i = 1, \ldots, m$$

$$x_j \in \mathbb{R} \qquad \forall j = 1, \ldots, p$$

$$x_j \in \mathbb{Z} \qquad \forall j = p+1, \ldots, n$$

MODÉLISER AVEC DES VARIABLES ENTIÈRES

- décisions binaires : vrai/faux, on/off $x \in \{0, 1\}$
- décisions discrètes : compte $z \in \{0, 1, \ldots, N\}$, niveau $\{10, 30, 60\}$, état $\{-1, 0, 1\}$
- conditions logiques : *si* $x = 1$ *alors* $z = 0$: $z \leq N.(1 - x)$
- relations non-linéaires :



MODÉLISER DES CONDITIONS LOGIQUES

condition	example	linéarisation
exclusion	c *faux ou vrai*	$y \in \{0,1\}$
exclusion	*soit* c_1 *soit* c_2	$y_1 + y_2 = 1$
disjonction	c_1 *ou* c_2	$y_1 + y_2 \geq 1$
implication	*si* c_1 *alors* c_2	$y_2 \geq y_1$
alternative	1 *parmi* n	$\sum_{i=1}^{n} y_i = 1$
compteur	k *parmi* n	$\sum_{i=1}^{n} y_i = k$
borne	*au moins* k *parmi* n	$\sum_{i=1}^{n} y_i \geq k$
borne	*au plus* k *parmi* n	$\sum_{i=1}^{n} y_i \leq k$

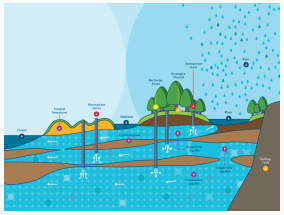
EXERCICE : MODÉLISER

extraction d'eau

décider de l'emplacement de pompes (toutes identiques) parmi un ensemble fini J de candidats pour minimiser le coût total, étant donnés :

- le coût d'installation C_j et le débit moyen Q_j d'une pompe à l'emplacement $j \in J$
- les limites minimale $Qmin$ et maximale $Qmax$ du débit moyen total d'extraction
- la limite maximale de 3 pompes installées
- l'interdiction de placer 2 pompes simultanément aux emplacement j_1 et j_2.

EXERCICE : EXTRACTION DE L'EAU



- coût C_j, débit C_j à l'emplacement $j \in J$
- débit min $Qmin$ et max $Qmax$
- nombre max 3 pompes
- exclusion j_1 et j_2

- $x_j \in \{0,1\}$: pompe installée en $j \in J$?
- minimisation du coût : $\sum_{j \in J} C_j x_j$
- limites de débit : $Qmin \leq \sum_{j \in J} Q_j x_j \leq Qmax$
- nombre max : $\sum_{j \in J} x_j \leq 3$
- exclusion : $x_1 + x_2 \leq 1$

EXERCICE : MODÈLE PLNE

$$
\begin{aligned}
\min \quad & \sum_{j \in J} C_j x_j \\
\text{s.t.} \quad & \sum_{j \in J} Q_j x_j \geq Qmin \\
& \sum_{j \in J} Q_j x_j \leq Qmax \\
& \sum_{j \in J} x_j \leq 3 \\
& x_1 + x_2 \leq 1 \\
& x_j \in \{0,1\}, \quad \forall j \in J
\end{aligned}
$$

EXERCICE : MODÉLISER

extraction d'eau (variante avec pompes individuelles)

les pompes sont maintenant choisies parmi un ensemble fini K et :

- le coût d'investissement C_k dépend uniquement de la pompe $k \in K$
- le débit moyen Q_{jk} dépend de la pompe $k \in K$ et de l'emplacement $j \in J$

$x_{jk} \in \{0,1\}$: pompe $k \in K$ installée en $j \in J$?

EXERCICE : MODÈLE PLNE

$x_{jk} \in \{0,1\}$: pompe $k \in K$ installée en $j \in J$?

$$\min \sum_{j\in J}\sum_{k\in K} C_k x_{jk}$$

$$\text{s.t. } Qmin \le \sum_{j\in J}\sum_{k\in K} Q_{jk}x_{jk} \le Qmax$$

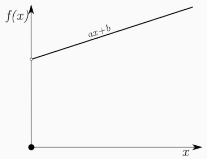
$$\sum_{j\in J} x_{jk} \le 1, \quad \forall k \in K$$

$$\sum_{k\in K} x_{jk} \le 1, \quad \forall j \in J$$

$$\sum_{k\in K} (x_{1k} + x_{2k}) \le 1$$

$$x_{jk} \in \{0,1\}, \quad \forall j \in J, k \in K.$$

MODÉLISER DES FONCTIONS NON-LINÉAIRES

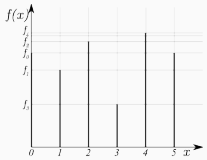


set-up :

$f(x) = ax + by$

$\epsilon y \le x \le Uy$

$y \in \{0,1\}$



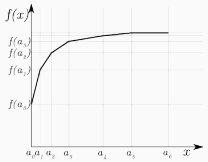
discret :

$f(x) = \sum_i y_i f_i$

$\sum_i i y_i = x$

$\sum_i y_i = 1$

$y_i \in \{0,1\}\ i = 0..n$



morceaux :

$f(x) = \sum_i \lambda_i f(a_i)$

$\sum_i a_i \lambda_i = x$

$\sum_i \lambda_i = 1$

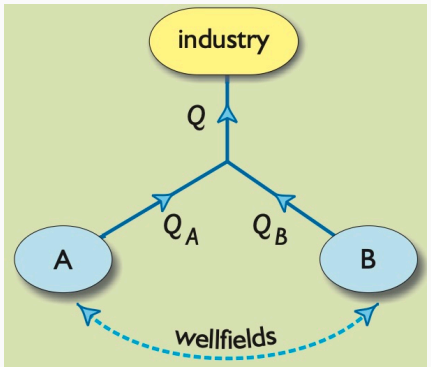
$\lambda_i \in [0,1]\ i = 0..n$

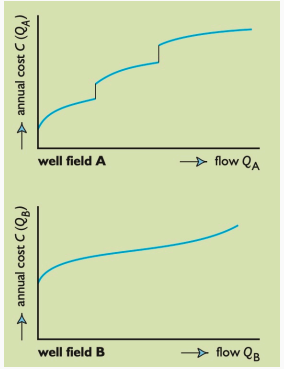
$SOS2(\lambda_i)$

EX : APPROVISIONNEMENT

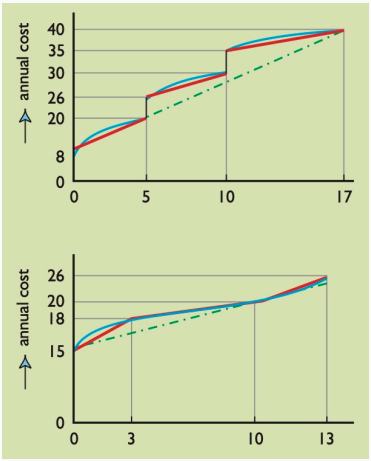
[Water Resources Systems Planning and Management, Unesco 2005]

Pour satisfaire sa demande en eau Q, une industrie exploite, à moindre coût, deux forages dont le coût opérationnel varie en fonction de l'extraction : $c = F(q)$.



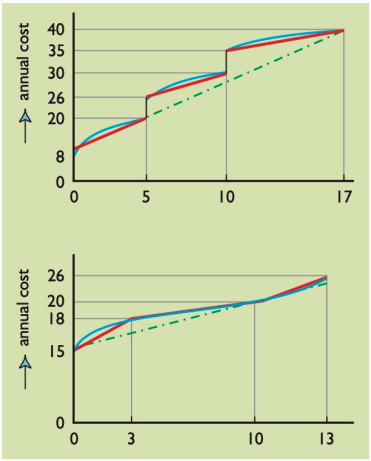


EX : APPROVISIONNEMENT



$$\begin{aligned}\min\ & c_a + c_b \\ \text{s.t. } & q_a + q_b = Q \\ & c_i = F_i(q_i), \quad \forall i = a, b \\ & 0 \le q_i \le Q_i, \quad \forall i = a, b\end{aligned}$$

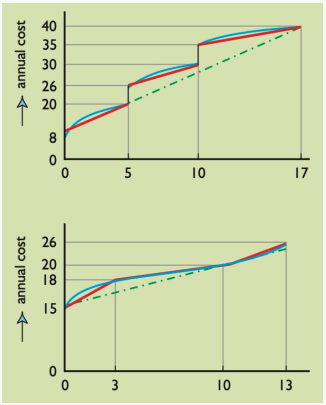
EX : APPROVISIONNEMENT



linéarisation + coût de setup (vert pointillé)

$$\begin{aligned}\min\ & c_a + c_b \\ \text{s.t. } & q_a + q_b = Q \\ & c_a = 8x_a + \frac{40-8}{17} q_a \\ & c_b = 15x_b + \frac{20-15}{10} q_b \\ & x_a, x_b \in \{0, 1\} \\ & 0 \le q_i \le Q_i x_i, \quad \forall i = a, b\end{aligned}$$

EX : APPROVISIONNEMENT



linéarisation par morceaux (rouge)

$$\begin{aligned}\min\ & c_a + c_b \\ \text{s.t. } & q_a + q_b = Q \\ & c_{ij} = F_{i(j-1)} x_{ij} + \frac{F'_{ij} - F_{i(j-1)}}{Q_{ij} - Q_{i(j-1)}} q_{ij}, \quad \forall i, j \\ & c_i = c_{i1} + c_{i2} + c_{i3}, \quad \forall i = a, b \\ & q_i = q_{i1} + q_{i2} + q_{i3}, \quad \forall i = a, b \\ & x_{i1} + x_{i2} + x_{i3} \le 1, \quad \forall i = a, b \\ & x_{ij} \in \{0, 1\}, \quad \forall i = a, b, j = 1, 2, 3 \\ & 0 \le q_{ij} \le (Q_{ij} - Q_{i(j-1)}) x_{ij}, \quad \forall i = a, b\end{aligned}$$

RÉSOLUTION DE PL

- La PL est une forme parmi les plus simples (convexe, différentiable) de l'optimisation numérique sous contraintes et de nombreux algorithmes (exacts ou approchés) s'appliquent
- les solveurs de PL (`gurobi, cplex, glpk, mosek...`) sont des logiciels intégrant des implémentations sophistiquées et efficaces d'algorithmes exacts dédiés tels que les algorithmes du simplexe, ou des points intérieurs
- aucun algorithme à implémenter alors, il suffit de rassembler les données A, b, c et de spécifier le modèle
- entrée PL : format texte (`lp`), langage de modélisation (`gams, ampl`), bibliothèque (`pyomo,matlab`) ou API de solveurs

SOLVEUR GUROBI ET SON API PYTHON

`gurobi + python = gurobipy`

- Gurobi est un solveur commercial, disponible gratuitement pour les étudiants et académiques
- une version de `gurobipy` limitée à de petits modèles est exécutable sur Google Colab
- des exemples de code de Jupyter Notebook sont disponibles, modifiables et exécutables :

`https://www.gurobi.com/jupyter_models/`

EX : DIMENSIONNEMENT DE CANALISATION

$$max \sum_p FLOW_p * diam_p$$
$$\text{s.t.} \sum_p COST_p * diam_p \leq BUDGET$$
$$0 \leq diam_p \leq MAXDIAM_p, \forall p.$$

```
import gurobipy as gp
from gurobipy import GRB
pipe = ['C1','C2']
MAXDIAM = [40, 60], BUDGET = 180
COST = [3, 2], FLOW = [3, 5]
model = gp.Model('Canalisation')
diam = m.addVars(pipe, ub=MAXDIAM, vtype="GRB.CONTINUOUS")
costdiam = LinExpr(COST, diam.values())
ctbudget = m.addConstr(costdiam <= BUDGET)
flowdiam = LinExpr(FLOW, diam.values())
m.setObjective(flowdiam, GRB.MAXIMIZE)
m.optimize()
m.display()
res = {'obj': m.objVal, 'time': m.runtime}
print(res)
xdict = {c: diam[c].x for c in pipe}
print(f"diametres a installer: {xdict}")
```


EX : IMPLÉMENTATION

- le programme linéaire de l'exercice de traitement des eaux usés, en le résolvant plusieurs fois pour différentes valeurs de budget
- le programme linéaire en nombres entiers de l'exercice d'extraction, en générant les données de manière aléatoire comme par exemple ci-dessous :

```
import random
import matplotlib.pyplot as plt
```

Données du problème

```
[7] random.seed(1)

    nplaces = 10                                          # nombre d'emplacements potentiels
    cost = [random.randint(5,25) for _ in range(nplaces)]      # cout d'investissement
    flow = [random.randint(100,300) for _ in range(nplaces)]   # debit moyen
    flowmin = 500
    flowmax = 600
    nmax = 3
```


ALGORITHMES D'OPTIMISATION

simulation : **évalue** la faisabilité et le score d'une solution

heuristique : **parcourt et évalue progressivement** quelques solutions

optimisation : parcourt et évalue **(implicitement) toutes** les solutions

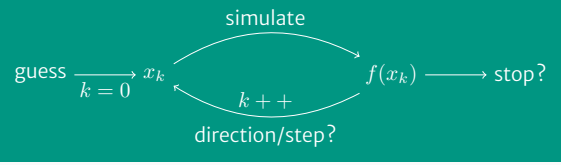
2 principes

- generate & test
- divide & conquer

GENERATE & TEST

méthodes numériques et black-box

1. évalue un candidat, 2. choix du prochain candidat

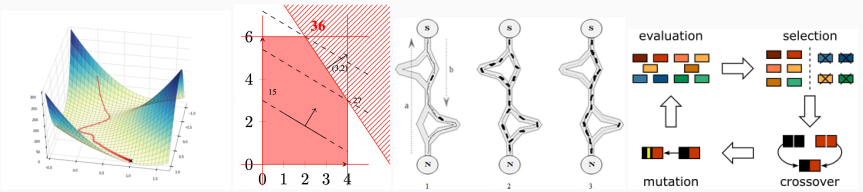


stratégies de choix : aléatoire, énumération, proximité, direction de descente,...

GENERATE & TEST : EXEMPLES

stratégies de choix : aléatoire, énumération, proximité, direction de descente,...

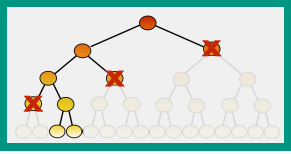
- **méthodes du 1er ordre** (*gradient*, *simplex*) : direction de descente $f'(x) < 0$
- **recherche locale** : choix du meilleur voisin
- **algorithmes particulaires** (*fourmis*) : mémoire collective
- **algorithmes évolutionnaires** (*génétique*) : reproduction d'une population



DIVIDE & CONQUER

principe

- séparer l'espace de recherche
- résoudre une relaxation / estimer borne
- backtracker ou continuer



Pour un problème de minimisation :

- **relaxation :** modèle simplifié, rapide à optimiser, donne une borne inférieure
- une solution réalisable donne une borne supérieure
- certificat d'optimalité si les bornes coincident

DIVIDE & CONQUER : EXEMPLES

- **heuristique gloutonne** : pas de backtrack
- **algorithmes de graphe, programmation dynamique**
- **méthodes arborescentes**
- **branch-and-bound** en optimisation combinatoire