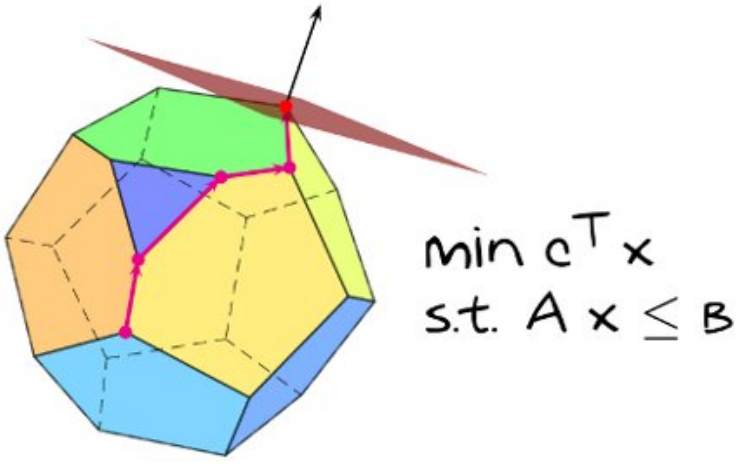
Linear and Discrete Optimization

L'algorithme du simplexe est-il efficace ?

- Algorithmes et leur analyse
- Notation grand O



Algorithmes

Un algorithme est un ensemble fini d'instructions, utilisé typiquement en langage de programmation, comme

- opérations arithmétiques
- comparaisons
- branchements conditionnels
- instructions d'écrire et de lire
- etc.

Le *temps d'exécution* d'un algorithme est le nombre d'instructions qu'il effectue (en fonction de la quantité de données à traiter).

Algorithmes

Un algorithme est un ensemble fini d'instructions, utilisé typiquement en langage de programmation, comme

- opérations arithmétiques
- comparaisons
- branchements conditionnels
- instructions d'écrire et de lire
- etc.

Le *temps d'exécution* d'un algorithme est le nombre d'instructions qu'il effectue (en fonction de la quantité de données à traiter).

Première approximation : calcul du nombre d'*instructions de base* comme

- opérations arithmétiques (addition, soustraction, multiplication, division)
- Comparaisons (c-à-d $<, \leqslant, =$ etc.)

Exemple: Produit scalaire

```
def multiply (a,b):
    s = 0
    for i in range(len(a)):
        s = s + a[i]*b[i]
    return s
```

$[0, \ldots, \text{len}(a)-1]$

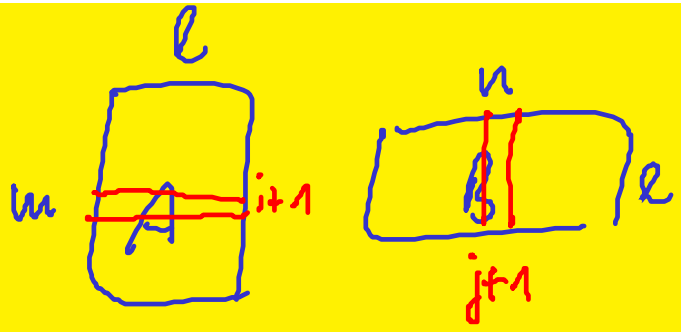
$$a \cdot b = \sum_{i=1}^{n} a_i \cdot b_i$$

- $a, b \in \mathbb{R}^n$
- Nombre de multiplications : n
- Nombre d'additions : n
- Total : $2 \cdot n$ *opérations de base*
- Taille des données est $2 \cdot n$.

Exemple: Produit matriciel

```
def multiply(A,B):
    m = A.rows
    l = A.cols
    n = B.cols
    C = Matrix.zeros(m,n)

    for i in range(m):
        for j in range(n):
            s = 0
            for k in range(l):
                s = s + A[i,k]*B[k,j]
            C[i,j] = s
    return C
```



$2l$ op de base

total : $m \cdot n \cdot 2l$

Exemple: Produit matriciel (cont.)

- $A \in \mathbb{R}^{m \times l}$, $B \in \mathbb{R}^{l \times n}$
- Multiplications : $m \cdot n \cdot l$
- Additions : $m \cdot n \cdot l$
- Taille des données est $m \cdot l + l \cdot n$

Exemple: Produit matriciel (cont.)

- $A \in \mathbb{R}^{m \times l}$, $B \in \mathbb{R}^{l \times n}$
- Multiplications : $m \cdot n \cdot l$
- Additions : $m \cdot n \cdot l$
- Taille des données est $m \cdot l + l \cdot n$

- Pour ces exemples, le calcul exact est possible.
- On s'est intéressé au *taux de croissance* du nombre d'instructions de base.

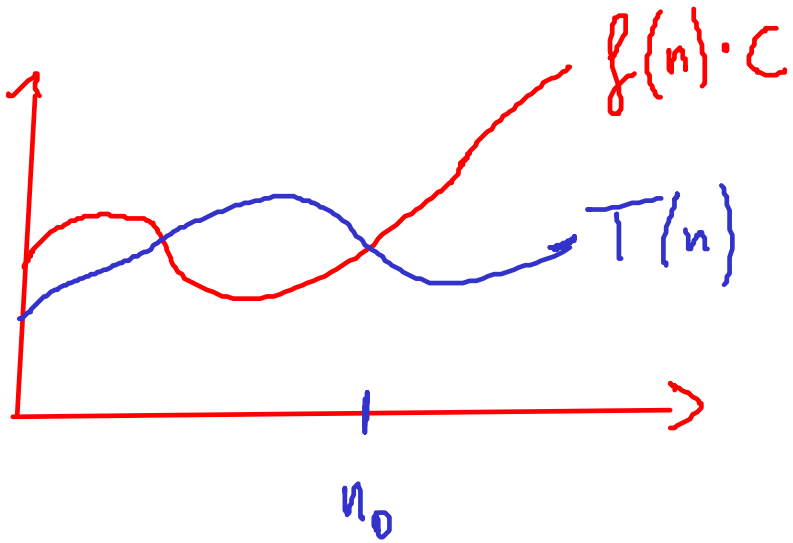
Notation O, Ω, Θ

Soit $T, f : \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ des fonctions

- $T(n) = O(f(n))$, s'il existe des constantes positives $n_o \in \mathbb{N}$ et $c \in \mathbb{R}_{>0}$ telles que

$T(n) \leqslant c \cdot f(n)$ pour tout $n \geqslant n_0$.

$T(n) \in O(f(n))$



Notation O, Ω, Θ

Soit $T, f : \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ des fonctions

- $T(n) = O(f(n))$, s'il existe des constantes positives $n_o \in \mathbb{N}$ et $c \in \mathbb{R}_{>0}$ telles que

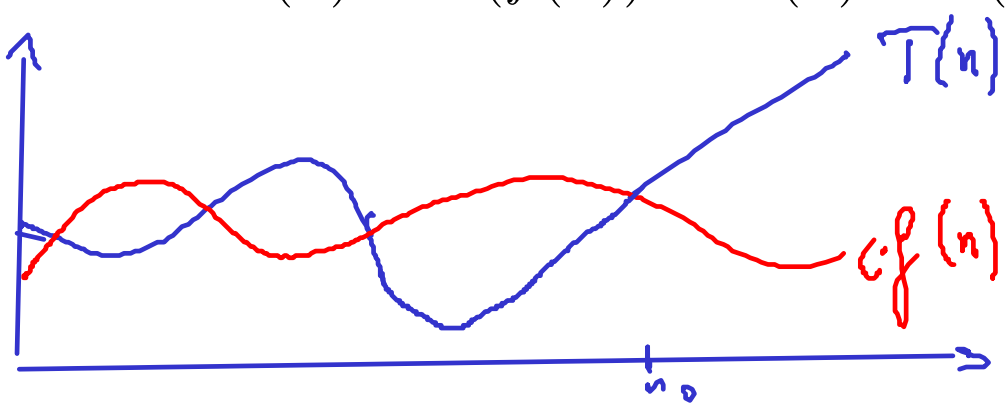
$$T(n) \leqslant c \cdot f(n) \text{ pour tout } n \geqslant n_0.$$

- $T(n) = \Omega(f(n))$, s'il existe des constantes $n_o \in \mathbb{N}$ et $c \in \mathbb{R}_{>0}$ telles que

$$T(n) \geqslant c \cdot f(n) \text{ pour tout } n \geqslant n_0.$$

- $T(n) = \Theta(f(n))$ si

$$T(n) = O(f(n)) \text{ et } T(n) = \Omega(f(n)).$$



Notation O, Ω, Θ

Soit $T, f : \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ des fonctions

- $T(n) = O(f(n))$, s'il existe des constantes positives $n_o \in \mathbb{N}$ et $c \in \mathbb{R}_{>0}$ telles que

$$T(n) \leqslant c \cdot f(n) \text{ pour tout } n \geqslant n_0.$$

- $T(n) = \Omega(f(n))$, s'il existe des constantes $n_o \in \mathbb{N}$ et $c \in \mathbb{R}_{>0}$ telles que

$$T(n) \geqslant c \cdot f(n) \text{ pour tout } n \geqslant n_0.$$

- $T(n) = \Theta(f(n))$ si

$$T(n) = O(f(n)) \text{ et } T(n) = \Omega(f(n)).$$

Example

La fonction $T(n) = 2n^2 + 3n + 1$ est dans $O(n^2)$, car pour tout $n \geqslant 1$ on a $2n^2 + 3n + 1 \leqslant 6n^2$. Dans ce cas, $n_0 = 1$ et $c = 6$.

Notation O, Ω, Θ

Soit $T, f : \mathbb{N} \to \mathbb{R}_{\geqslant 0}$ des fonctions

- $T(n) = O(f(n))$, s'il existe des constantes positives $n_o \in \mathbb{N}$ et $c \in \mathbb{R}_{>0}$ telles que

$$T(n) \leqslant c \cdot f(n) \text{ pour tout } n \geqslant n_0.$$

- $T(n) = \Omega(f(n))$, s'il existe des constantes $n_o \in \mathbb{N}$ et $c \in \mathbb{R}_{>0}$ telles que

$$T(n) \geqslant c \cdot f(n) \text{ pour tout } n \geqslant n_0.$$

- $T(n) = \Theta(f(n))$ si

$$T(n) = O(f(n)) \text{ et } T(n) = \Omega(f(n)).$$

$\forall n \geq 1 : 2n^2 + 3n + 1 \geq n^2$

$n_0 = 1, c = 1$

Example

La fonction $T(n) = 2n^2 + 3n + 1$ est dans $O(n^2)$, car pour tout $n \geqslant 1$ on a $2n^2 + 3n + 1 \leqslant 6n^2$. Dans ce cas, $n_0 = 1$ et $c = 6$.
D'une manière similaire on trouve $T(n) = \Omega(n^2)$, car pour tout $n \geqslant 1$ on a $2n^2 + 3n + 1 \geqslant n^2$. Donc $T(n)$ est dans $\Theta(n^2)$

$\Rightarrow T = \Theta(n^2)$

Quiz

La fonction $f(n) = n^2 \log n$ est

- $= O(n^3)$
- $= O(n)$
- $= \Omega(n)$
- $= \Omega(n^2)$
- $= O(n^{2+\varepsilon})$ pour chaque $\varepsilon > 0$

$$\lim_{n\to\infty} \frac{n^2 \log n}{n^{2+\varepsilon}} = \lim_{n\to\infty} \frac{\log n}{n^{\varepsilon}} = 0 \quad \forall \varepsilon > 0$$

Temps de calcul d'un algorithme

On estime le temps d'exécution d'un algorithme en fonction de la *taille des données*.

Exemple: Le produit scalaire de deux vecteurs de dimension n peut être effectué en temps $O(n)$ *(linéaire)*.

Temps de calcul d'un algorithme

On estime le temps d'exécution d'un algorithme en fonction de la *taille des données*.

Exemple: Le produit scalaire de deux vecteurs de dimension n peut être effectué en temps $O(n)$ *(linéaire)*.

Exemple: Le produit scalaire de deux matrices $n \times n$ peut être effectué en temps $O(n^3)$. La taille des données est $\Theta(n^2)$. Mesuré par la taille des données, l'algorithme s'effectue en temps $O(m^{1.5})$. $(m = n^2)$

Temps de calcul d'un algorithme

On estime le temps d'exécution d'un algorithme en fonction de la *taille des données*.

Exemple: Le produit scalaire de deux vecteurs de dimension n peut être effectué en temps $O(n)$ *(linéaire)*.

Exemple: Le produit scalaire de deux matrices $n \times n$ peut être effectué en temps $O(n^3)$. La taille des données est $\Theta(n^2)$. Mesuré par la taille des données, l'algorithme s'effectue en temps $O(m^{1.5})$. $(m = n^2)$

Un algorithme se déroule en temps polynomial s'il effectue $O(n^k)$ opérations de base pour quelque $k \in \mathbb{N}$, où n est la *taille* des données.

Quiz

```
def listexp(L):
    s = 2
    for i in L:
        s = s**2
    return s
```

$= 5^2$

$n = 4$ $\left(\left(\left(2^2\right)^2\right)^2\right) = 2^{2 \cdot 2 \cdot 2 \cdot 2} = 2^4$

Supposons L contient n éléments.

- L'algorithme effectue $O(n)$ opérations de base.
- L'algorithme effectue $\Omega(2^n)$ opérations de base.
- Après la dernière itération, $s = 2^{n+1}$.
- Après la dernière itération, $s = 2^{2^n}$.

Temps polynomial : Redéfinition

Un algorithme se déroule en *temps polynomial*, s'il effectue $O(n^k)$ opérations de base *sur des rationnels de taille* $O(n^k)$ pour quelque $k \in \mathbb{N}$, où n est la taille des données.

y compris le codage binaire des nombres

- $x \in \mathbb{Z}$: $\text{size}(x) = \lceil \log(|x| + 1) \rceil$
- $x \in \mathbb{Q}$: $\text{size}(x) = \text{size}(p) + \text{size}(q)$, où $x = p/q$ pour $p, q \in \mathbb{Z}$, $q \geqslant 1$ et $\gcd(p, q) = 1$

$13 = 8 + 4 + 1$ $\quad$ $|101$

Temps polynomial : Redéfinition

Un algorithme se déroule en *temps polynomial*, s'il effectue $O(n^k)$ opérations de base *sur des rationnels de taille* $O(n^k)$ pour quelque $k \in \mathbb{N}$, où n est la taille des données.

- $x \in \mathbb{Z}$: $\text{size}(x) = \lceil \log(|x| + 1) \rceil$
- $x \in \mathbb{Q}$: $\text{size}(x) = \text{size}(p) + \text{size}(q)$, où $x = p/q$ pour $p, q \in \mathbb{Z}$, $q \geqslant 1$ et $\gcd(p, q) = 1$

Quiz:

$a, b \in \mathbb{N}$, alors $\text{size}(a \cdot b) =$

- $O(\text{size}(a) + \text{size}(b))$
- $\max\{\text{size}(a), \text{size}(b)\} + 1$

$\log(a \cdot b) = \log a + \log b$

Exemple: L'algorithme d'Euclide

- Pour $a, b \in \mathbb{Z}$, $b \neq 0$ on dit b *divise* a s'il existe un $x \in \mathbb{Z}$ tel que $a = b \cdot x$. On écrit $b \mid a$.

$5 \mid 30$ $\quad$ $30 = 5 \cdot 6$

Exemple: L'algorithme d'Euclide

- Pour $a, b \in \mathbb{Z}$, $b \neq 0$ on dit b *divise* a s'il existe un $x \in \mathbb{Z}$ tel que $a = b \cdot x$. On écrit $b \mid a$.
- Pour $a, b, c \in \mathbb{Z}$, si $c \mid a$ et $c \mid b$, alors c est un *diviseur commun* de a et b.

$3 \mid 60 \quad 3 \mid 42$

Exemple: L'algorithme d'Euclide

- Pour $a, b \in \mathbb{Z}$, $b \neq 0$ on dit b *divise* a s'il existe un $x \in \mathbb{Z}$ tel que $a = b \cdot x$. On écrit $b \mid a$.
- Pour $a, b, c \in \mathbb{Z}$, si $c \mid a$ et $c \mid b$, alors c est un *diviseur commun* de a et b.
- Si au moins un des deux entiers a et b est non-zéro, il existe un *plus grand diviseur commun* de a et b. Il est dénoté par $\gcd(a, b)$.

$\gcd(60, 42) = 6$

Exemple: L'algorithme d'Euclide

- Pour $a, b \in \mathbb{Z}$, $b \neq 0$ on dit b *divise* a s'il existe un $x \in \mathbb{Z}$ tel que $a = b \cdot x$. On écrit $b \mid a$.
- Pour $a, b, c \in \mathbb{Z}$, si $c \mid a$ et $c \mid b$, alors c est un *diviseur commun* de a et b.
- Si au moins un des deux entiers a et b est non-zéro, il existe un *plus grand diviseur commun* de a et b. Il est dénoté par $\gcd(a, b)$.
- Pour $a, b \in \mathbb{Z}$ avec $b > 0$ il existe deux entiers $q, r \in \mathbb{Z}$ uniques tels que

$$a = q \cdot b + r, \text{ et } 0 \leqslant r < b. \qquad \textit{(Division entière)}$$

Exemple: L'algorithme d'Euclide

- Pour $a, b \in \mathbb{Z}$, $b \neq 0$ on dit b *divise* a s'il existe un $x \in \mathbb{Z}$ tel que $a = b \cdot x$. On écrit $b \mid a$.
- Pour $a, b, c \in \mathbb{Z}$, si $c \mid a$ et $c \mid b$, alors c est un *diviseur commun* de a et b.
- Si au moins un des deux entiers a et b est non-zéro, il existe un *plus grand diviseur commun* de a et b. Il est dénoté par $\gcd(a, b)$.
- Pour $a, b \in \mathbb{Z}$ avec $b > 0$ il existe deux entiers $q, r \in \mathbb{Z}$ uniques tels que

$$a = q \cdot b + r, \text{ et } 0 \leqslant r < b. \qquad \textit{(Division entière)}$$

$60 = 1 \cdot 42 + 18$ $\quad q = 1, r = 18$

- Pour $a, b \in \mathbb{Z}$ avec $b > 0$ et $q, r \in \mathbb{Z}$ comme au-dessus, on a $\gcd(a, b) = \gcd(b, r)$.

$d \mid a, d \mid b : a = x_1 d, b = x_2 \cdot d \quad a = q \cdot x_2 \cdot d + r$

$r = (x_1 - q x_2) d$

$d \mid b \quad d \mid r : b = x_1 d \quad r = x_2 d \quad a = q \cdot x_1 \cdot d + x_2 d \quad d \mid a$

Exemple: L'algorithme d'Euclide (cont.)

```
# Condition a>=b>=0, not both equal to 0
def Euclid(a,b):
    if b == 0:
        return a
    else:
        r = a%b
        return Euclid(b,r)
```

$gcd(a,0) = a$

$a = q \cdot b + r$

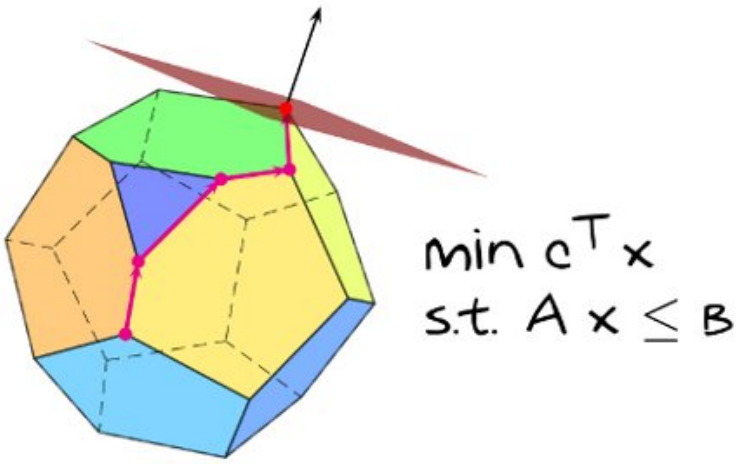
$r > a/2$: $a = qb + r > a$ ↯

- $r \leqslant a/2$
- Le premier paramètre réduit de moitié toute deuxième itération
- Nombre d'itérations: $O(\log a) = O(\text{size}(a))$
- Algorithme en temps linéaire

Linear and Discrete Optimization

L'algorithme du simplexe est-il efficace ?

- Une itération de l'algorithme du simplexe



Une itération de l'algorithme du simplexe

$\lambda_B^T = c^T A_B^{-1}$

`while` B n'est pas optimale

Soit $i \in B$ l'index tel que $\lambda_i < 0$

Calculer $d \in \mathbb{R}^n$ tel que $a_j^T d = 0$, $j \in B \setminus \{i\}$ et $a_i^T d = -1$

Determiner $K = \{k : 1 \leqslant k \leqslant m,\ a_k^T d > 0\}$

$A \cdot d$

`if` $K = \emptyset$

affirmer PL non-borné

`else`

Soit $k \in K$ l'index où $\min_{k \in K} (b_k - a_k^T x^*)/a_k^T d$ est atteint

update $B := B \setminus \{i\} \cup \{k\}$

$b - Ax^*$

Une itération de l'algorithme du simplexe (cont.)

- Supposons $A \in \mathbb{Q}^{m \times n}$, $c \in \mathbb{Q}^n$, $b \in \mathbb{Q}^m$ (données rationnelles)
- Calculer $\lambda_B^T = c^T \cdot A_B^{-1}$ $\longrightarrow O(n^2)$
- Calculer d (le négative d'une colonne de A_B^{-1}) $\longrightarrow O(n)$
- Déterminer K (en calculant $A \cdot d$) $\longrightarrow O(m \cdot n)$
- Déterminer l'index de l'inégalité qui entre dans la base (en calculant $x^* = A_B^{-1} b_B$, $b - Ax^*$ et Ad)
- Si l'on sait A_B^{-1}, la quantité totale est : $O(m \cdot n)$

$O(n^2)$ $O(m \cdot n)$

Questions:

- La taille de A_B^{-1} est-elle polynomiale par rapport à la taille des données (A, b, c) ?
- Combien de temps faut-il pour calculer A_B^{-1} ?

Inversion de matrices

Quiz:

$$A = \begin{pmatrix} p_{11}/q_{11} & \cdots & p_{1n}/q_{1n} \\ & \cdots & \\ p_{n1}/q_{n1} & \cdots & p_{nn}/q_{nn} \end{pmatrix} \in \mathbb{Q}^{n \times n}.$$

La *taille* du produit des dénominateurs $\prod_{i=1,j=1}^{n} q_{ij}$ est

$$size\left(\prod_{i=1,j=1}^{n} q_{ij}\right) = \Theta\left(\sum_{i=1,j=1}^{n} q_{ij}\right)$$

- linéaire par rapport à la taille des données
- non polynomiale par rapport à la taille des données

Inversion de matrices

Quiz:

$$A = \begin{pmatrix} p_{11}/q_{11} & \cdots & p_{1n}/q_{1n} \\ & \cdots & \\ p_{n1}/q_{n1} & \cdots & p_{nn}/q_{nn} \end{pmatrix} \in \mathbb{Q}^{n\times n}.$$

La *taille* du produit des dénominateurs $\prod_{i=1,j=1}^{n} q_{ij}$ est

Ecrire $A = 1/Q \cdot A'$ où Q est le produit des dénominateurs et $A' \in \mathbb{Z}^{n\times n}$

Parce que $A^{-1} = Q \cdot (A')^{-1}$, inverser A' au lieu de A. Sans perte de généralité, on suppose que A est intégrale.

Formule d'inversion de matrices

Pour $A \in \mathbb{R}^{m \times n}$, $1 \leqslant i \leqslant m$ et $1 \leqslant j \leqslant n$, A_{ij} dénoter la matrice obtenue de A en omettant la $i^{\text{ème}}$ ligne et $j^{\text{ème}}$ colonne.

$$A = \begin{pmatrix} 9 & 1 & 7 \\ 3 & 4 & 5 \end{pmatrix} \qquad A_{12} = \begin{pmatrix} 3 & 5 \end{pmatrix}$$

Matrice des cofacteurs :

$$A^{-1} = \frac{1}{\det(A)} \begin{pmatrix} \det(A_{11}) & -\det(A_{21}) & \det(A_{31}) & \dots \\ -\det(A_{12}) & \det(A_{22}) & -\det(A_{32}) & \dots \\ \det(A_{13}) & -\det(A_{23}) & \det(A_{33}) & \dots \\ \vdots & \vdots & \vdots & \dots \\ \vdots & \vdots & \vdots & \dots \end{pmatrix}$$

Inégalité de Hadamard

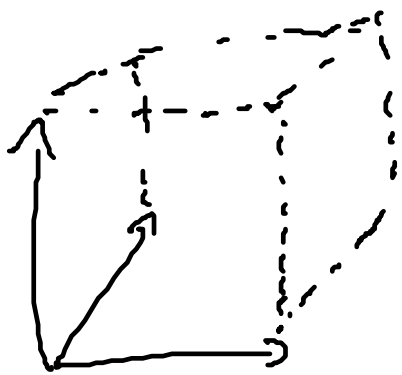
$A = (a_1 \dots a_n)$

Théorème (Hadamard)

$$|\det(A)| \leqslant \prod_{i=1}^{n} \|a_i\|_2 \leqslant n^{n/2} \cdot B^n,$$

$\|a_i\|_2 \leq \left\| \begin{pmatrix} B \\ \vdots \\ B \end{pmatrix} \right\|_2 = \sqrt{n \cdot B^2} = \sqrt{n} \cdot B$

où B est une borne supérieure aux valeurs absolues des entrées de A.



$|\det(A)| \approx \mathrm{vol}(a_1 \dots a_n)$

$\log\left(|\det(A)|\right) \leq \frac{n}{2} \cdot \log n + n \log B$

Si $A \in \mathbb{Z}^{n \times n}$ est intègrale, alors
$\mathrm{size}(\det(A)) = O(n \log n + n \cdot \mathrm{size}(B))$. *Polynomial en* $\mathrm{size}(A)$.

Taille de l'inverse

Corollaire

Soit $A \in \mathbb{Q}^{n \times n}$ une matrice inversible. La taille de A^{-1} est polynomiale par rapport à la taille de A.

Questions:

- La taille de A_B^{-1} est-elle polynomiale par rapport à la taille des données (A, b, c) ?
- Combien de temps faut-il pour calculer A_B^{-1} ?

Taille de l'inverse

Corollaire

Soit $A \in \mathbb{Q}^{n \times n}$ une matrice inversible. La taille de A^{-1} est polynomiale par rapport à la taille de A.

Questions:

- La taille de A_B^{-1} est-elle polynomiale par rapport à la taille des données (A, b, c) ? ✔
- Combien de temps faut-il pour calculer A_B^{-1} ?

Taille de l'inverse

Corollaire

Soit $A \in \mathbb{Q}^{n \times n}$ une matrice inversible. La taille de A^{-1} est polynomiale par rapport à la taille de A.

Questions:

- La taille de A_B^{-1} est-elle polynomiale par rapport à la taille des données (A, b, c) ? ✔
- Combien de temps faut-il pour calculer A_B^{-1} ? ?

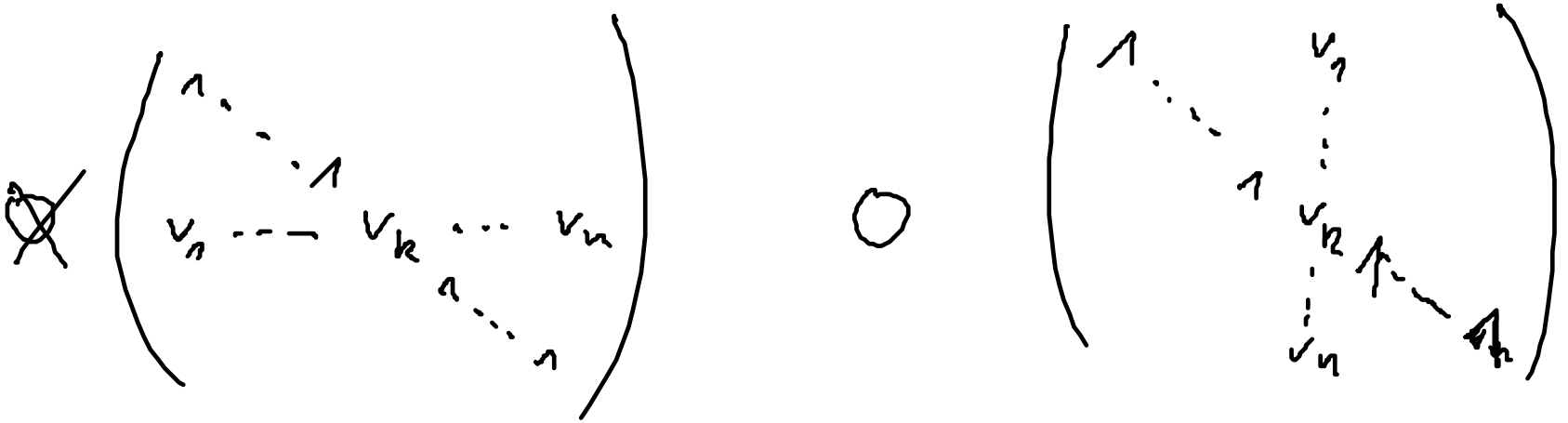
Actualiser A_B^{-1}

Supposons que la base B soit précédée par B' avec

$$\begin{array}{rcl} B' & = & \{b_1, \ldots, b_{k-1}, \quad b'_k \quad , b_{k+1}, \ldots, b_n\} \\ B & = & \{b_1, \ldots, b_{k-1}, \quad b_k \quad , b_{k+1}, \ldots, b_n\} \end{array}$$

Quiz: $A_B \cdot A_{B'}^{-1}$ est sous forme ?

$v^T = a_k^T A_{B'}^{-1}$



Actualiser A_B^{-1} (cont.)

$$\begin{array}{rcl} B' & = & \{b_1, \ldots, b_{k-1}, \quad b'_k \quad , b_{k+1}, \ldots, b_n\} \\ B & = & \{b_1, \ldots, b_{k-1}, \quad b_k \quad , b_{k+1}, \ldots, b_n\} \end{array}$$

- Calculer $a_{b_k}^T \cdot A_{B'}^{-1} = (v_1, \ldots, v_k, \ldots, v_n)$
- A_B^{-1} est obtenue de $A_{B'}^{-1}$ en effectuant les opérations suivantes sur $A_{B'}^{-1}$:
  - Pour chaque colonne $i \neq k$: Soustraire v_i/v_k fois colonne k de colonne i
  - Diviser colonne k par v_k

$O(n)$

$O(n)$ $\rightsquigarrow O(n^2)$

$O(n)$

Quantité totale :

- $O(n^2)$
- $O(n^3)$

Complexité d'une itération

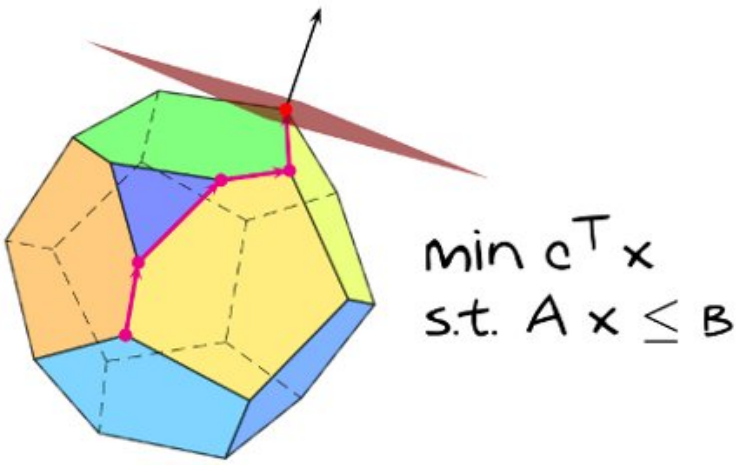
Théorème

Une itération de l'algorithme du simplexe effectue un nombre total de $O(m \cdot n)$ opérations sur des rationnels dont la taille est polynomiale par rapport à la taille des données.

Linear and Discrete Optimization

L'algorithme du simplexe est-il efficace ?

- Le nombre d'itérations



Mauvaises nouvelles et nouvelles partiellement positives

- Le nombre de sommets peut grandir d'une relation exponentielle avec le nombre de variables et de contraintes.
- Pour la plupart des règles de pivotage, une borne exponentielle a été montrée (Klee & Minty 1972).
- Néanmoins, l'algorithme du simplexe est très efficace en pratique et effectue typiquement $O(m)$ étapes.
- Le temps de calcul *en moyenne* de l'algorithme du simplexe (sous un certain modèle probabiliste naturel) est polynomial (Borgwardt 1982).
- Si un programme linéaire quelconque est perturbé (chaque coéfficient par $N(0, \sigma)$), alors l'espérance du nombre d'iterations de l'algorithme du simplexe est polynomial en $1/\sigma$, n et m (Spielman and Teng 2004).

Un problème non résolu célèbre de l'optimisation

Problème non résolu

Existe-il une règle de pivotage pour l'algorithme du simplexe qui assurerait que l'algorithme se termine après un nombre polynomial d'étapes ?

Voir aussi problème 9 de la liste de Smale : *Mathematical Problems for the Next Century* (Smale 1998)

Le mieux connu pour l'instant :

- Kalai (1992, 1997) et Matoušek, Sharir & Welzl (1996) fournissent des règles de pivotage *aléatoires* avec *nombre moyen* de $2^{O(\sqrt{m})}$ itérations.
- Des bornes inférieures pour celles qui égalisent quasiment les bornes supérieures données par Friedmann, Hansen & Zwick (2011).

Programmation linéaire en temps polynomial

Supposons que la taille de tous les coéfficients du programme linéaire $\max\{c^T x : Ax \leqslant b\}$ (composants de A, b et c) est bornée par ϕ.

Méthode de l'ellipsoïde (Khachiyan 1979)

On peut résoudre un programme linéaire $\max\{c^T x : x \in \mathbb{R}^n,\ Ax \leqslant b\}$ avec un nombre polynomial (en $m + n + \phi$) d'opérations de base.

De plus, la taille de tous les nombres apparaissant pendant le déroulement de l'algorithme est polynomiale en $m + n + \phi$.

Question : sans dépendance de ϕ ?

Voir aussi (Grötschel, Lovász & Schrijver 1984)

Programmation linéaire en temps polynomial

Supposons que la taille de tous les coéfficients du programme linéaire $\max\{c^T x : Ax \leqslant b\}$ (composants de A, b et c) est bornée par ϕ.

Méthode de l'ellipsoïde (Khachiyan 1979)

On peut résoudre un programme linéaire $\max\{c^T x : x \in \mathbb{R}^n,\ Ax \leqslant b\}$ avec un nombre polynomial (en $m + n + \phi$) d'opérations de base.

De plus, la taille de tous les nombres apparaissant pendant le déroulement de l'algorithme est polynomiale en $m + n + \phi$.

Voir aussi (Grötschel, Lovász & Schrijver 1984)

Programmation linéaire en temps polynomial

Supposons que la taille de tous les coéfficients du programme linéaire $\max\{c^T x : Ax \leqslant b\}$ (composants de A, b et c) est bornée par ϕ.

Méthode de l'ellipsoïde (Khachiyan 1979)

On peut résoudre un programme linéaire $\max\{c^T x : x \in \mathbb{R}^n,\ Ax \leqslant b\}$ avec un nombre polynomial (en $m + n + \phi$) d'opérations de base.

De plus, la taille de tous les nombres apparaissant pendant le déroulement de l'algorithme est polynomiale en $m + n + \phi$.

Voir aussi (Grötschel, Lovász & Schrijver 1984)

Programmation linéaire en temps polynomial

Supposons que la taille de tous les coéfficients du programme linéaire $\max\{c^T x : Ax \leqslant b\}$ (composants de A, b et c) est bornée par ϕ.

Méthode de l'ellipsoïde (Khachiyan 1979)

On peut résoudre un programme linéaire $\max\{c^T x : x \in \mathbb{R}^n,\ Ax \leqslant b\}$ avec un nombre polynomial (en $m + n + \phi$) d'opérations de base.

De plus, la taille de tous les nombres apparaissant pendant le déroulement de l'algorithme est polynomiale en $m + n + \phi$.

Voir aussi (Grötschel, Lovász & Schrijver 1984)

Programmation linéaire en temps polynomial

Supposons que la taille de tous les coéfficients du programme linéaire $\max\{c^T x : Ax \leqslant b\}$ (composants de A, b et c) est bornée par ϕ.

Méthode de l'ellipsoïde (Khachiyan 1979)

On peut résoudre un programme linéaire $\max\{c^T x : x \in \mathbb{R}^n,\ Ax \leqslant b\}$ avec un nombre polynomial (en $m + n + \phi$) d'opérations de base.

De plus, la taille de tous les nombres apparaissant pendant le déroulement de l'algorithme est polynomiale en $m + n + \phi$.

Voir aussi (Grötschel, Lovász & Schrijver 1984)

Programmation linéaire en temps polynomial

Supposons que la taille de tous les coéfficients du programme linéaire $\max\{c^T x : Ax \leqslant b\}$ (composants de A, b et c) est bornée par ϕ.

Méthode de l'ellipsoïde (Khachiyan 1979)

On peut résoudre un programme linéaire $\max\{c^T x : x \in \mathbb{R}^n,\ Ax \leqslant b\}$ avec un nombre polynomial (en $m + n + \phi$) d'opérations de base.

De plus, la taille de tous les nombres apparaissant pendant le déroulement de l'algorithme est polynomiale en $m + n + \phi$.

Voir aussi (Grötschel, Lovász & Schrijver 1984)