

# Introduction au Calcul Scientifique

LI217

UPMC (Paris 6)  
France

2012-2013

# Équipe pédagogique et Évaluation

## Cours:

- ▶ Jean-Claude BAJARD

## Travaux Dirigés et Travaux sur Machines:

- ▶ **Lundi:** Jérémy Berthomieu & Jules Svartz
- ▶ **Mardi:** Stef Graillat – Olga Kupriianova
- ▶ **Mercredi:** Valérie Ménissier-Morain – Pierre Fortin
- ▶ **Groupe PIMA:** Christoph Lauter – Olga Kupriianova

## Évaluation:

- ▶ Examen 60%
- ▶ CC 30% Partiel (semaine du 1er avril) + 10% TME

## Calculs sur Ordinateur

- ▶ Notre but, trouver un compromis entre :
  - ▶ la vitesse (le temps qu'il faut, tout de suite)
  - ▶ la précision (exacte, approchée)
  - ▶ le coût (mémoire, nombre de processeurs)
- ▶ le coeur :
  - ▶ la représentation des données (systèmes de numération, polynômes, matrices)
  - ▶ l'algorithmique (construction de méthodes de calcul)
- ▶ Les approches :
  - ▶ théoriques (recherche de bornes de complexité)
  - ▶ logicielles (bibliothèques, outils)
  - ▶ matérielles (Arithmetic-Logic Unit ALU, systèmes embarqués)

## Calculs sur Ordinateur : Où ?

- ▶ **Calcul scientifique :**
  - ▶ systèmes embarqués, robotique, aéronautique → sciences de l'ingénieur...
  - ▶ calcul intensif, masse de données, recherche exhaustive, météorologie,...
- ▶ **Cryptologie :**
  - ▶ algorithmes de chiffrement...
  - ▶ Cryptanalyse...
- ▶ **Imagerie :**
  - ▶ Traitement du signal, imagerie médicale..
  - ▶ Géométrie numérique, CAO, Géométrie algorithmique, triangulation...
- ▶ **l'intelligence artificielle**
  - ▶ Google (pagerank, valeurs propres,...)
  - ▶ fouille de données, ...

## Calcul Scientifique : Pourquoi ?



- ▶ Le vol 501 d'Ariane 5, le 4 juin 1996, explosion après 40s de vol, erreur de conversion numérique.
- ▶ Le 25 février 1991, un antimissile Patriot américain a raté sa cible, le temps était évalué en dixième de seconde...
- ▶ 1994, bug du pentium dû à une mauvaise implantation de la division.

# Programme

Les entiers et l'Ordinateur

Division euclidienne - congruences - algorithme d'Euclide

Représentation des flottants - algorithmes élémentaires

Représentation des matrices et arithmétique - algorithmes et complexité

Algorithme de Gauss sur les flottants, sur les rationnels

Algorithmes de calcul de déterminants et produits vectoriels

Décomposition LU - techniques de programmation

Algorithmes pour l'inversion de matrice - sensibilisation aux erreurs d'arrondis

Evaluation/Interpolation

Résolution d'équations

# Les entiers et l'Ordinateur

## Axiomes de Peano

Les Mathématiciens définissent récursivement les entiers naturels ( $\mathbb{N}$ ) avec la notion de successeur.

1. 0 est un entier naturel.
2. Tout entier naturel  $n$  possède un successeur, noté  $\sigma(n)$ .
3. Pour tout  $n \in \mathbb{N}$ ,  $\sigma(n) \neq 0$ .<sup>1</sup>
4. Si  $n \neq t$  alors  $\sigma(n) \neq \sigma(t)$ .  
*Si  $\sigma(n) = \sigma(t)$  alors  $n = t$ .*
5. Si  $P \subset \mathbb{N}$ ,  $0 \in P$  et pour tout  $n \in P$ ,  $\sigma(n) \in P$  alors  $P = \mathbb{N}$ .<sup>2</sup>  
*Si une propriété est vérifiée par 0 et si, pour tout entier naturel  $n$  qui la vérifie,  $\sigma(n)$  la vérifie également, alors la propriété est vraie sur  $\mathbb{N}$ .*

---

<sup>1</sup>cet item et le suivant permettent de définir la notion de prédécesseur, noté  $\rho(n)$  avec  $\sigma(\rho(n)) = \rho(\sigma(n)) = n$

<sup>2</sup>preuve par récurrence



## Définition récursive des opérations $+$ et $\times$

Cette axiomatique permet de définir récursivement les opérations (algorithmes):

- ▶ L'addition, 
$$\begin{cases} a + 0 & = & a \\ a + \sigma(b) & = & \sigma(a + b) \end{cases}$$
- ▶ La multiplication, 
$$\begin{cases} a \times 0 & = & 0 \\ a \times \sigma(b) & = & (a \times b) + a \end{cases}$$

## Les entiers relatifs

Loi de groupe pour  $+$ , et d'anneau avec  $\times$

- ▶ On définit la relation d'équivalence  $\mathcal{R}$  sur  $\mathbb{N} \times \mathbb{N}$  :  
 $(x, y)\mathcal{R}(x', y')$  équivaut à  $x + y' = x' + y$ .<sup>3</sup>
- ▶ Les classes d'équivalence de  $\mathbb{N} \times \mathbb{N}$  suivant  $\mathcal{R}$  forment un ensemble noté  $\mathbb{Z}$  **des entiers relatifs**.<sup>4</sup>
- ▶ On définit l'addition :  $(x, y) + (x', y') = (x + x', y + y')$  sur  $\mathbb{Z}$
- ▶ et la multiplication :  
 $(x, y) \times (x', y') = (x \times x' + y \times y', x' \times y + x \times y')$
- ▶ On note  $x$  pour la classe  $(x, 0)$  et  $-x$  pour celle de  $(0, x)$

---

<sup>3</sup>En fait  $(x, y)$  est équivalent à  $(x - y, 0)$  si  $x \geq y$  et à  $(0, y - x)$  sinon

<sup>4</sup> $(x, x)$  est de la classe de  $(0, 0)$ , que l'on note 0

## Loi de groupe pour $+$ , et d'anneau avec $\times$

### Les entiers relatifs

- ▶  $\mathbb{Z}$  muni de l'addition est un **groupe**
  - ▶ 0 est l'élément **neutre**
  - ▶ tout  $x \in \mathbb{Z}$  admet un **opposé** noté  $-x$  :  
 $x + (-x) = (-x) + x = 0$
  - ▶ **associativité** :  $x + (y + z) = (x + y) + z$
- ▶ l'addition sur  $\mathbb{Z}$  est **commutative** :  $x + y = y + x$
- ▶  $(\mathbb{Z}, +, \times)$  est un **anneau**
  - ▶  $(\mathbb{Z}, +)$  est un **groupe commutatif**
  - ▶ **associativité** :  $x \times (y \times z) = (x \times y) \times z$
  - ▶ **distributivité** :  $x \times (y + z) = (x \times y) + (x \times z)$
- ▶ 1 est l'élément neutre de  $\times$ , l'anneau  $(\mathbb{Z}, +, \times)$  est **unitaire**
- ▶ commutativité :  $x \times y = y \times x$ , l'anneau  $(\mathbb{Z}, +, \times)$  est **commutatif**

## Autres opérations et division euclidienne

- ▶ **Soustraction** : addition de l'opposé,  $a - b = a + (-b)$
- ▶ **Ordre total**  $a \leq b$  si et seulement si  $(b - a) \in \mathbb{N}$
- ▶ **Valeur absolue** :  $|a| = a$  si  $a \in \mathbb{N}$ ,  $|a| = -a$  sinon
- ▶  $\mathbb{N}$  et  $\mathbb{Z}$  sont archimédiens : pour tout couple d'entiers non nuls  $a$  et  $b$ , il existe un entier  $c$  tel que  $b \leq a \times c$ .
- ▶ **Division euclidienne** : pour tout couple d'entiers  $a$  et  $b$ , il existe  $q$  et  $r$  tels que :  $b = a \times q + r$  avec  $0 \leq r < |a|$ .
- ▶ On note :  $q = b/a$  ( ou  $b \div a$  ) et  $r = b \% a$  ( ou  $b \bmod a$  )

## L'écriture des nombres

- ▶ Système de représentation dit de **position** (Gerbert d'Aurillac (940-1003), pape Sylvestre II)
- ▶ Choix d'une base  $\beta$   
et de **chiffres** (alphabet)  $\mathcal{C} = \{0, 1, 2, \dots, \beta - 1\}$
- ▶ Le nombre  $n$  s'écrit :  $n_{k-1} \dots n_3 n_2 n_1 n_0 = \sum_{i=0}^{k-1} n_i \beta^i$  avec pour  
tout  $i$ ,  $n_i \in \mathcal{C}$
- ▶ Correspondance entre **valeurs** et **symboles**
- ▶ Un système pratique pour l'ordre de grandeur et les opérations

## Exemple de base

- ▶ **Base dix** et  $\mathcal{C} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$  :  
 $9281 = 9 \times 10^3 + 2 \times 10^2 + 8 \times 10 + 1$
- ▶ **Base deux** et  $\mathcal{C} = \{0, 1\}$  :  $10010001 = 1 \times 2^7 + 1 \times 2^4 + 1$
- ▶ **Base Hexadécimale (16)** et  
 $\mathcal{C} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$  :  
 $D3C2 = 13 \times 16^3 + 3 \times 16^2 + 12 \times 16 + 2$

## Algorithme d'écriture dans la base

Un algorithme glouton représentation de l'entier  $n$  en base  $\beta$ :

$$B_0 \leftarrow 1$$

$$i \leftarrow 0$$

**tant que**  $B_i < n$  **faire**

$$B_{i+1} \leftarrow B_i \times \beta$$

$$i \leftarrow i + 1$$

**tant que**  $i \geq 0$

$$c \leftarrow 0$$

**tant que**  $c \times B_i \leq n$  **faire**  $c \leftarrow c + 1$

$$n_i \leftarrow c - 1$$

$$n \leftarrow n - n_i \times B_i$$

$$i \leftarrow i - 1$$

## Algorithme d'écriture dans la base

Un algorithme utilisant la division euclidienne,  
représentation de l'entier  $n$  en base  $\beta$ :

$i \leftarrow 0$

**tant que**  $n > 0$  **faire**

$n_i \leftarrow n \% \beta$

$n \leftarrow n / \beta$

$i \leftarrow i + 1$



## Algorithme d'addition

- ▶ **L'algorithme dépend du système de représentation.** Dans un système de position classique, l'addition s'effectue en partant des chiffres de poids faibles, en propageant une retenue qui vaut 0 ou 1.
- ▶ **Opérateur de base** l'addition de deux chiffres  $a_i$  et  $b_i$  (notion de table) et la retenue  $c_i$ , il génère un chiffre de la somme  $s_i$  et la retenue sortante  $c_{i+1}$  :  
$$a_i + b_i + c_i = c_{i+1} \times \beta + s_i \leq \beta + (\beta - 2) + c_i$$
- ▶ Nous remarquons que si  $0 \leq c_i \leq 1$  alors  $0 \leq c_{i+1} \leq 1$

## Algorithme d'addition

**Entrées** base  $\beta$ ,  $A = a_{k-1} \dots a_1 a_0$  et  $B = b_{k-1} \dots b_1 b_0$

**Sorties**  $S = s_k s_{k-1} \dots s_1 s_0$

**Corps**  $c \leftarrow 0$

**Pour**  $i = 0$  à  $k - 1$  **faire**

$s_i \leftarrow a_i + b_i + c$  (lecture dans une table)

**Si**  $s_i \geq \beta$  **alors**

$c \leftarrow 1$

$s_i \leftarrow s_i - \beta$

**Sinon**  $c \leftarrow 0$

$s_k \leftarrow c$

## La base deux

Pas besoin de table pour l'addition

**Entrées** base 2,  $A = a_{k-1} \dots a_1 a_0$  et  $B = b_{k-1} \dots b_1 b_0$

**Sorties**  $S = s_k s_{k-1} \dots s_1 s_0$

**Corps**  $c \leftarrow 0$

**Pour**  $i = 0$  à  $k - 1$  faire

$s_i \leftarrow a_i \oplus b_i \oplus c$  ( $\oplus$  ou exclusif)

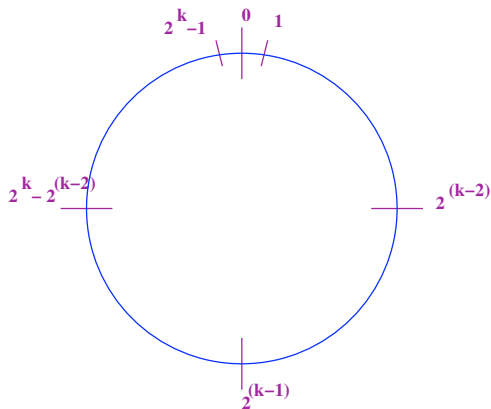
$c \leftarrow a_i \cdot b_i \oplus a_i \cdot c \oplus b_i \cdot c$  (. et)

$s_k \leftarrow c$

## Les entiers de l'ordinateur

- ▶ Au sein de la machine, les nombres sont représentés en **base 2**, un chiffre est appelé "**bit**" (binary digit).
- ▶ Un entier est codé sur un **nombre fini  $k$  de bits** ( $k = 32$  ou  $64$ )
- ▶ De ce fait, seuls les nombres strictement inférieurs à  $2^k$  sont représentables.
- ▶ L'arithmétique est donc **modulaire**, modulo  $2^k$ . (restes de la division euclidienne par  $2^k$ ).
- ▶ En C, ces entiers sont nommés `unsigned int`.

## Les entiers de l'ordinateur



## Premiers effets de bord

$$10^8 \rightarrow 100000000$$

$$10^9 \rightarrow 1000000000$$

$$10^{10} \rightarrow 1410065408$$

$$10^{11} \rightarrow 1215752192$$

$$10^{12} \rightarrow 3567587328$$

$$2^{30} \rightarrow 1073741824$$

$$2^{31} \rightarrow 2147483648$$

$$2^{32} \rightarrow 0$$

$$2^{33} \rightarrow 0$$

## Algorithme d'addition avec contrôle

**Entrées** base 2,  $A = a_{k-1} \dots a_1 a_0$  et  $B = b_{k-1} \dots b_1 b_0$

**Sorties**  $S = s_{k-1} \dots s_1 s_0$  et  $OF$  (overflow)

**Corps**  $c \leftarrow 0$

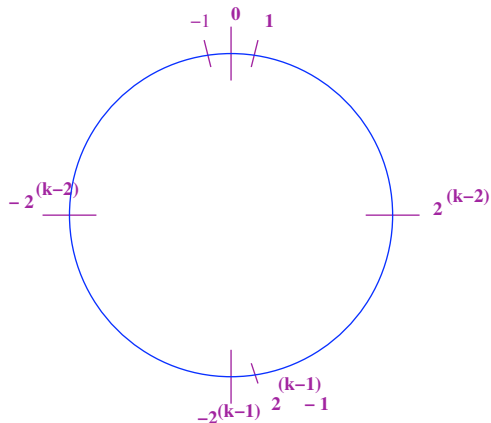
**Pour**  $i = 0$  à  $k - 1$  **faire**

$s_i \leftarrow a_i \oplus b_i \oplus c$  ( $\oplus$  ou exclusif)

$c \leftarrow a_i \cdot b_i \oplus a_i \cdot c \oplus b_i \cdot c$  (. et)

$OF \leftarrow c$

## Les entiers signés de l'ordinateur





## Les entiers signés de l'ordinateur

écritures binaire	valeurs absolues	valeurs signées
0000000000000000	0	0
0000000000000001	1	1
0000000000000010	2	2
0000000000000011	3	3
...	...	...
0111111111111110	$2^{15} - 2$	$2^{15} - 2$
0111111111111111	$2^{15} - 1$	$2^{15} - 1$
1000000000000000	$2^{15} = 2^{16} - 2^{15}$	$-2^{15}$
1000000000000001	$2^{15} + 1$	$-(2^{15} - 1)$
...	...	...
1111111111111110	$2^{16} - 2$	$-2$
1111111111111111	$2^{16} - 1$	$-1$

## Second effets de bord

$$10^8 \rightarrow 100000000$$

$$10^9 \rightarrow 1000000000$$

$$10^{10} \rightarrow 1410065408$$

$$10^{11} \rightarrow 1215752192$$

$$10^{12} \rightarrow -727379968$$

$$2^{30} \rightarrow 1073741824$$

$$2^{31} \rightarrow -2147483648$$

$$2^{32} \rightarrow 0$$

$$2^{33} \rightarrow 0$$

## Addition et soustraction en complément à la base

- ▶ Sur  $k$  bits, un entier signé  $x$  est représenté par  $X$ , de la façon suivante:

Si  $0 \leq x \leq 2^{k-1} - 1$  alors représentation  $X = x$  en binaire

Si  $-2^{k-1} \leq x < 0$  alors représentation  $X = 2^k + x$  en binaire.

- ▶ **Construction de l'opposé** : Soit  $X$  la représentation de  $x$  et  $Y$  celle de  $-x$  alors  $Y = 2^k - X$  (et  $X = 2^k - Y$ )

On peut encore écrire :  $Y = (2^k - 1) - X + 1$  où

$(2^k - 1) - X$  est le complément à 1 de  $X$

- ▶  $opp(0000110101111000) = 1111001010000111 + 1 = 1111001010001000$

## Addition et soustraction en complément à la base

$$\begin{array}{r} 00001100 \\ + 01001001 \\ \hline 01010101 \end{array}$$

$$\begin{array}{r} 10001100 \\ + 01001001 \\ \hline 11010101 \end{array}$$

$$\begin{array}{r} 01001100 \\ + 01001001 \\ \hline 10010101 \end{array}$$

$$\begin{array}{r} 10001100 \\ + 11001001 \\ \hline 1 \ 01010101 \end{array}$$

$$\begin{array}{r} 10001100 \\ + 11110100 \\ \hline 1 \ 10000000 \end{array}$$

## Les entiers en mémoire et sur disque

- ▶ Les nombres suivant les architectures sont stockés  
octets de poids forts en tête Big Endian  
ou octet de poids faibles en tête Little Endian (ce qui est les cas des intels)
- ▶ Le nombre (ici en hexadécimal)  $X = 0A\ 12\ B1\ 2F$  est stocké sous la forme  $2F\ B1\ 12\ 0A$

## La multiplication stupide

Entrées  $A$  et  $B$

Sorties  $P = A \times B$

Corps  $P \leftarrow 0$

**Pour**  $i = 1$  à  $B$  faire

$P \leftarrow P + A$

La complexité de cet algorithme est dite exponentielle en la taille des opérandes concernant le nombre d'additions (*si  $k$  chiffres binaires, le nombre d'itérations est de l'ordre de  $k \times 2^k$* )

## La multiplication avec les tables de multiplication

**Entrées** base  $\beta$ ,  $A = a_{k-1} \dots a_1 a_0$  et  $B = b_{k-1} \dots b_1 b_0$

**Sorties**  $P = p_{2k-1} \dots p_1 p_0 = A \times B$

**Corps**  $P \leftarrow 0$

**Pour**  $i = 0$  à  $k - 1$  **faire**

$T \leftarrow 0$

**Pour**  $j = 0$  à  $k - 1$  **faire**

$T \leftarrow T + a_j \times b_i \times \beta^{i+j}$

$P \leftarrow P + T$

Le produit par  $\beta^t$  est un décalage de  $t$  positions vers la gauche  
(ajout de  $t$  zéros sur les poids faibles)

Le produit de deux chiffres  $a_j \times b_i$  est lu dans un table.

## La multiplication avec les tables de multiplication

- ▶ Dans la boucle **Pour  $j = 0$  à  $k - 1$  faire**,  
on a  $T \leq (\beta - 1)\beta^{i+j}$  et  
 $a_j \times b_i \times \beta^{i+j} \leq (\beta - 2) \times \beta^{i+j+1} + \beta^{i+j}$ , donc en sortie de  
boucle,  $T \leq (\beta - 1)\beta^{i+j+1}$   
Ainsi  $T \leftarrow T + a_j \times b_i \times \beta^{i+j}$  se réduit à **au plus deux  
additions de deux chiffres**.
- ▶ L'addition  $P \leftarrow P + T$  est au plus linéaire.
- ▶ Ici, l'algorithme est dit **quadratique** en la taille des opérandes  
en nombre d'additions de deux chiffres (*de l'ordre du carré du  
nombre de chiffres*)



## La multiplication en base deux

**Entrées** base 2,  $A = a_{k-1} \dots a_1 a_0$  et  $B = b_{k-1} \dots b_1 b_0$

**Sorties**  $P = p_{2k-1} \dots p_1 p_0 = A \times B$

**Corps**  $P \leftarrow 0$

$D \leftarrow A$

**Pour**  $i = 0$  à  $k - 1$  **faire**

**Si**  $b_i = 1$  **alors**  $P \leftarrow P + D$

$D \leftarrow D + D$

## La multiplication diviser pour régner

Nous supposons que  $k$  le nombre de chiffres est une puissance de deux :  $k = 2^t$ .

- ▶ Les opérandes peuvent être coupées en deux :

$$A = A_1\beta^{k/2} + A_0 \text{ et } B = B_1\beta^{k/2} + B_0$$

- ▶ Le produit peut s'écrire

$$A \times B = (A_1 \times B_1)\beta^k + (A_1 \times B_0 + A_0 \times B_1)\beta^{k/2} + A_0 \times B_0$$

- ▶ Mais,

$$(A_1 \times B_0 + A_0 \times B_1) = (A_1 - A_0) \times (B_0 - B_1) + (A_1 \times B_1) + A_0 \times B_0$$

- ▶ Ainsi le produit de deux nombres de  $k$  bits se réduit à trois produits de nombres de  $k/2$  bits.

## La multiplication diviser pour régner

### Karatsuba( $A, B, k$ )

**Entrées** base  $\beta$ ,  $A = a_{k-1} \dots a_1 a_0$  et  $B = b_{k-1} \dots b_1 b_0$

**Sorties**  $P = p_{2k-1} \dots p_1 p_0 = A \times B$

**Corps** Si  $k = 1$  alors  $P = a_0 \times b_0$  (classique)

Sinon

$t1 \leftarrow \lfloor k/2 \rfloor$  et  $t2 \leftarrow k - t1$

$A_1 \leftarrow a_{k-1} \dots a_{t1}$  et  $A_0 \leftarrow a_{t1-1} \dots a_0$  et  $Sa \leftarrow 1$

$B_1 \leftarrow b_{k-1} \dots b_{t1}$  et  $B_0 \leftarrow b_{t1-1} \dots b_0$  et  $Sb \leftarrow 1$

Si  $A_0 \geq A_1$  alors  $D \leftarrow A_0 - A_1$  sinon  $D \leftarrow A_1 - A_0$ ,  $Sa \leftarrow -1$

Si  $B_0 \geq B_1$  alors  $E \leftarrow B_0 - B_1$ ,  $Sb \leftarrow -1$  sinon  $E \leftarrow B_1 - B_0$ ,

$T \leftarrow \text{Karatsuba}(A_1, B_1, t2)$

$U \leftarrow \text{Karatsuba}(A_0, B_0, t1)$

$V \leftarrow \text{Karatsuba}(D, E, t2)$

$V \leftarrow (Sa * Sb) * V + T + U$

$P \leftarrow T \times \beta^k + V \times \beta^{k/2} + U$

## La multiplication diviser pour régner

### Complexité

- ▶ Supposons que  $k = 2^n$  et que  $T(k)$  est le temps de calcul
- ▶  $T(k) = 3 \times T(k/2) + \alpha k$  ( $\alpha k$  complexité des additions)  
 $3T(k/2) = 3^2 \times T(k/4) + 3\alpha k/2$   
 $3^2 T(k/4) = 3^3 \times T(k/8) + 3^2 \alpha k/4$   
 $3^{n-1} T(k/2^{n-1}) = 3^n \times T(k/2^n) + 3^{n-1} \alpha k/2^{n-1}$
- ▶  $T(k) = 3^n T(1) + \alpha k \frac{(3/2)^n - 1}{3/2 - 1}$   
 $T(k) = 3^n T(1) + 2\alpha(3^n - k)$  car  $k = 2^n$   
 $T(k) = k^{\log_2(3)} T(1) + 2\alpha \times k^{\log_2(3)} - 2\alpha \times k$  car  $3^n = k^{\log_2(3)}$
- ▶ Complexité en  $O(k^{\log_2(3)})$

# Division euclidienne - congruences - algorithme d'Euclide

## Division Euclidienne

$$a = d \times q + r \text{ avec } 0 \leq r < |d|$$

$$\begin{array}{r}
 24571 \quad | \quad \underline{53} \\
 4 \times 5300 \quad | \quad 463 \\
 \underline{-21200} \\
 3371 \\
 6 \times 530 \quad | \\
 \underline{-3180} \\
 191 \\
 3 \times 53 \quad | \\
 \underline{-159} \\
 32
 \end{array}$$

Alignement

diviseur  $\times$  puissance de la base  
(donne la position du chiffre du quotient)

choix d'un chiffre du quotient

soustraction du produit

chiffre quotient  $\times$  diviseur

passage au chiffre suivant

$$24571 = 463 \times 53 + 32$$

# Evaluation d'une valeur approchée du quotient

D. Knuth, The Art of Computer Programming

- ▶  $a = d \times q + r$  avec  $0 \leq r < |d|$
- ▶ Nous nous plaçons dans le cas où  $0 \leq q < \beta$ , et  $\beta$  est la base.  
*Autrement dit, le quotient se réduit à un chiffre.*
- ▶  $a = a_n a_{n-1} a_{n-2} \dots a_1 a_0$  et  $d = d_{n-1} d_{n-2} \dots d_1 d_0$
- ▶ Nous estimons  $q$  à partir de  $\tilde{q} = \min \left( \left\lfloor \frac{a_n \beta + a_{n-1}}{d_{n-1}} \right\rfloor, \beta - 1 \right)$ .
- ▶ **Remarque 1 :** Nous avons  $q \leq \tilde{q}$
- ▶ **Remarque 2 :** Si  $\lfloor \beta/2 \rfloor \leq d_{n-1}$  nous avons  $\tilde{q} - 2 \leq q \leq \tilde{q}$

# Un algorithme de Division Euclidienne

D. Knuth, The Art of Computer Programming

## ChifQuot

Entrées :  $a = a_n a_{n-1} \dots a_1 a_0$  et  $d = d_{n-1} d_{n-2} \dots d_1 d_0$   
avec  $\lfloor \beta/2 \rfloor \leq d_{n-1}$  et  $a < \beta \times d$ .

Sorties :  $q$  et  $a_n a_{n-1} \dots a_1 a_0$  tels que :  
 $0 \leq a_n a_{n-1} \dots a_1 a_0 < d$

Corps :  $q \leftarrow \min \left( \left\lfloor \frac{a_n \beta + a_{n-1}}{d_{n-1}} \right\rfloor, \beta - 1 \right)$   
 $(a_n \dots a_0) \leftarrow (a_n \dots a_0) - q \times (d_{n-1} \dots d_0)$

**Tant que**  $(a_n \dots a_0) < 0$  **faire**

$$q \leftarrow q - 1$$

$$(a_n \dots a_0) \leftarrow (a_n \dots a_0) + (d_{n-1} \dots d_0)$$



# Un algorithme de Division Euclidienne

D. Knuth, The Art of Computer Programming

**Entrées :**  $a = a_{n+m}a_{n+m-1} \dots a_1a_0$  et  $d = d_{n-1}d_{n-2} \dots d_1d_0$   
avec  $a_{n+m} = 0$

**Sorties :**  $q_m \dots q_0$  et  $r_{n-1} \dots r_0$   
avec  $0 \leq r_{n-1} \dots r_0 < d_{n-1} \dots d_0$

**Normalisation**  $c \leftarrow 0$

**Tant que**  $d_{n-1} < \lfloor \beta/2 \rfloor$  **faire**

$a \leftarrow a + a$ ,  $d \leftarrow d + d$  et  $c \leftarrow c + 1$

**Boucle Pour**  $j = m$  à  $0$  **faire**

$(q_j, a_{n+j} \dots a_j) \leftarrow \text{ChifQuot}(a_{n+j} \dots a_j, d_{n-1} \dots d_0)$

**Dénormalisation**  $r_{n-1} \dots r_0 \leftarrow a_{n-1} \dots a_0 \div 2^c$

## Un algorithme de Division Euclidienne

Un petit exemple 24571/53 et 21571/59

24571 par 53

245/53

$$\tilde{q} = \min \left( \left\lfloor \frac{24}{5} \right\rfloor, 9 \right) = 4$$

---

337/53

$$\tilde{q} = \min \left( \left\lfloor \frac{33}{5} \right\rfloor, 9 \right) = 6$$

---

191/53

$$\tilde{q} = \min \left( \left\lfloor \frac{19}{5} \right\rfloor, 9 \right) = 3$$

21571 par 59

215/59

$$\tilde{q} = \min \left( \left\lfloor \frac{21}{5} \right\rfloor, 9 \right) = 4$$

$$215 - 4 \times 59 = -21$$

$$\tilde{q} = 3, 215 - 3 \times 59 = 38$$

---

387 par 59

$$\tilde{q} = \min \left( \left\lfloor \frac{38}{5} \right\rfloor, 9 \right) = 7$$

$$387 - 7 \times 59 = -26$$

$$\tilde{q} = 6, 387 - 6 \times 59 = 33$$

---

331 par 59

$$\tilde{q} = \min \left( \left\lfloor \frac{33}{5} \right\rfloor, 9 \right) = 6$$

$$331 - 6 \times 59 = -23$$

$$331 - 5 \times 59 = 36$$

# Le PGCD

## Rappels de math

$d$  est un diviseur de  $a$ , s'il existe  $\alpha$  tel que  $a = \alpha \times d$ ,  $a$  est un multiple de  $d$ .

Propriété :  $\text{pgcd}(a, 0) = a$

Propriété :  $\text{pgcd}(a, a) = a$

Propriété :  $\text{pgcd}(a, b) = \text{pgcd}(b, a)$

Propriété : si  $a > b$  alors  $\text{pgcd}(a, b) = \text{pgcd}(a - b, b)$

Propriété : si  $a > b$  alors  $\text{pgcd}(a, b) = \text{pgcd}(a \% b, b)$

# Le PGCD

## Rappels de math (suite)

**Définition :** Si  $\text{pgcd}(a, b) = 1$  alors  $a$  et  $b$  sont dits **premiers entre eux**.

**Propriété :** Si  $a$  et  $b$  sont choisis aléatoirement, la probabilité qu'ils soient premiers entre eux est de  $\frac{6}{\pi^2} \simeq 0.6$

**Simplification de fractions :** **fraction irréductible** si le pgcd du numérateur et du dénominateur vaut 1.

**Calcul du ppcm :**  $\text{ppcm}(a, b) = \frac{a \times b}{\text{pgcd}(a, b)}$

# Le PGCD - Euclide

## Algorithme à base de soustraction

Entrées  $A$  et  $B$

Sortie  $U$  le pgcd de  $A$  et de  $B$

Corps

$$U \leftarrow A$$

$$V \leftarrow B$$

**Tant que**  $V > 0$  **faire**

Si  $V > U$  alors

$$T \leftarrow V, V \leftarrow U, U \leftarrow T$$

$$U \leftarrow U - V$$

*Invariant de boucle* :  $U \geq V$  et  $\text{pgcd}(U, V) = \text{pgcd}(U - V, V)$

*Terminaison* : si  $V = 0$  alors  $U = \text{pgcd}(U, V)$

## Le PGCD - Euclide

Exemple

U	V	U - V
1479	699	780
780	699	81
699	81	618
618	81	537
...	...	...
213	81	132
132	81	51
81	51	30
51	30	21
30	21	9
21	9	12
12	9	3
9	3	6
6	3	3
3	3	0
3	0	

# Le PGCD - Euclide

## Algorithme à base de division

Entrées  $A$  et  $B$

Sortie  $U$  le pgcd de  $A$  et de  $B$

Corps

$$U \leftarrow A$$

$$V \leftarrow B$$

**Tant que**  $V > 0$  **faire**

$$T \leftarrow U \% V$$

$$U \leftarrow V$$

$$V \leftarrow T$$

# Le PGCD

Algorithme à base de division : exemple

U	V	U % V
1479	699	81
699	81	51
81	51	30
51	30	21
30	21	9
21	9	3
9	3	0
3	0	



# Le PGCD

## Algorithme récursif

**Entrées**  $A$  et  $B$  avec  $A > B$

**Sortie** le pgcd de  $A$  et de  $B$

**Corps**

Si  $B = 0$  alors  $A$

Sinon  $\text{PGCD}(B, A \% B)$

# Le PGCD

## Le pire cas la suite de Fibonacci

- ▶ La suite de Fibonacci est définie par récurrence :

$$\begin{cases} F_0 = 0 \\ F_1 = 1 \\ F_n = F_{n-1} + F_{n-2} \end{cases}$$

- ▶ Equation caractéristique :  $x^2 - x - 1 = 0$ , solutions  $\phi = \frac{1+\sqrt{5}}{2}$  et  $\phi' = \frac{1-\sqrt{5}}{2}$  et  $F_n = \frac{1}{\sqrt{5}}(\phi^n - \phi'^n)$
- ▶  $\text{pgcd}(F_n, F_{n-1})$  demande  $n$  itérations, soit une complexité par rapport à la taille de  $F_n$  en base  $\phi$  ( $\log_{\phi} N$ ).

# Le PGCD

## Algorithme binaire (parité)

**Entrées**  $A$  et  $B$  avec  $A > B$ ,  $A$  et  $B$  impairs

**Sortie** le pgcd de  $A$  et de  $B$

**Corps**

**Si**  $B = 0$  **alors**  $A$

**Sinon**

$T \leftarrow A - B$

**Tant que**  $T$  pair **faire**  $T \leftarrow T/2$

**Si**  $T > B$  **alors** PGCD( $T, B$ )

**Sinon** PGCD( $B, T$ )

Comme  $A$  et  $B$  sont impairs,  $T$  est pair.

# Le PGCD

Algorithme à binaire (parité): Exemple

U	V	U - V
1479	699	780 (/4 → 195)
699	195	504 (/8 → 63)
195	63	132 (/4 → 33)
63	33	30 (/2 → 15)
33	15	18 (/2 → 9)
15	9	6 (/2 → 3)
9	3	6 (/2 → 3)
3	3	0
3	0	

# Calcul modulaire Congruence

## rappels de math

**Définition :** Deux entiers  $a$  et  $b$  sont congruents modulo  $n$  (un entier) si  $a - b$  est un multiple de  $n$ .

On note  $a \equiv b \pmod{n}$

**Propriété :** La relation de congruence est une **relation d'équivalence**,

**Réflexive :**  $a \equiv a \pmod{n}$

**Symétrique :**  $a \equiv b \pmod{n} \Leftrightarrow b \equiv a \pmod{n}$

**Transitive :**  $a \equiv b \pmod{n}$  et  $b \equiv c \pmod{n} \Leftrightarrow a \equiv c \pmod{n}$

# Calcul modulaire Congruence

## Opérations et Congruence

**Addition :** Si  $a \equiv b \pmod{n}$  et  $c \equiv d \pmod{n}$   
alors  $a + c \equiv b + d \pmod{n}$

**Multiplication :** Si  $a \equiv b \pmod{n}$  et  $c \equiv d \pmod{n}$   
alors  $a \times c \equiv b \times d \pmod{n}$

**Puissance :** Si  $a \equiv b \pmod{n}$  et  $0 < k$  alors  $a^k \equiv b^k \pmod{n}$

**Inverse :** Si  $\text{pgcd}(a, n) = 1$   
alors il existe  $b$  tel que  $a \times b \equiv 1 \pmod{n}$

**Théorème** (Petit théorème de Fermat) Si  $n$  nombre premier et  $\text{pgcd}(a, n) = 1$  alors  $a^{n-1} \equiv 1 \pmod{n}$

## Calcul modulaire Congruence

L'anneau  $\mathbb{Z}/n\mathbb{Z}$

**Addition :**  $(\mathbb{Z}/n\mathbb{Z}, +)$  est un **groupe commutatif**.

**Multiplication :**  $(\mathbb{Z}/n\mathbb{Z}, +, \times)$  est un **anneau commutatif unitaire**.

**Inverse :** Attention, seuls les nombres premiers avec  $n$ , ont un inverse.

**Exemple :** Résoudre dans  $\mathbb{Z}/10\mathbb{Z}$  l'équation  $2x \equiv 6 \pmod{10}$   
2 n'est pas inversible, nous remarquons deux solutions  $x = 3$  et  $x = 8$   
Par contre  $3x \equiv 6 \pmod{10}$ , n'a qu'une seule solution, 3 est inversible,  $3^{-1} = 7 \pmod{10}$   
 $x = 2$  est l'unique solution.

## Euclide étendu - Inverse Modulaire

### rappels de math

**Identité de Bezout :** Soient  $a$  et  $b$  deux entiers, il existe deux entiers  $u$  et  $v$  tels que :  $a \times u + b \times v = \text{pgcd}(a, b)$

**Remarque 1 :**  $a$  et  $b$  premiers entre eux  $\Leftrightarrow a \times u + b \times v = 1$

**Remarque 2 :**  $a \times u + b \times v = 1$   
 $\Rightarrow a^{-1} \equiv u \pmod{b}$  et  $b^{-1} \equiv v \pmod{a}$



## Euclide étendu - Inverse Modulaire

L'algorithme :  $u_1 \times a + u_2 \times b = u_3$

Entrées :  $a$  et  $b$

Sortie :  $u_1 \times a + u_2 \times b = \text{pgcd}(a, b)$

Initialisation :  $(u_1, u_2, u_3) \leftarrow (1, 0, a)$   
 $(v_1, v_2, v_3) \leftarrow (0, 1, b)$

Itération : tant que  $v_3 \neq 0$  :  $q = \lfloor u_3 \div v_3 \rfloor$   
 $(t_1, t_2, t_3) \leftarrow (u_1, u_2, u_3) - q \times (v_1, v_2, v_3)$   
 $(u_1, u_2, u_3) \leftarrow (v_1, v_2, v_3)$   
 $(v_1, v_2, v_3) \leftarrow (t_1, t_2, t_3)$

## Euclide étendu - Inverse Modulaire

### L'algorithme allégé

Entrées :  $a$  et  $b$

Sortie : si  $u_3 = 1$  alors  $u_1 \equiv a^{-1} \pmod{b}$

Initialisation :

$$\begin{aligned}(u_1, u_3) &\leftarrow (1, a) \\ (v_1, v_3) &\leftarrow (0, b)\end{aligned}$$

Itération : tant que  $v_3 \neq 0$  :  $q = \lfloor u_3 \div v_3 \rfloor$

$$\begin{aligned}(t_1, t_3) &\leftarrow (u_1, u_3) - q \times (v_1, v_3) \\ (u_1, u_3) &\leftarrow (v_1, v_3) \\ (v_1, v_3) &\leftarrow (t_1, t_3)\end{aligned}$$

## Un algorithme de Division Exacte

Le dividende  $a$  est un multiple du diviseur  $d$

- ▶ Si on considère  $N$  premier avec le diviseur  $d$  et tel que  $N > \alpha$ ,  
 $a = \alpha \times d$ , alors  $\alpha \bmod N = a \times d^{-1} \bmod N$
- ▶ Mais le calcul de  $d^{-1} \bmod N$  peut être coûteux.
- ▶ Si  $d$  est premier avec la base  $\beta$  alors  $\alpha_0 = a_0 \times d_0^{-1} \bmod \beta$ .  
 et  $a - \alpha_0 \times d = 0 \bmod \beta$
- ▶ On peut ensuite construire  $\alpha_1$ ,  
 On pose  $t = \frac{a - \alpha_0 \times d}{\beta}$  et on calcule  $\alpha_1 = t_0 \times d_0^{-1} \bmod \beta$ .  
 On a  $t - \alpha_1 \times d = 0 \bmod \beta$   
 et,  $t \times \beta - \alpha_1 \times \beta \times d = 0 \bmod \beta^2$   
 or,  $t \times \beta - \alpha_1 \times \beta \times d = a - (\alpha_0 + \alpha_1 \times \beta) \times d$

## Un algorithme de Division Exacte

l'algorithme avec  $\beta^{m+n} > a$  et  $d > \beta^{n-1}$

**Entrées**  $a_{n+m-1} \dots a_0$  et  $d_{n-1} \dots d_0$ , base  $\beta$

**Sorties**  $q_m \dots q_0$  tel que  $a = q \times d$

**Corps**  $s \leftarrow d_0^{-1} \bmod \beta$

$t \leftarrow a$

Pour  $i = 0$  à  $m$  faire

$q_i \leftarrow t_0 \times s \bmod \beta$

$t \leftarrow (t - d \times q_i) / \beta$

## Un algorithme de Division Exacte

l'algorithme avec  $\beta^{m+n} > a$  et  $d > \beta^{n-1}$

- ▶  $a := 13 * 273$ ;  $d := 13$ ;  $b := 10$ ;  $m := 2$ ;  $\text{DivExact}(a, d, b, m)$ ;  
 $s = 7$  ;  $t = 3549$   
 $q[0] = 3$  ;  $t = 351$   
 $q[1] = 7$  ;  $t = 26$   
 $q[2] = 2$  ;  $t = 0$
- ▶  $a := 13 * 273$ ;  $d := 13$ ;  $b := 2$ ;  $m := 8$ ;  $\text{DivExact}(a, d, b, m)$ ;  
 $s = 1$  ;  $t = 3549$   
 $q[0] = 1$  ;  $t = 1768$   
 $q[1] = 0$  ;  $t = 884$   
 $q[2] = 0$  ;  $t = 442$   
 $q[3] = 0$  ;  $t = 221$   
 $q[4] = 1$  ;  $t = 104$   
 $q[5] = 0$  ;  $t = 52$   
 $q[6] = 0$  ;  $t = 26$   
 $q[7] = 0$  ;  $t = 13$   
 $q[8] = 1$  ;  $t = 0$

# Représentation des flottants algorithmes élémentaires

## Les nombres rationnels

### Quelques rappels

**Relation d'équivalence** : Construction de l'ensemble des nombres rationnels  $\mathbb{Q}$  peut se faire à partir d'une relation d'équivalence : soient  $(a, b)$  et  $(c, d)$  deux couples de  $\mathbb{Z} \times \mathbb{Z}^*$ ,  $(a, b)\mathcal{R}(c, d) \Leftrightarrow a \times d = b \times c$

**Addition** :  $(a, b) + (c, d) = (a \times d + c \times b, b \times d)$

**Multiplication** :  $(a, b) \times (c, d) = (a \times c, b \times d)$

$(\mathbb{Q}, +, \times)$  est un **corps commutatif** :  $(\mathbb{Q}, +)$  et  $(\mathbb{Q}^*, \times)$  sont des groupes commutatifs et la multiplication est distributive sur l'addition.

**Inverse** :  $(a, b) \times (b, a) = (a \times b, b \times a) = (1, 1)$

# Les nombres réels

## Quelques rappels

**Suites rationnelles de Cauchy** :  $(u_n)_{n \in \mathbb{N}}$  avec  $u_n \in \mathbb{Q}$ , telle que pour tout  $\epsilon$ , il existe  $n_0$ , qui pour tout  $n \geq n_0$  et tout  $m \geq n_0$ ,  $|u_n - u_m| \leq \epsilon$

**Relation d'équivalence** : Deux suites de Cauchy  $(u_n)_{n \in \mathbb{N}}$  et  $(v_n)_{n \in \mathbb{N}}$  sont équivalentes si la suite  $(u_n - v_n)_{n \in \mathbb{N}}$  converge vers zéro.

**Corps des réels  $\mathbb{R}$**  :  $(\mathbb{R}, +)$  et  $(\mathbb{R}^*, \times)$  sont deux groupes commutatifs et la multiplication est distributive sur l'addition.



# Représentation des nombres réels

## Exemples

**Partie entière - partie fractionnaire** En base  $\beta$  un nombre peut être représenté par  $A = \sum_{-\infty}^{+\infty} a_i \beta^i$

**Nombres rationnels** La partie fractionnaire  $A_f = \sum_{-\infty}^{-1} a_i \beta^i$  est périodique à partir d'un certain rang.

$$\frac{237}{315} = 0.7523809523809\overline{523809}$$

**Nombres réels** La partie fractionnaire peut être apériodique.

$$\sqrt{2} = 1.414213562373095048801688724209\dots$$

$$\sqrt{2} = 1 + \frac{1}{2} - \frac{1}{8} + \frac{1}{16} - \frac{5}{128} \dots + (-1)^{n+1} \frac{\binom{2n}{n}}{(2n-1)2^{2n}}$$

$$e = 1 + \sum_{n=1}^{\infty} \frac{1}{n!} = 2.71828182845904523536\dots$$

$$\pi = 4 \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} = 3.141592653589793238\dots$$

## écriture scientifique

- ▶ Représentation des nombres normalisée

$$X = (-1)^s \times x_0, x_1 x_2 \dots x_n \times \beta^e = (-1)^s \times \sum_{i=0}^n x_i \times \beta^{-i+e}$$

avec  $x_0 \neq 0$  (sauf si  $X = 0$ ).

- ▶ Le nombre de chiffres est fixé : notion de valeur approchée.
- ▶  $\frac{-237}{315} = (-1)^1 \times 7.523809523 \times 10^{-1}$   
 $\sqrt{2} = (-1)^0 \times 1.414213562 \times 10^0$

## Norme IEEE 754

- ▶ Un nombre est représenté en simple précision (32 bits) ou en double précision (64 bits) de la façon suivante :

s(1)	e = exp biaisé (8 ou 11)	... mantisse (23 ou 52)
------	--------------------------	-------------------------

- ▶ Si  $e \neq 0$  et  $e \neq 255(11111111)$  ou  $e \neq 2047(111111111111)$

$$(-1)^s \times 2^{(e)-b} \times 1, \dots \text{mantisse} \dots$$

Le biais vaut  $b = 127$  (01111111) en simple, et  $b = 1023$  (01111111111) en double.

- ▶ si, exp biaisé = 0:

$$(-1)^s \times 2^{1-\text{biais}} \times 0, \dots \text{mantisse} \dots$$

- ▶ Si  $e = 255(11111111)$  (ou  $e = 2047(111111111111)$ ) :  
alors le nombre représente l'infini si la mantisse est nulle, ou NaN (not a number) sinon.

## Remarques

- ▶ Si  $e \neq 0$  et  $e \neq 255(11111111)$  ou  $e \neq 2047(11111111111)$  alors l'écriture est dite normalisée et la mantisse est précédée par un 1 dit implicite.
- ▶ Sinon, l'écriture est dite dénormalisée. Ce qui permet de représenter les petites valeurs, ou encore des indicateurs.
- ▶ Attention 0 peut avoir un signe



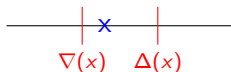
## Valeurs extrêmes

Nombre	Représentation en double précision (hexadécimale)	Valeur décimale
+0	00000000 00000000	0.0
-0	80000000 00000000	-0.0
1	3FF00000 00000000	1.0
2	40000000 00000000	2.0
Normalisé maximum	7FEFFFFFF FFFFFFFF	$1.7976931348623157e + 308$
Normalisé positif minimum	00100000 00000000	$2.2250738585072014e - 308$
Dénormalisé maximum	000FFFFFF FFFFFFFF	$2.2250738585072009e - 308$
Dénormalisé positif minimum	00000000 00000001	$4.9406564584124654e - 324$
$-\infty$	FFF00000 00000000	$-\infty$
$+\infty$	7FF00000 00000000	$+\infty$
<i>NAN</i>	7FF xxxx...xxxxxxxx	Not A Number

## Norme IEEE-754: nécessité d'arrondir

- ▶ Si  $x$  et  $y$  deux nombres exactement représentables en machine, alors le résultat d'une opération  $res = x \odot y$  n'est pas forcément représentable en machine.
- ▶ Par exemple, en base  $B = 10$ , le nombre  $1/3$  n'est pas représentable avec un nombre fini de chiffres.  
En base 2,  $1/10$  n'est pas représentable exactement en flottant.
- ▶ Il faut *arrondir* le résultat, autrement dit, trouver un nombre représentable voisin et savoir comment il a été choisi.

## Norme IEEE-754: modes d'arrondis



La norme propose 4 modes d'arrondi :

- ▶ arrondi *vers*  $+\infty$  (ou par excès), noté  $\Delta(x)$  : retourne le plus petit nombre machine supérieur ou égal au résultat exact  $x$
- ▶ arrondi *vers*  $-\infty$  (ou par défaut), noté  $\nabla(x)$  : retourne le plus grand nombre machine inférieur ou égal au résultat exact  $x$
- ▶ arrondi *vers* 0, noté  $\mathcal{Z}(x)$  : retourne  $\Delta(x)$  pour les nombres négatifs et  $\nabla(x)$  pour les positifs
- ▶ arrondi *au plus près*, noté  $\circ(x)$  : retourne le nombre machine le plus proche du résultat exact  $x$  (pour le milieu de deux nombres machine consécutifs on choisit celui dont la mantisse se termine par un 0, on parle d'*arrondi pair*)

Les trois premiers modes d'arrondis sont dits *dirigés*.



## Norme IEEE-754: propriété de l'arrondi correct

Soient  $x$  et  $y$  sont deux nombres exactement représentables en machine,  $\odot$  une des opérations rationnelles  $+$ ,  $-$ ,  $\times$ ,  $/$  et  $\diamond$  le mode d'arrondi choisi parmi les 4 modes IEEE.

La norme IEEE exige que le résultat d'une opération  $x \odot y$  soit égal à  $\diamond(x \odot_{\text{exact}} y)$ . Le résultat doit être le même que si on effectuait le calcul en précision infinie puis on arrondissait ce résultat.

Idem pour la racine carrée.

C'est la propriété de *l'arrondi correct*.

*La norme IEEE décrit un algorithme pour l'addition, la soustraction, la multiplication, la division et la racine carrée et exige que ses implémentations produisent le même résultat que cet algorithme.*

## Norme IEEE-754: comparaisons

La norme impose que l'opération de comparaison soit exacte et ne produise pas de dépassement de capacité.

Les comparaisons spécifiées dans la norme sont :

- ▶ égalité
- ▶ supérieur
- ▶ inférieur

Le signe de zéro n'est pas pris en compte.

Dans le cas de comparaisons impliquant un NaN, la comparaison retourne faux.

Plus précisément dans le cas de l'égalité : si  $x = \text{NaN}$  alors  $x = x$  retourne faux et  $x \neq x$  retourne vrai (l'égalité n'est pas réflexive mais c'est un moyen d'identifier un NaN)

## Norme IEEE-754: les drapeaux

Aucun calcul ne doit entraver le bon fonctionnement de la machine. Un mécanisme de 5 drapeaux permet d'informer le système sur le comportement des opérations:

**INVALID operation** le résultat par défaut est NaN

**DIVIDE by ZÉRO** le résultat est  $\pm\infty$

**OVERFLOW** dépassement de capacité vers  $\infty$ : le résultat est soit  $\pm\infty$  soit le plus grand nombre représentable (en valeur absolue) suivant le signe du résultat exact et du mode d'arrondi

**UNDERFLOW** dépassement de capacité vers 0 : le résultat est soit  $\pm 0$  soit un dénormalisé

**INEXACT** résultat inexact : levé lorsque que le résultat d'une opération n'est pas exact (presque toujours ignoré)

Ces drapeaux, une fois levés, le restent pendant tout le calcul jusqu'à une remise à zéro volontaire (*sticky flags*). Ils peuvent être lus et écrits par l'utilisateur.

# L'addition

## Principe

1. **Alignement** des mantisses en fonction du plus grand exposant

Exemple : addition de  $9 + 1/10$

$$9 = 0\ 10000010\ 001000000000000000000000$$

$$1/10 = 0\ 01111011\ 10110011001100110011001$$

$$1/10 = 0\ 10000010\ 00000110110011001100110011001$$

2. **Addition mantisses**

$$9 = 0\ 10000010\ 001000000000000000000000$$

$$1/10 = 0\ 10000010\ 0000001101100110011001100110011001$$

$$9 + 1/10 = 0\ 10000010\ 0010001101100110011001100110011001$$

3. **Normalisation** suivant l'arrondi (ici inférieur)

$$9 + 1/10 = 0\ 10000010\ 001000110110011001100110011$$

## Effet de bord : l'absorption

- ▶ L'absorption se produit lors de l'addition (voire la soustraction) de deux nombres très différents en ordre de grandeur : le plus petit peut ne pas être pris en compte.

- ▶ Exemple en base 10 sur dix chiffres de mantisse :

$$1.203941025 * 10^{13} + 1.2 * 10^1$$

Alignement :  $1.203941025 * 10^{13} + 0.0000000000012 * 10^{13}$

Addition :  $1.2039410250012 * 10^{13}$

Normalisation :  $1.203941025 * 10^{13}$

## Effet de bord : L'algorithme de Gentleman (1969)

**Retour**  $b$  la base et  $c$  le nombre de chiffres.

**Initialisation**  $a \leftarrow 1.0$ ;  $b \leftarrow 1.0$ ;

**Tant que**  $((((a + 1.0) - a) - 1.0) = 0)$

**faire**  $a \leftarrow a \times 2$

**Tant que**  $((((a + b) - a) - b) <> 0)$  **faire**

$b \leftarrow b + 1.0$

$c \leftarrow 0$ ;  $a \leftarrow 1$ ;

**Tant que**  $((((a + 1) - a) - 1.0) = 0)$  **faire**

$a \leftarrow a * b$ ;  $c \leftarrow c + 1$ ;

`gentleman()`; (*testé sur Maple*)

$b = 10.0$   $c = 10$

## Effets de bord : l'élimination

- ▶ Lors d'une soustraction de deux valeurs très proches il peut y avoir une grosse perte d'information.
- ▶ élimination
- ▶ Exemple :  $1.203941025 * 10^{13} - 1.203941012 * 10^{13}$ 
  - Alignement :  $1.203941025 * 10^{13} - 1.203941012 * 10^{13}$
  - Addition :  $0.000000013 * 10^{13}$
  - Normalisation :  $1.300000000 * 10^5$
- ▶ Nous n'avons plus que deux chiffres significatifs.

# L'addition

## Avertissement

- ▶ Les propriétés sur  $\mathbb{R}$  ne sont pas conservées.
- ▶ Exemple de l'associativité : en base 10 sur dix chiffres  
 $((9.999999999 + 0.0000000004) + 0.0000000003) + 0.0000000003 = 9.999999999$   
et  
 $9.999999999 + ((0.0000000004 + 0.0000000003) + 0.0000000003) = 10.000000000$



# La multiplication

## Principe

- ▶ Nous désirons effectuer :  
 $1.203941025 * 10^{13} \times 8.2124351623 * 10^1$
- ▶ Nous sommes en base dix avec des mantisses de dix chiffres

### 1. **Produit mantisses**

$$1203941025 \times 82124351623 = 98872876070455033575$$

### 2. **Somme exposant** $13 + 1 = 14$

### 3. **Normalisation** $9.887287607 * 10^{14}$

# La multiplication

## Avertissement

- ▶ Les propriétés sur  $\mathbb{R}$  ne sont pas conservées.
- ▶ Exemple de l'inverse : en base 10 sur dix chiffres

$$\frac{1}{3} = 3.33333333 * 10^{-1}$$

$$3 * \frac{1}{3} = 9.99999999 * 10^{-1}$$

$$1 - 3 * \frac{1}{3} = 1 * 10^{-10}$$

# Représentation des matrices et arithmétique

— — — —

## Algorithmes et complexité

# Notion de Vecteurs

## Quelques rappels

Un vecteur  $V$  est un élément d'un espace vectoriel

Un espace vectoriel  $E$  est un ensemble défini sur un corps  $\mathbb{K}$  (par exemple  $\mathbb{R}$ ) muni d'une loi interne (addition) et d'une loi externe (produit par un scalaire  $\lambda \cdot V$ )  
 $(E, +)$  est un groupe abélien, le produit par un scalaire est distributif, associatif et possède un élément neutre  $1_{\mathbb{K}}$ .

Base  $(B_1, \dots, B_n)$  famille de vecteurs, génératrice (tout  $V \in E$ , s'écrit  $V = \sum_{i=1}^n \lambda_i B_i$ ) et libre  
 $(\sum_{i=1}^n \lambda_i B_i = 0_E \Rightarrow \lambda_i = 0_{\mathbb{K}})$

# Notion de Vecteurs

## Quelques rappels

La **Dimension** est donnée par le nombre d'éléments de la base

**Exemples :**  $\mathbb{R}^n$ , l'espace vectoriel de dimension  $n$  sur  $\mathbb{R}$  :  $\mathbb{R}^2$  le plan,  $\mathbb{R}^3$  l'espace...

**Coordonnées :** d'un vecteur sont les coefficients de son écriture

dans la base :  $V = \sum_{i=1}^n v_i B_i$ , on note  $\begin{pmatrix} v_1 \\ v_2 \\ \dots \\ v_n \end{pmatrix}$

## Opérations sur les Vecteurs

**Addition** Soient  $V \in E$  et  $W \in E$ ,

$$V \begin{pmatrix} v_1 \\ v_2 \\ \dots \\ v_n \end{pmatrix} + W \begin{pmatrix} w_1 \\ w_2 \\ \dots \\ w_n \end{pmatrix} = S \begin{pmatrix} v_1 + w_1 \\ v_2 + w_2 \\ \dots \\ v_n + w_n \end{pmatrix} \in E$$

**Produit par un scalaire** Soient  $\lambda \in \mathbb{K}$  et  $V \in E$ ,

$$\lambda \times V \begin{pmatrix} v_1 \\ v_2 \\ \dots \\ v_n \end{pmatrix} = P \begin{pmatrix} \lambda v_1 \\ \lambda v_2 \\ \dots \\ \lambda v_n \end{pmatrix} \in E$$

**Combinaison linéaire** Soient  $\lambda_i \in \mathbb{K}$  et  $V_i \in E$ ,  $\sum_{i=1}^n \lambda_i V_i \in E$

## Opérations sur les Vecteurs

Transposé Soit  $V \in E$ ,

$$\left[ V \begin{pmatrix} v_1 \\ v_2 \\ \dots \\ v_n \end{pmatrix} \right]^T = V^T ( v_1 \quad v_2 \quad \dots \quad v_n )$$

Vecteur colonne et vecteur ligne

Produit Scalaire Soient  $V \in E$  et  $W \in E$ ,

$$V \cdot W = ( v_1 \quad v_2 \quad \dots \quad v_n ) \cdot \begin{pmatrix} w_1 \\ w_2 \\ \dots \\ w_n \end{pmatrix} = \sum_{i=1}^n v_i \times w_i \in \mathbb{K}$$

A deux vecteurs associe un scalaire

# Opérations sur les Vecteurs

## Produit Scalaire : Algorithme

ProdScal( $V, W, n$ )

**Entrées**  $V ( v_1 \ v_2 \ \dots \ v_n )$  et  $W ( w_1 \ w_2 \ \dots \ w_n )$   
deux vecteurs de  $E$ .

**Sortie**  $\lambda \in \mathbb{K}$  avec  $\lambda = V \cdot W$

**Corps**  $\lambda \leftarrow 0$

Pour  $i = 1$  à  $n$  faire

$\lambda \leftarrow \lambda + v_i \times w_i$



# Notion de Matrices

## Quelques rappels

**Application linéaire** : Soient  $E$  et  $F$  deux espaces vectoriels sur  $\mathbb{K}$  de dimensions respectives  $n$  et  $m$ , une application  $\mathcal{A}$  de  $E$  dans  $F$  est linéaire ssi :

pour tous  $\lambda, \nu \in \mathbb{K}$  et  $V, W \in E$

$$\mathcal{A}(\lambda V + \nu W) = \lambda \mathcal{A}(V) + \nu \mathcal{A}(W) \in F$$

**Matrice associée** : Une application linéaire est parfaitement définie par la donnée des images des éléments d'une base de  $E$ .

L'ensemble des vecteurs images forment une matrice  $A$  de dimension  $m \times n$  où chaque colonne est le vecteur image d'un élément de la base de  $E$ .

## Notion de Matrices

### Application linéaire, matrice

Soient  $E$  et  $F$  deux espaces vectoriels sur  $\mathbb{K}$  de dimensions respectives  $n$  et  $m$ ,  $(e_1, \dots, e_n)$  une base de  $E$  et  $(f_1, \dots, f_m)$  base de  $F$ .

On note  $\begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m,1} & a_{m,2} & \dots & a_{m,n} \end{pmatrix}$  la matrice associée

à l'application linéaire  $\mathcal{A}$  telle que  $\mathcal{A}(e_k) = \sum_{i=1}^m a_{i,k} f_i$

# Notion de Matrices

## Quelques rappels

**Endomorphisme** est une application de  $E$  dans  $E$ , la matrice associée est de dimension  $n \times n$ . Elle est dite carrée.

**Changement de base** l'opération de changement de base dans un espace vectoriel  $E$  se traduit par un endomorphisme bijectif (automorphisme)

# Opérations sur les Matrices

## Addition

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m,1} & a_{m,2} & \dots & a_{m,n} \end{pmatrix} + \begin{pmatrix} b_{1,1} & b_{1,2} & \dots & b_{1,n} \\ b_{2,1} & b_{2,2} & \dots & b_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ b_{m,1} & b_{m,2} & \dots & b_{m,n} \end{pmatrix}$$

$$= \begin{pmatrix} a_{1,1} + b_{1,1} & a_{1,2} + b_{1,2} & \dots & a_{1,n} + b_{1,n} \\ a_{2,1} + b_{2,1} & a_{2,2} + b_{2,2} & \dots & a_{2,n} + b_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m,1} + b_{m,1} & a_{m,2} + b_{m,2} & \dots & a_{m,n} + b_{m,n} \end{pmatrix}$$

## Opérations sur les Matrices

### Produit par un scalaire

$$\lambda \cdot \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m,1} & a_{m,2} & \dots & a_{m,n} \end{pmatrix} = \begin{pmatrix} \lambda a_{1,1} & \lambda a_{1,2} & \dots & \lambda a_{1,n} \\ \lambda a_{2,1} & \lambda a_{2,2} & \dots & \lambda a_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ \lambda a_{m,1} & \lambda a_{m,2} & \dots & \lambda a_{m,n} \end{pmatrix}$$

- ▶ L'ensemble des applications linéaires de  $E$  sur  $F$  muni de l'addition et du produit par un scalaire forme un espace vectoriel.

# Opérations sur les Matrices

## Algorithme d'Addition

AddMat( $A, B, m, n$ )

**Entrées**  $A$  et  $B$  deux matrices  $m \times n$

**Sortie**  $S$  une matrice  $m \times n$

**Corps** Pour  $i = 1$  à  $m$  faire

    Pour  $j = 1$  à  $n$  faire

$$s_{i,j} \leftarrow a_{i,j} + b_{i,j}$$

Complexité quadratique :  $m \times n$  opérations

# Opérations sur les Matrices

## Produit Matrice Vecteur : application linéaire

**Propriété** Soit  $\mathcal{A}$  une application linéaire de  $E$  dans  $F$  de matrice  $A$  de dimension  $m \times n$ .

Pour  $V \in E$  nous avons,

$$\mathcal{A}(V) = \mathcal{A}\left(\sum_{j=1}^n v_j e_j\right) = \sum_{j=1}^n v_j \mathcal{A}(e_j)$$

**Définition du produit matrice vecteur**

$$\mathcal{A}(V) = A \times V = \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m,1} & a_{m,2} & \dots & a_{m,n} \end{pmatrix} \times \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}$$

# Opérations sur les Matrices

## Produit Matrice Vecteur : Algorithme

ProdMatVec( $A, V, m, n$ )

**Entrées**  $A$  une matrice  $m \times n$  et  $V \in E$  avec  $L_i \in E$  la ligne  $i$  de la matrice  $A$ .

**Sortie**  $W \in F$

**Corps** Pour  $i = 1$  à  $m$  faire  
 $w_i \leftarrow \text{ProdScal}(L_i, V, n)$



# Opérations sur les Matrices

## Produit Matrice-Vecteur : Algorithme

ProdMatVec( $A, V, m, n$ )

**Entrées**  $A$  une matrice  $m \times n$  et  $V \in E$  avec  $L_i \in E$  la ligne  $i$  de la matrice  $A$ .

**Sortie**  $W \in F$

**Corps** Pour  $i = 1$  à  $m$  faire

$$w_i \leftarrow 0$$

Pour  $j = 1$  à  $n$  faire

$$w_i \leftarrow w_i + a_{i,j} \times v_j$$

Complexité quadratique :  $m \times n$  opérations

# Opérations sur les Matrices

## Composition d'applications linéaires : Multiplication de matrices

**Définition** Soient  $\mathcal{A}$  et  $\mathcal{B}$  deux applications linéaires respectivement de  $E_n$  dans  $F_m$ , et de  $F_m$  dans  $G_k$ , on appelle application composée  $\mathcal{P}$  de  $E_n$  dans  $G_k$ , telle que  $\mathcal{P}(V) = \mathcal{B}(\mathcal{A}(V))$ .

**Propriété**  $\mathcal{P}$  est une application linéaire de  $E_n$  dans  $G_k$ .

**Propriété** La matrice  $P$  associée à  $\mathcal{P}$  est de dimension  $k \times n$  appelée matrice produit :  $P = B \times A$  obtenue par les images de la base de  $E_n$  par  $\mathcal{P}$ .

# Opérations sur les Matrices

## Multiplication de matrices

$$\begin{pmatrix} b_{1,1} & b_{1,2} & \dots & b_{1,m} \\ b_{2,1} & b_{2,2} & \dots & b_{2,m} \\ \vdots & \vdots & \vdots & \vdots \\ b_{k,1} & b_{k,2} & \dots & b_{k,m} \end{pmatrix} \times \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m,1} & a_{m,2} & \dots & a_{m,n} \end{pmatrix}$$

$$= \begin{pmatrix} p_{1,1} & p_{1,2} & \dots & p_{1,n} \\ p_{2,1} & p_{2,2} & \dots & p_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ p_{k,1} & p_{k,2} & \dots & p_{k,n} \end{pmatrix} \text{ avec}$$

$$p_{i,j} = \begin{pmatrix} b_{i,1} & b_{i,2} & \dots & b_{i,m} \end{pmatrix} \cdot \begin{pmatrix} a_{1,j} \\ a_{2,j} \\ \vdots \\ a_{m,j} \end{pmatrix}$$

# Opérations sur les Matrices

## Algorithme de Multiplication

ProdMat ( $A, B, m, n, k$ )

**Entrées**  $A$  une matrice  $m \times n$  et  $B$  une matrice  $k \times m$  avec  $C_j \in E$  la colonne  $j$  de la matrice  $A$ .

**Sortie**  $P$  une matrice  $k \times n$  avec  $P = B \times A$  avec  $P_j$  colonne  $j$ .

**Corps** Pour  $j = 1$  à  $n$  faire

$$P_j \leftarrow \text{ProdMatVec}(B, C_j, k, m)$$

Complexité cubique :  $m \times n \times k$  opérations

# Opérations sur les Matrices

## Algorithme de Multiplication

ProdMat ( $A, B, m, n, k$ )

**Entrées**  $A$  une matrice  $m \times n$  et  $B$  une matrice  $k \times m$  avec  $C_j \in E$  la colonne  $j$  de la matrice  $A$  et  $L_i \in E$  la ligne  $i$  de la matrice  $B$ .

**Sortie**  $P$  une matrice  $k \times n$  avec  $P = B \times A$

**Corps** Pour  $i = 1$  à  $k$  faire

    Pour  $j = 1$  à  $n$  faire

$$p_{i,j} \leftarrow \text{ProdScal}(L_i, C_j, m)$$

Complexité cubique :  $m \times n \times k$  opérations

# Opérations sur les Matrices

## Algorithme de Multiplication

ProdMat ( $A, B, m, n, k$ )

**Entrées**  $A$  une matrice  $m \times n$  et  $B$  une matrice  $k \times m$

**Sortie**  $P$  une matrice  $k \times n$  avec  $P = B \times A$

**Corps** Pour  $i = 1$  à  $k$  faire

    Pour  $j = 1$  à  $n$  faire

$p_{i,j} \leftarrow 0$

        Pour  $t = 1$  à  $m$  faire

$p_{i,j} \leftarrow p_{i,j} + b_{i,t} \times a_{t,j}$

Complexité cubique :  $m \times n \times k$  opérations

## Opérations sur les Matrices carrées $m = n = k$

### Algorithme de Multiplication : Découpage en bloc

Si  $n = 2^t$  alors nous pouvons avoir une approche récursive

$$\begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} \times \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} = \begin{pmatrix} P_{1,1} & P_{1,2} \\ P_{2,1} & P_{2,2} \end{pmatrix}$$

avec  $P_{i,j} = B_{i,1} \times A_{1,j} + B_{i,2} \times A_{2,j}$

et  $P_{i,j}, A_{i,j}, B_{i,j}$  des matrices de dimensions  $\frac{n}{2} \times \frac{n}{2}$

Intérêt : meilleure utilisation de la mémoire cache

# Opérations sur les Matrices

## Algorithme Récursif de Multiplication

ProdMatRec ( $A, B, n$ )

**Entrées**  $A$  et  $B$  deux matrices  $n \times n$

**Sortie**  $P$  une matrice  $n \times n$  avec  $P = B \times A$

**Corps** Si  $n = 1$  alors  $p_{1,1} \leftarrow b_{1,1} \times a_{1,1}$

Sinon

Pour  $i = 1$  à 2

    Pour  $j = 1$  à 2

$A_{i,j} \leftarrow \text{Quart}(A, i, j, n)$

$B_{i,j} \leftarrow \text{Quart}(B, i, j, n)$

Pour  $i = 1$  à 2

    Pour  $j = 1$  à 2

$P_{i,j} = \text{ProMatRec}(B_{i,1}, A_{1,j}, \frac{n}{2}) + \text{ProMatRec}(B_{i,2}, A_{2,j}, \frac{n}{2})$

Complexité cubique :  $n^3$  opérations



# Opérations sur les Matrices

## Algorithme Récursif de Multiplication

Quart( $A, s, t, n$ )

**Entrées**  $A$  une matrice  $n \times n$

**Sortie**  $Q$  une matrice  $\frac{n}{2} \times \frac{n}{2}$

**Corps** Pour  $i = 1$  à  $\frac{n}{2}$

    Pour  $j = 1$  à  $\frac{n}{2}$

$$q_{i,j} \leftarrow a_{(s-1)\frac{n}{2}+i, (t-1)\frac{n}{2}+j}$$

## Opérations sur les Matrices

### Produit par bloc : Strassen (1969)

Si  $n = 2^t$  alors nous pouvons avoir une approche récursive

$$\begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} \times \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} = \begin{pmatrix} P_{1,1} & P_{1,2} \\ P_{2,1} & P_{2,2} \end{pmatrix}$$

$$\begin{cases} T_1 = (B_{1,1} + B_{2,2}) \times (A_{1,1} + A_{2,2}) \\ T_2 = (B_{2,1} + B_{2,2}) \times A_{1,1} \\ T_3 = B_{1,1} \times (A_{1,2} - A_{2,2}) \\ T_4 = B_{2,2} \times (A_{2,1} - A_{1,1}) \\ T_5 = (B_{1,1} + B_{1,2}) \times A_{2,2} \\ T_6 = (B_{2,1} - B_{1,1}) \times (A_{1,1} + A_{1,2}) \\ T_7 = (B_{1,2} - B_{2,2}) \times (A_{2,1} + A_{2,2}) \end{cases} \quad \begin{cases} P_{1,1} = T_1 + T_4 - T_5 + T_7 \\ P_{1,2} = T_3 + T_5 \\ P_{2,1} = T_2 + T_4 \\ P_{2,2} = T_1 - T_2 + T_3 + T_6 \end{cases}$$

et  $P_{i,j}$ ,  $A_{i,j}$ ,  $B_{i,j}$  des matrices de dimensions  $\frac{n}{2} \times \frac{n}{2}$

# Opérations sur les Matrices

## Produit par bloc Strassen : Complexité

- ▶ Soit  $S(n)$  le nombre d'opérations élémentaires de l'algorithme de Strassen
- ▶  $S(n) = 7 \times S\left(\frac{n}{2}\right) + \alpha \times \left(\frac{n}{2}\right)^2$
- ▶ Si  $n = 2^t$  alors  $S(n) \simeq 7^t = 2^{t \log_2(7)} = n^{\log_2(7)} < n^3$

# Opérations sur les Matrices

Modes de stockage : sur un vecteur

POUR SIMPLIFIER

Déclaration :  $M[m \times n]$  un vecteur

Parcours lignes : Pour  $i = 0$  à  $m - 1$

Pour  $j = 0$  à  $n - 1$

$M[i \times n + j]$

Parcours colonne : Pour  $j = 0$  à  $n - 1$

Pour  $i = 0$  à  $m - 1$

$M[i + j \times n]$

# Opérations sur les Matrices

Modes de stockage : tableau de vecteurs

Déclaration :  $M[m][n]$  un tableau de  $m$  vecteurs de dimension  $n$

Parcours lignes : Pour  $i = 0$  à  $m - 1$

Pour  $j = 0$  à  $n - 1$

$M[i][j]$

# Opérations sur les Matrices

## Modes de stockage Dynamique

Sur un vecteur : Réserve d'un espace de taille  $m \times n$

Sur un tableau de vecteurs : Réserve d'un tableau d'adresses de taille  $m$

Pour chaque adresse réserve d'un espace pour stocker un vecteur de taille  $n$

# Algorithme de Gauss sur les flottants, sur les rationnels

## Résolution d'un système linéaire

### Approche matricielle

Nous désirons résoudre le système :

$$\begin{cases} a_{0,0}x_0 + a_{0,1}x_1 + \cdots + a_{0,n-1}x_{n-1} & = & b_0 \\ a_{1,0}x_0 + a_{1,1}x_1 + \cdots + a_{1,n-1}x_{n-1} & = & b_1 \\ & \vdots & \vdots \\ a_{n-1,0}x_0 + a_{n-1,1}x_1 + \cdots + a_{n-1,n-1}x_{n-1} & = & b_{n-1} \end{cases}$$

Ce qui peut se traduire par :

$$\begin{pmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,n-1} \\ \vdots & \vdots & \cdots & \vdots \\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{pmatrix}$$



# Résolution d'un système linéaire

## Méthode directe

- Nous pouvons utiliser les formules de Cramer

$$x_i = \frac{\det B_i}{\det A}$$

$$\text{avec } B_i = \begin{pmatrix} a_{0,0} & \cdot & a_{0,i-1} & b_0 & a_{0,i+1} & \cdot & a_{0,n-1} \\ a_{1,0} & \cdot & a_{1,i-1} & b_1 & a_{1,i+1} & \cdot & a_{1,n-1} \\ \vdots & \vdots & \cdot & \vdots & \cdot & \cdot & \cdot \\ a_{n-1,0} & \cdot & a_{n-1,i-1} & b_{n-1} & a_{n-1,i+1} & \cdot & a_{n-1,n-1} \end{pmatrix}$$

- Cette méthode est très coûteuse, le calcul d'un déterminant par les cofacteurs étant en  $O(n!)$

# Résolution d'un système linéaire

## Cas d'une matrice triangulaire

Notre Système est de la forme :

$$\begin{pmatrix} a_{0,0} & a_{0,1} & \dots & \dots & a_{0,n-1} \\ 0 & a_{1,1} & \dots & \dots & a_{1,n-1} \\ \vdots & \vdots & \dots & \dots & \vdots \\ 0 & 0 & \dots & a_{n-2,n-2} & a_{n-2,n-1} \\ 0 & 0 & \dots & 0 & a_{n-1,n-1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-2} \\ x_{n-1} \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-2} \\ b_{n-1} \end{pmatrix}$$

Nous remarquons que si  $a_{n-1,n-1} \neq 0$  alors  $x_{n-1} = \frac{b_{n-1}}{a_{n-1,n-1}}$

## Résolution d'un système linéaire

### Cas d'une matrice triangulaire (2)

Le système peut donc se réduire à  $n - 1$  équations sachant que

$$x_{n-1} = \frac{b_{n-1}}{a_{n-1,n-1}}$$

$$\begin{pmatrix} a_{0,0} & a_{0,1} & \dots & a_{0,n-2} \\ 0 & a_{1,1} & \dots & a_{1,n-2} \\ \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \dots & a_{n-2,n-2} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-2} \end{pmatrix} = \begin{pmatrix} b_0 - a_{0,n-1}x_{n-1} \\ b_1 - a_{1,n-1}x_{n-1} \\ \vdots \\ b_{n-2} - a_{n-2,n-1}x_{n-1} \end{pmatrix} = \begin{pmatrix} b'_0 \\ b'_1 \\ \vdots \\ b'_{n-2} \end{pmatrix}$$

De la même façon que précédemment nous calculons,

$$x_{n-2} = \frac{b'_{n-2}}{a_{n-2,n-2}}, \text{ pour } a_{n-2,n-2} \neq 0$$

Cette approche récursive demande à ce que les termes de la diagonale soient non nuls.

# Résolution d'un système linéaire

## Cas d'une matrice triangulaire (3)

ResTriangle( $A, B, n$ )

**Entrées**  $A$  une matrice triangulaire supérieure de dimension  $n \times n$

$B$  un vecteur de taille  $n$

$C_{n-1}$  le vecteur dernière colonne de  $A$

**Sortie** un vecteur  $X$  solution du système

**Corps** Si  $n = 1$  alors  $X = B/A$

Sinon

$$x \leftarrow \frac{b_{n-1}}{a_{n-1, n-1}}$$

$$A' \leftarrow \text{RedMat}(A, n-1)$$

$$B' \leftarrow B - x \cdot C_{n-1}$$

$$X \leftarrow (\text{ResTriangle}(A', B', n-1), x)$$

# Résolution d'un système linéaire

## Cas d'une matrice triangulaire (4)

$$\begin{bmatrix} 10 & 7 & 8 & 7 \\ 0 & 5 & 13 & 5 \\ 0 & 0 & 20 & 13 \\ 0 & 0 & 0 & 6 \end{bmatrix}, \begin{bmatrix} 3 \\ 2 \\ 3 \\ 3 \end{bmatrix} \rightarrow \begin{bmatrix} 10 & 7 & 8 \\ 0 & 5 & 13 \\ 0 & 0 & 20 \end{bmatrix}, \begin{bmatrix} -1/2 \\ -1/2 \\ -7/2 \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} 10 & 7 \\ 0 & 5 \end{bmatrix}, \begin{bmatrix} \frac{9}{10} \\ \frac{71}{40} \end{bmatrix} \rightarrow \left[ -\frac{317}{200} \right]$$

$$\left[ -\frac{317}{2000}, \frac{71}{200}, -\frac{7}{40}, 1/2 \right]$$

# Résolution d'un système linéaire

## Cas d'une matrice triangulaire (5) Complexité

- ▶ Le nombre de produits pour un système de  $n$  équation est noté  $T(n)$
- ▶ Nous avons la formule de récurrence :  $T(n) = T(n - 1) + n$  et  $T(1) = 1$
- ▶ Donc le nombre total de produits est  $\frac{n^2+n}{2}$

# Résolution d'un système linéaire

## Cas d'une matrice triangulaire (6), version itérative

ResTriangleIt( $A, B, n$ )

**Entrées**  $A$  une matrice triangulaire supérieure  $n \times n$   
 $B$  un vecteur de taille  $n$

**Sortie** un vecteur  $B$  solution du système et  $A$  élément neutre.

**Corps** Pour  $i = n-1$  à  $0$  faire

$$b_i \leftarrow \frac{b_i}{a_{i,i}}$$

Si ( $i > 0$ ) alors

Pour  $j = i - 1$  à  $0$  faire

$$b_j \leftarrow b_j - b_i * a_{j,i}$$

$$a_{j,i} \leftarrow 0$$

$$a_{i,i} \leftarrow 1$$

# Résolution d'un système linéaire

## Cas d'une matrice triangulaire (6), version itérative

$$\begin{aligned}
 & \begin{bmatrix} 10 & 3 & 8 & 7 \\ 0 & 5 & 3 & 5 \\ 0 & 0 & 2 & 3 \\ 0 & 0 & 0 & 6 \end{bmatrix}, \begin{bmatrix} 3 \\ 2 \\ 3 \\ 3 \end{bmatrix} \rightarrow \begin{bmatrix} 10 & 3 & 8 & 0 \\ 0 & 5 & 3 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} -1/2 \\ -1/2 \\ 3/2 \\ 1/2 \end{bmatrix} \\
 & \rightarrow \begin{bmatrix} 10 & 3 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} -13/2 \\ -11/4 \\ 3/4 \\ 1/2 \end{bmatrix} \rightarrow \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} -\frac{97}{20} \\ -\frac{11}{20} \\ 3/4 \\ 1/2 \end{bmatrix} \\
 & \rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} -\frac{97}{200} \\ -\frac{11}{20} \\ 3/4 \\ 1/2 \end{bmatrix}
 \end{aligned}$$



# Triangularisation du système

## Méthode de Gauss

- ▶ L'idée est de faire une combinaison linéaire des lignes de notre système pour arriver à une matrice triangulaire.
- ▶ Une fois la matrice triangulaire obtenue, nous appliquons la résolution précédente.
- ▶ Par contre ici, l'importance du choix de la ligne pivot influe sur la qualité, et peut donc demander une réorganisation des lignes.

## Triangularisation du système

### Algorithme de Gauss naïf sans réorganisation (1)

NaifGauss( $A, B, n$ )

**Entrée**  $A$  une matrice et un vecteur  $B$

**Sortie**  $A$  triangularisée et  $B$  modifiée

**Pour**  $i = 0$  à  $n - 1$  **faire**

**Pour**  $j = i + 1$  à  $n - 1$  **faire**

$$B_j := B_j - (A_{j,i}/A_{i,i}) * B_i;$$

$$A_{j,i..n-1} := A_{j,i..n-1} - (A_{j,i}/A_{i,i}) * A_{i,i..n-1};$$

**retourne**  $A$  et  $B$

Complexité de l'ordre de  $\frac{n(n-1)(2n-1)}{6} + \frac{n(n-1)}{2}$

# Triangularisation du système

## Algorithme de Gauss naïf sans réorganisation (2)

$$\begin{bmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{bmatrix}, \begin{bmatrix} 32 \\ 23 \\ 33 \\ 31 \end{bmatrix}$$

$$\begin{bmatrix} 10 & 7 & 8 & 7 \\ 0 & 1/10 & 2/5 & 1/10 \\ 0 & 2/5 & \frac{18}{5} & \frac{17}{5} \\ 0 & 1/10 & \frac{17}{5} & \frac{51}{10} \end{bmatrix}, \begin{bmatrix} 32 \\ 3/5 \\ \frac{37}{5} \\ \frac{43}{5} \end{bmatrix} \rightarrow \begin{bmatrix} 10 & 7 & 8 & 7 \\ 0 & 1/10 & 2/5 & 1/10 \\ 0 & 0 & 2 & 3 \\ 0 & 0 & 3 & 5 \end{bmatrix}, \begin{bmatrix} 32 \\ 3/5 \\ 5 \\ 8 \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} 10 & 7 & 8 & 7 \\ 0 & 1/10 & 2/5 & 1/10 \\ 0 & 0 & 2 & 3 \\ 0 & 0 & 0 & 1/2 \end{bmatrix}, \begin{bmatrix} 32 \\ 3/5 \\ 5 \\ 1/2 \end{bmatrix} \rightarrow \begin{bmatrix} 10 & 7 & 8 & 7 \\ 0 & 1/10 & 2/5 & 1/10 \\ 0 & 0 & 2 & 3 \\ 0 & 0 & 0 & 1/2 \end{bmatrix}, \begin{bmatrix} 32 \\ 3/5 \\ 5 \\ 1/2 \end{bmatrix}$$

# Triangularisation du système

## Algorithme de Gauss naïf sans réorganisation (3) Resolution

$$\begin{bmatrix} 10 & 7 & 8 & 0 \\ 0 & 1/10 & 2/5 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 25 \\ 1/2 \\ 2 \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} 10 & 7 & 0 & 0 \\ 0 & 1/10 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 17 \\ 1/10 \\ 1 \\ 1 \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 10 \\ 1 \\ 1 \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

## Triangularisation du système

### Algorithme de Gauss naïf sans réorganisation (4) Remarques

- ▶ L'algorithme de triangularisation est sans effet sur une matrice triangulaire supérieure.
- ▶ Il ne fonctionne qu'à la condition que le pivot  $A_{i,j}$  soit non nul. Il faut donc le vérifier au cours de l'algorithme.
  - ⇒ Pour ceci on échange avec une ligne non encore prise en compte comme pivot, de façon à avoir un pivot non nul.
  - ⇒ Si cette opération n'est pas possible alors le système n'a pas de solution unique.

# Triangularisation du système

## Algorithme de Gauss avec choix du premier non nul

### PremGauss( $A, B, n$ )

**Entrée**  $A$  une matrice et un vecteur  $B$

**Sortie**  $A$  triangularisée et  $B$  modifiée

**Pour**  $i = 0$  à  $n - 1$  faire

**Si**  $A_{i,i} = 0$  et  $i < n$  alors  $j = i + 1$ ,

**Tant que**  $A_{j,i} = 0$  et  $j < n - 1$  faire  $j = j + 1$

**Si**  $j = n - 1$  et  $A_{j,i} = 0$  alors Pas de solution

**Sinon**  $T_{i..n-1} := A_{i..n-1}; A_{i..n-1} := A_{j..n-1}; A_{j..n-1} := T_{i..n-1}$

$X := B_i; B_j := B_j; B_j := X$

**Pour**  $j = i + 1$  à  $n - 1$  faire

$B_j := B_j - (A_{j,i}/A_{i,i}) * B_i;$

$A_{j,i..n-1} := A_{j,i..n-1} - (A_{j,i}/A_{i,i}) * A_{i,i..n-1};$

**retourne**  $A$  et  $B$

## Triangularisation du système

### Algorithme de Gauss avec choix du premier non nul

- ▶ Cette approche évite donc les division par zéro, avec un contrôle en  $O(n)$  pour chaque recherche.
- ▶ Par contre ce n'est pas la meilleure stratégie quant à la qualité du résultat.
- ▶ En fait, il est préférable de choisir le plus grand pivot possible à chaque pas. Pour ceci deux stratégies :
  - ▶ Partielle, en cherchant dans la colonne du pivot en cours la plus grande valeur puis en permutant les lignes. Cette approche reste en  $O(n)$  pour chaque recherche.
  - ▶ Globale, en cherchant la plus grande valeur dans la sous-matrice à traiter, puis en permutant colonnes puis ligne. Cette approche passe en  $O(n^2)$  et s'avère trop coûteuse pour un résultat pas meilleur.

# Triangularisation du système

## Algorithme de Gauss avec choix sur la colonne (1)

### PartielGauss( $A, , B, n$ )

**Entrée**  $A$  une matrice et un vecteur  $B$

**Sortie**  $A$  triangularisée et  $B$  modifiée

**Pour**  $i = 0$  à  $n - 1$  **faire**

$max := |A[i, i]|; k := i;$

**if**  $i < n - 1$  **then for**  $j$  **from**  $i + 1$  **to**  $n - 1$  **do**

$\text{if } (max < |A[j, i]|) \text{ then } k := j; max := |A[j, i]|;$

**if**  $(max = 0)$  **then** Pas de solution;

**else**

**for**  $j$  **from**  $i$  **to**  $n - 1$  **do**  $T := A[i, j]; A[i, j] := A[k, j]; A[k, j] := T;$

$X := B[i]; B[i] := B[k]; B[k] := X;$

**else if**  $(A[n - 1, n - 1] = 0)$  **then** Pas de solution;

**Pour**  $j = i + 1$  à  $n - 1$  **faire**

$B_j := B_j - (A_{j,i}/A_{i,i}) * B_i;$

$A_{j,i..n-1} := A_{j,i..n-1} - (A_{j,i}/A_{i,i}) * A_{i,i..n-1};$

**retourne**  $A$  et  $B$



# Triangularisation du système

## Algorithme de Gauss avec choix sur la colonne(2)

$$\begin{bmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{bmatrix}, \begin{bmatrix} 32 \\ 23 \\ 33 \\ 31 \end{bmatrix} \rightarrow \begin{bmatrix} 10 & 7 & 8 & 7 \\ 0 & 1/10 & 2/5 & 1/10 \\ 0 & 2/5 & \frac{18}{5} & \frac{17}{5} \\ 0 & 1/10 & \frac{17}{5} & \frac{51}{10} \end{bmatrix}, \begin{bmatrix} 32 \\ 3/5 \\ \frac{37}{5} \\ \frac{43}{5} \end{bmatrix}$$

$$\begin{bmatrix} 10 & 7 & 8 & 7 \\ 0 & 2/5 & \frac{18}{5} & \frac{17}{5} \\ 0 & 1/10 & 2/5 & 1/10 \\ 0 & 1/10 & \frac{17}{5} & \frac{51}{10} \end{bmatrix}, \begin{bmatrix} 32 \\ \frac{37}{5} \\ 3/5 \\ \frac{43}{5} \end{bmatrix} \rightarrow \begin{bmatrix} 10 & 7 & 8 & 7 \\ 0 & 2/5 & \frac{18}{5} & \frac{17}{5} \\ 0 & 0 & -1/2 & -3/4 \\ 0 & 0 & 5/2 & \frac{17}{4} \end{bmatrix}, \begin{bmatrix} 32 \\ \frac{37}{5} \\ -5/4 \\ \frac{27}{4} \end{bmatrix}$$

$$\begin{bmatrix} 10 & 7 & 8 & 7 \\ 0 & 2/5 & \frac{18}{5} & \frac{17}{5} \\ 0 & 0 & 5/2 & \frac{17}{4} \\ 0 & 0 & -1/2 & -3/4 \end{bmatrix}, \begin{bmatrix} 32 \\ \frac{37}{5} \\ \frac{27}{4} \\ -5/4 \end{bmatrix} \rightarrow \begin{bmatrix} 10 & 7 & 8 & 7 \\ 0 & 2/5 & \frac{18}{5} & \frac{17}{5} \\ 0 & 0 & 5/2 & \frac{17}{4} \\ 0 & 0 & 0 & 1/10 \end{bmatrix}, \begin{bmatrix} 32 \\ \frac{37}{5} \\ \frac{27}{4} \\ 1/10 \end{bmatrix}$$

# Résolution d'un système

## Conditionnement <sup>5</sup>

Système initial :

$$A := \begin{bmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{bmatrix}, B := \begin{bmatrix} 32 \\ 23 \\ 33 \\ 31 \end{bmatrix}$$

Réduction de Gauss :

$$A := \begin{bmatrix} 10 & 7 & 8 & 7 \\ 0 & 2/5 & \frac{18}{5} & \frac{17}{5} \\ 0 & 0 & 5/2 & \frac{17}{4} \\ 0 & 0 & 0 & 1/10 \end{bmatrix}, B := \begin{bmatrix} 32 \\ \frac{37}{5} \\ \frac{27}{4} \\ 1/10 \end{bmatrix}$$

Résolution du système:

$$A := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, B := \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

<sup>5</sup>(voir PG Ciarlet : Introduction à l'analyse numérique matricielle et à l'optimisation)

# Résolution d'un système

## Conditionnement (2)

Système initial :

$$A := \begin{bmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{bmatrix}, B := \begin{bmatrix} 32.1 \\ 22.9 \\ 33.1 \\ 30.9 \end{bmatrix}$$

Réduction de Gauss :

$$A := \begin{bmatrix} 10 & 7 & 8 & 7 \\ 0 & 2/5 & \frac{18}{5} & \frac{17}{5} \\ 0 & 0 & 5/2 & \frac{17}{4} \\ 0 & 0 & 0 & 1/10 \end{bmatrix}, B := \begin{bmatrix} 32.1 \\ 7.42 \\ 6.575 \\ -0.11 \end{bmatrix}$$

Résolution du système:

$$A := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, B := \begin{bmatrix} 9.200000000 \\ -12.600000000 \\ 4.500000000 \\ -1.100000000 \end{bmatrix}$$

# Résolution d'un système

## Conditionnement (2)

- ▶ Nous constatons qu'une moindre variation sur les entrées peuvent avoir de lourdes conséquences sur les sorties.
- ▶ Cela est très dépendant du système. Le conditionnement se mesure :

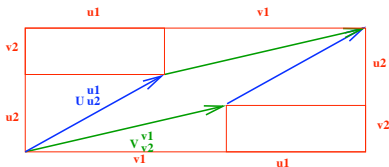
$$\text{cond}(A) = \|A\| \cdot \|A^{-1}\| \text{ avec } \|A\| = \sup_{\|v\|=1} \|Av\|$$

- ▶ Un système est bien conditionné si  $\text{cond}(A)$  est petit.
- ▶ Le passage des rationnels aux flottants peut être catastrophique. Par contre en flottants les calculs seront beaucoup plus rapides.

# Algorithmes de calcul de déterminants et produits vectoriels

# Déterminant et surface

## Interprétation géométrique

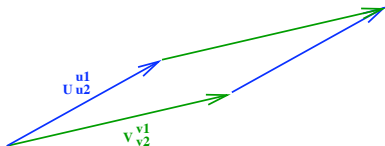


- ▶ Deux vecteurs en dimension 2  $V \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$  et  $U \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$
- ▶ Surface définie par ces deux vecteurs :  $S = |v_1 u_2 - v_2 u_1|$
- ▶ Surface signée :

$$\det(V, U) = \begin{vmatrix} v_1 & u_1 \\ v_2 & u_2 \end{vmatrix} = v_1 u_2 - v_2 u_1$$

# Déterminant et surface

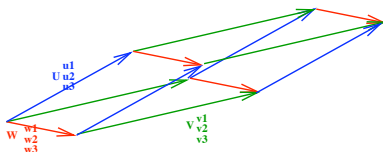
## Interprétation trigonométrique



- ▶ Le signe donne l'orientation : Positive = sens trigonométrique, Négative = sens inverse.
- ▶ Nous avons  $\det(V, U) = \|V\| \cdot \|U\| \cdot \sin(V, U)$
- ▶ **Propriété** :  $\det(V, U) = 0 \Leftrightarrow U$  et  $V$  colinéaires

# Déterminant et volume

## interprétation géométrique



- ▶ Trois vecteurs  $V \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}$  et  $U \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}$  et  $W \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix}$

- ▶ Volume signé :

$$\det(V, U, W) = \begin{vmatrix} v_1 & u_1 & w_1 \\ v_2 & u_2 & w_2 \\ v_3 & u_3 & w_3 \end{vmatrix} = v_1 \cdot \begin{vmatrix} u_2 & w_2 \\ u_3 & w_3 \end{vmatrix} - v_2 \cdot \begin{vmatrix} u_1 & w_1 \\ u_3 & w_3 \end{vmatrix} + v_3 \cdot \begin{vmatrix} u_1 & w_1 \\ u_2 & w_2 \end{vmatrix}$$

- ▶ Le signe donne l'orientation dans une base orthonormale directe :  
Positive = sens direct , Négative = sens indirect.



## Déterminant de $n$ vecteurs

$E$  espace vectoriel de dimension  $n$

- ▶ Le **déterminant de  $n$  vecteurs**  $(V_1, \dots, V_n)$  peut être définie comme une **forme  $n$ -linéaire alternée**  $\det$  de  $E^n$  dans  $K$  telle

$$\text{que : } \det\left(\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 1 \end{pmatrix}\right) = 1$$

- ▶ **Forme  $n$ -linéaire :**

$$\det(V_1, \dots, aV_i + bU_i, \dots, V_n) = a \cdot \det(V_1, \dots, V_i, \dots, V_n) + b \cdot \det(V_1, \dots, U_i, \dots, V_n)$$

- ▶ **Alternée :**

$$\det(V_1, \dots, V_i, \dots, V_j, \dots, V_n) + \det(V_1, \dots, V_j, \dots, V_i, \dots, V_n) = 0$$

$$\text{D'où, } \det(V_1, \dots, V_i, \dots, V_i, \dots, V_n) = 0$$

- ▶ **Déterminant d'une matrice :**  $\det(A)$ , c'est le déterminant des vecteurs composant la matrice.

## Petits rappels

### Identité et transposée

- ▶ **Identité** : la matrice  $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 1 \end{pmatrix}$  représente l'application identité pour le produit matrice vecteur. Elle est aussi l'élément neutre pour le produit de matrices  $n \times n$ . Elle est composée des vecteurs de la base canonique.
- ▶ La **transposée** d'une matrice est la matrice dont les lignes sont les colonnes de l'autre :

$$\begin{pmatrix} a_{0,0} & a_{0,1} & \dots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \dots & a_{1,n-1} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n-1,0} & a_{n-1,1} & \dots & a_{n-1,n-1} \end{pmatrix}^t = \begin{pmatrix} a_{0,0} & a_{1,0} & \dots & a_{n-1,0} \\ a_{0,1} & a_{1,1} & \dots & a_{n-1,1} \\ \vdots & \vdots & \vdots & \vdots \\ a_{0,n-1} & a_{1,n-1} & \dots & a_{n-1,n-1} \end{pmatrix}$$

# Déterminant de $n$ vecteurs

## Propriétés

- ▶ D'un produit :  $\det(A \times B) = \det(A) \times \det(B)$
- ▶ Produit par un scalaire :  $\det(\lambda A) = \lambda^n \det(A)$
- ▶ Transposée :  $\det(A) = \det(A^t)$

- ▶ Calcul : 
$$\det(A) = \sum_{\sigma \in \mathcal{S}_n} \epsilon(\sigma) \prod_{i=0}^{n-1} a_{\sigma(i),i}$$

où  $\sigma \in \mathcal{S}_n$  représente les **permutations** de  $n$  éléments et  $\epsilon(\sigma)$  la **signature** (parité de la décomposition en **transpositions** : échange de deux éléments):  $\epsilon(\sigma) = 1$  si paire,  $-1$  sinon.

Le nombre de permutations de  $n$  éléments est  $n!$  ( $n$  choix pour le premier,  $n - 1$  pour le suivant, etc)

- ▶ Cette propriété est issue de la décomposition des vecteurs formant la matrice dans la base canonique.

## Calcul du déterminant

### Un important théorème

- Soit une matrice  $M$  de la forme :

$$M = \begin{pmatrix} a_{0,0} & a_{0,1} & \dots & a_{0,n-1} \\ 0 & a_{1,1} & \dots & a_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n-1,1} & \dots & a_{n-1,n-1} \end{pmatrix}$$

$$\text{Alors, } \det(M) = a_{0,0} \times \det \left( \begin{pmatrix} a_{1,1} & \dots & a_{1,n-1} \\ \vdots & \ddots & \vdots \\ a_{n-1,1} & \dots & a_{n-1,n-1} \end{pmatrix} \right)$$

- *En fait ce résultat provient directement de la formule de calcul avec les permutations.*

## Calcul du déterminant

### Notion de mineurs et de cofacteurs: propriété

- ▶ On définit le **mineur**  $A_{i,j}$  d'une matrice  $A$  de dimension  $n \times n$  comme la matrice de dimension  $(n-1) \times (n-1)$  issue de la matrice  $A$  privée de sa  $i^{\text{ème}}$  ligne et de sa  $j^{\text{ème}}$  colonne.
- ▶ On déduit du théorème précédent que :

$$\det(A) = \sum_{i=0}^{n-1} (-1)^{i+k} a_{i,k} \det(A_{i,k}) = \sum_{j=0}^{n-1} (-1)^{j+k} a_{k,j} \det(A_{k,j})$$

- ▶  $(-1)^{i+j} \det(A_{i,j})$  est appelé le **cofacteur** de  $a_{i,j}$

## Calcul du déterminant

### Algorithme via les cofacteurs

DetM( $A, n$ ) (*à partir de la première colonne*)

**Entrées** une matrice  $A$  de dimension  $n \times n$

**Sortie**  $det = \det(A)$

**Corps** **if** ( $n = 1$ ) **then**  $det := a_{0,0}$

**else**

$det := 0;$

**for**  $i = 0$  **to**  $n - 1$  **do**

$B := \text{Mineur}(A, i, 0, n);$

$det := det + (-1)^{i \bmod 2} * a_{i,0} * \text{DetM}(B, n - 1)$

## Calcul du déterminant

### Complexité de l'algorithme via les cofacteurs

- ▶ On note  $TDM(n)$  le nombre de produits pour le calcul du déterminant d'une matrice de taille  $n$ .
- ▶ On a,  $TDM(n) = n * TDM(n - 1) + n$
- ▶ On en déduit que  $TDM(n) = O(n!)$
- ▶ La complexité est identique à celle via les permutations

# Calcul du déterminant

## Notion de cofacteurs : Exemple

$$\det \begin{pmatrix} 8 & 6 & 1 & 1 \\ 7 & 1 & 10 & 5 \\ 10 & 7 & 3 & 7 \\ 7 & 5 & 3 & 1 \end{pmatrix} = 8 \det \begin{pmatrix} 1 & 10 & 5 \\ 7 & 3 & 7 \\ 5 & 3 & 1 \end{pmatrix} - 7 \det \begin{pmatrix} 6 & 1 & 1 \\ 7 & 3 & 7 \\ 5 & 3 & 1 \end{pmatrix} \quad (1)$$

$$+ 10 \det \begin{pmatrix} 6 & 1 & 1 \\ 1 & 10 & 5 \\ 5 & 3 & 1 \end{pmatrix} - 7 \det \begin{pmatrix} 6 & 1 & 1 \\ 1 & 10 & 5 \\ 7 & 3 & 7 \end{pmatrix} \quad (2)$$

$$= 287 \quad (3)$$



# Calcul du déterminant

## Cas des matrices triangulaires

- Considérons la **matrice triangulaire supérieure** :

$$A = \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & & a_{0,n-1} \\ 0 & a_{1,1} & a_{1,2} & & a_{1,n-1} \\ 0 & 0 & a_{2,2} & & a_{2,n-1} \\ \vdots & & \vdots & & \vdots \\ \vdots & & \vdots & & \vdots \\ 0 & 0 & 0 & \dots & a_{n-1,n-1} \end{pmatrix}$$

- L'algorithme précédent nous donne :  $\det(A) = \prod_{i=0}^{n-1} a_{i,i}$
- Remarque : le *déterminant est stable pour la combinaison linéaire dans Gauss et change de signe à chaque transposition*, nous pouvons utiliser l'algorithme de Gauss pour le calcul du déterminant.

## Calcul du déterminant

### Cas des matrices triangulaires : exemple

$$\det \begin{pmatrix} 8 & 6 & 1 & 1 \\ 0 & 1 & 10 & 5 \\ 0 & 0 & 3 & 7 \\ 0 & 0 & 0 & 1 \end{pmatrix} = 8 \det \begin{pmatrix} 1 & 10 & 5 \\ 0 & 3 & 7 \\ 0 & 0 & 1 \end{pmatrix} \quad (4)$$

$$= 8 \times 1 \times \det \begin{pmatrix} 3 & 7 \\ 0 & 1 \end{pmatrix} \quad (5)$$

$$= 8 \times 1 \times 3 \times \det \begin{pmatrix} 1 \end{pmatrix} \quad (6)$$

$$= 8 \times 1 \times 3 \times 1 \quad (7)$$

$$= 24 \quad (8)$$

# Calcul du déterminant

## Adaptation de l'algorithme de Gauss

### DetGauss( $A, n$ )

Entrée  $A$  une matrice

Sortie  $det = \det(A)$

$signe := 0$

**For**  $i = 1$  **to**  $n$  **do**

$max := |A[i, i]|; k := i;$

**if**  $i < n$  **then for**  $j = i + 1$  **to**  $n$  **do**

**if**  $(max < |A[j, i]|)$  **then**  $k := j; max := |A[j, i]|;$

**if**  $(max = 0)$  **then**  $det := 0$

**else**

**for**  $j = i$  **to**  $n$  **do**  $T := A[i, j]; A[i, j] := A[k, j]; A[k, j] := T;$

**if**  $k \neq i$  **then**  $signe := (1 + signe) \bmod 2;$

**else if**  $(A[n, n] = 0)$  **then**  $det := 0$

**For**  $j = i + 1$  **to**  $n$  **do**

$A_{j,i..n+1} := A_{j,i..n+1} - (A_{j,i}/A_{i,i}) * A_{i,i..n+1};$

$det := 1;$

**for**  $i = 1$  **to**  $n$  **do**  $det := det * A_{i,i};$

$det := det * (-1)^{signe}$

# Calcul du déterminant

## Adaptation de l'algorithme de Gauss : Exemple

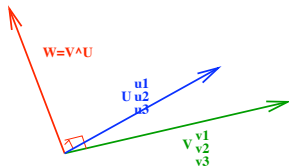
$$\begin{bmatrix} 2 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{bmatrix}, 0 \rightarrow \begin{bmatrix} 8 & 6 & 10 & 9 \\ 0 & -1/4 & -11/4 & -\frac{23}{8} \\ 0 & 11/2 & 11/2 & \frac{19}{4} \\ 0 & -1/4 & 1/4 & \frac{17}{8} \end{bmatrix}, 1$$

$$\rightarrow \begin{bmatrix} 8 & 6 & 10 & 9 \\ 0 & 11/2 & 11/2 & \frac{19}{4} \\ 0 & 0 & -5/2 & -\frac{117}{44} \\ 0 & 0 & 1/2 & \frac{103}{44} \end{bmatrix}, 0 \rightarrow \begin{bmatrix} 8 & 6 & 10 & 9 \\ 0 & 11/2 & 11/2 & \frac{19}{4} \\ 0 & 0 & -5/2 & -\frac{117}{44} \\ 0 & 0 & 0 & \frac{199}{110} \end{bmatrix}, 0$$

$$\rightarrow \begin{bmatrix} 8 & 6 & 10 & 9 \\ 0 & 11/2 & 11/2 & \frac{19}{4} \\ 0 & 0 & -5/2 & -\frac{117}{44} \\ 0 & 0 & 0 & \frac{199}{110} \end{bmatrix}, 0 \rightarrow \det := -199$$

# Produit vectoriel

## En dimension 3



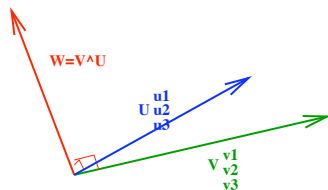
- ▶ Soient deux vecteurs  $V$  et  $U$  dans une base orthonormale directe: Le **produit vectoriel** de ces vecteurs, est le vecteur

$W = V \wedge U$  tel que :

$$w_1 = \det \begin{pmatrix} v_2 & u_2 \\ v_3 & u_3 \end{pmatrix}, w_2 = -\det \begin{pmatrix} v_1 & u_1 \\ v_3 & u_3 \end{pmatrix}, w_3 = \det \begin{pmatrix} v_1 & u_1 \\ v_2 & u_2 \end{pmatrix}$$

# Produit vectoriel

## En dimension 3 : Propriétés



- ▶  $V \wedge U = -U \wedge V$
- ▶ **Produit mixte** :  $[V, U, W] = V \cdot (U \wedge W) = \det(V, U, W)$

## Produit vectoriel

En dimension  $n$

- ▶ Soient  $n - 1$  vecteurs  $V_i$  dans une base orthonormale directe en dimension  $n$ : Le **produit vectoriel** de ces vecteurs, est le vecteur  $W = \bigwedge_{i=1}^{n-1} V_i$  tel que :

$$w_k = (-1)^k \Delta_k$$

pour  $k = 0..n - 1$ , où  $\Delta_k$  représente le déterminant de la matrice  $(n - 1) \times (n - 1)$  formée par les vecteurs  $V_i$  privée de la  $k^{ieme}$  ligne.

- ▶ Autrement dit si on considère la matrice  $A = (V_0, V_1, \dots, V_{n-1})$ , les coordonnées du produit vectoriel de  $(V_1, \dots, V_{n-1})$  sont obtenus via les **cofacteurs**  $(-1)^k A_{k,0}$  de la matrice  $A$ .

# Décomposition LU - techniques de programmation



## Résolution de $A \times X = B$

$A, X, B$  trois matrices

Nous désirons résoudre le système :

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{pmatrix} \times \begin{pmatrix} x_{0,0} & x_{0,1} & x_{0,2} \\ x_{1,0} & x_{1,1} & x_{1,2} \\ x_{2,0} & x_{2,1} & x_{2,2} \end{pmatrix} = \begin{pmatrix} b_{0,0} & b_{0,1} & b_{0,2} \\ b_{1,0} & b_{1,1} & b_{1,2} \\ b_{2,0} & b_{2,1} & b_{2,2} \end{pmatrix}$$

Ce qui revient à résoudre avec la méthode de Gauss les systèmes :

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{pmatrix} \times \begin{pmatrix} x_{0,i} \\ x_{1,i} \\ x_{2,i} \end{pmatrix} = \begin{pmatrix} b_{0,i} \\ b_{1,i} \\ b_{2,i} \end{pmatrix} \text{ pour } i = 0, 1, 2$$

## Résolution de $A \times X = B$

$A, X, B$  trois matrices (2)

- ▶ Dans le cadre de matrice  $n \times n$ , l'utilisation de la réduction de Gauss demande

Triangularisation :  $\frac{n(n-1)(2n-1)}{6}$  opérations

Actualisation de  $B$  :  $\frac{n(n-1)}{2}$  opérations

Résolution remontante :  $\frac{n^2 + n}{2}$  opérations

Si  $n$  matrices  $n \times n$  : un total de l'ordre de  $\frac{n^4}{3}$  opérations

- ▶ Le problème de cette approche est que lors de la triangularisation de la matrice  $A$  le vecteur  $B_j$  doit être actualisé.

## Résolution de $A \times X = B$

$A, X, B$  trois matrices (3)

Prenons un exemple,

$$A := \begin{bmatrix} 10 & 7 & 8 \\ 7 & 5 & 6 \\ 8 & 6 & 10 \end{bmatrix} \quad B := \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \rightarrow \begin{bmatrix} 10 & 7 & 8 \\ 0 & 1/10 & 2/5 \\ 0 & 0 & 2 \end{bmatrix}, \quad \begin{bmatrix} 1 \\ 13/10 \\ -3 \end{bmatrix}$$

$$A := \begin{bmatrix} 10 & 7 & 8 \\ 7 & 5 & 6 \\ 8 & 6 & 10 \end{bmatrix} \quad B := \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} 10 & 7 & 8 \\ 0 & 1/10 & 2/5 \\ 0 & 0 & 2 \end{bmatrix}, \quad \begin{bmatrix} 2 \\ 3/5 \\ -3 \end{bmatrix}$$

$$A := \begin{bmatrix} 10 & 7 & 8 \\ 7 & 5 & 6 \\ 8 & 6 & 10 \end{bmatrix} \quad B := \begin{bmatrix} 5 \\ 1 \\ 5 \end{bmatrix} \rightarrow \begin{bmatrix} 10 & 7 & 8 \\ 0 & 1/10 & 2/5 \\ 0 & 0 & 2 \end{bmatrix}, \quad \begin{bmatrix} 5 \\ -5/2 \\ 11 \end{bmatrix}$$

Les trois matrices triangulaires sont identiques, mais les vecteurs  $B_j$  ont été actualisés

# Décomposition L U

## Approche de la méthode

- ▶ Nous désirons effectuer la triangularisation une seule fois en conservant l'actualisation.
- ▶ Nous remarquons que la modification faite peut se traduire matriciellement :

$$\begin{aligned}
 & \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{pmatrix} \\
 &= \begin{pmatrix} 1 & 0 & 0 \\ \frac{a_{1,0}}{a_{0,0}} & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ 0 & a_{1,1} - \frac{a_{1,0}}{a_{0,0}} a_{0,1} & a_{1,2} - \frac{a_{1,0}}{a_{0,0}} a_{0,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{pmatrix} \\
 &= \begin{pmatrix} 1 & 0 & 0 \\ \frac{a_{1,0}}{a_{0,0}} & 1 & 0 \\ \frac{a_{2,0}}{a_{0,0}} & 0 & 1 \end{pmatrix} \times \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ 0 & a_{1,1} - \frac{a_{1,0}}{a_{0,0}} a_{0,1} & a_{1,2} - \frac{a_{1,0}}{a_{0,0}} a_{0,2} \\ 0 & a_{2,1} - \frac{a_{2,0}}{a_{0,0}} a_{0,1} & a_{2,2} - \frac{a_{2,0}}{a_{0,0}} a_{0,2} \end{pmatrix}
 \end{aligned}$$

# Décomposition L U

Approche de la méthode : exemple

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 10 & 7 & 8 \\ 7 & 5 & 6 \\ 8 & 6 & 10 \end{bmatrix} =, \begin{bmatrix} 10 & 7 & 8 \\ 7 & 5 & 6 \\ 8 & 6 & 10 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ \frac{7}{10} & 1 & 0 \\ 4/5 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 10 & 7 & 8 \\ 0 & 1/10 & 2/5 \\ 0 & 2/5 & \frac{18}{5} \end{bmatrix} = \begin{bmatrix} 10 & 7 & 8 \\ 7 & 5 & 6 \\ 8 & 6 & 10 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ \frac{7}{10} & 1 & 0 \\ 4/5 & 4 & 1 \end{bmatrix}, \begin{bmatrix} 10 & 7 & 8 \\ 0 & 1/10 & 2/5 \\ 0 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 10 & 7 & 8 \\ 7 & 5 & 6 \\ 8 & 6 & 10 \end{bmatrix}$$

# Décomposition L U

## Approche de la méthode : Résolution

**Décomposition :** Supposons qu'une matrice  $A$  se décompose sous la forme d'un produit :  $A = L \times U$  où  $L$  est une matrice triangulaire inférieure (Low) et  $U$  une matrice triangulaire supérieure (Up)

**Résolution**  $A \times X = B$  où  $A = L \times U$  est une matrice  $n \times n$ ,  $X$  et  $B$  deux vecteurs de dimension  $n$

1. Résolution de  $L \times Y = B$
2. Résolution de  $U \times X = Y$
3. Ainsi  $X$  est solution de  $L \times U \times X = B$   
autrement dit de  $A \times X = B$

# Décomposition L U

## Résolution : Exemple

Nous avons,

$$\begin{bmatrix} 10 & 7 & 8 \\ 7 & 5 & 6 \\ 8 & 6 & 10 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ \frac{7}{10} & 1 & 0 \\ 4/5 & 4 & 1 \end{bmatrix} \times \begin{bmatrix} 10 & 7 & 8 \\ 0 & 1/10 & 2/5 \\ 0 & 0 & 2 \end{bmatrix}$$

Ainsi, La résolution de

$$\begin{bmatrix} 10 & 7 & 8 \\ 7 & 5 & 6 \\ 8 & 6 & 10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 32 \\ 23 \\ 33 \end{bmatrix}$$

Se fait en deux étapes

# Décomposition L U

## Résolution : Exemple (2)

Première étape :

$$\begin{bmatrix} 1 & 0 & 0 \\ \frac{7}{10} & 1 & 0 \\ 4/5 & 4 & 1 \end{bmatrix} \times \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 32 \\ 23 \\ 33 \end{bmatrix} \rightarrow \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 32 \\ 3/5 \\ 5 \end{bmatrix}$$

Seconde étape :

$$\begin{bmatrix} 10 & 7 & 8 \\ 0 & 1/10 & 2/5 \\ 0 & 0 & 2 \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 32 \\ 3/5 \\ 5 \end{bmatrix} \rightarrow \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ -4 \\ 5/2 \end{bmatrix}$$



# Décomposition L U

## Résolution Descendante Triangulaire

ReducTriangL( $L, B, n$ )

**Entrées**  $L$  matrice triangulaire inférieure avec diagonale de 1,  
 $B$  vecteur,  $n$  la dimension.

**Sorties**  $L = Id$  et  $B$  solution du système

**Corps** **for**  $i = 0$  **to**  $n - 1$  **do**

**for**  $j = i + 1$  **to**  $n - 1$  **do**

$B[j] := B[j] - B[i] * L[j, i];$

$L[j, i] := 0;$

$L[i, i] := 1$

# Décomposition L U

## Résolution Montante Triangulaire

ReducTriangU( $U, Y, n$ )

**Entrées**  $U$  matrice triangulaire supérieure,  $Y$  vecteur,  $n$  la dimension.

**Sorties**  $L = Id$  et  $Y$  solution du système

**Corps** **for**  $i = n - 1$  **to**  $0$  **do**

$Y[i] := Y[i]/U[i,i];$

**for**  $j = i - 1$  **to**  $0$  **do**

$Y[j] := Y[j] - Y[i] * U[j, i];$

$U[j, i] := 0;$

$U[i, i] := 1$

# Décomposition L U

## Complexités des Résolutions Triangulaires

- ▶  $\text{ReducTriangL}(L, B, n)$  représente  $\frac{n^2-n}{2}$  opérations (produits/divisions)
- ▶  $\text{ReducTriangU}(U, Y, n)$  représente  $\frac{n^2+n}{2}$  opérations (produits/divisions)
- ▶ Donc la résolution d'un système de la forme  $L \times U \times X = B$  représente  $n^2$  opérations

# Décomposition L U

Approche de la méthode : retour

- Nous avons observé que:

$$\begin{aligned}
 & \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{pmatrix} \\
 &= \begin{pmatrix} 1 & 0 & 0 \\ \frac{a_{1,0}}{a_{0,0}} & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ 0 & a_{1,1} - \frac{a_{1,0}}{a_{0,0}} a_{0,1} & a_{1,2} - \frac{a_{1,0}}{a_{0,0}} a_{0,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{pmatrix} \\
 &= \begin{pmatrix} 1 & 0 & 0 \\ \frac{a_{1,0}}{a_{0,0}} & 1 & 0 \\ \frac{a_{2,0}}{a_{0,0}} & 0 & 1 \end{pmatrix} \times \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ 0 & a_{1,1} - \frac{a_{1,0}}{a_{0,0}} a_{0,1} & a_{1,2} - \frac{a_{1,0}}{a_{0,0}} a_{0,2} \\ 0 & a_{2,1} - \frac{a_{2,0}}{a_{0,0}} a_{0,1} & a_{2,2} - \frac{a_{2,0}}{a_{0,0}} a_{0,2} \end{pmatrix} \\
 &= \begin{pmatrix} 1 & 0 & 0 \\ \frac{a_{1,0}}{a_{0,0}} & 1 & 0 \\ \frac{a_{2,0}}{a_{0,0}} & 0 & 1 \end{pmatrix} \times \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ 0 & a'_{1,1} & a'_{1,2} \\ 0 & a'_{2,1} & a'_{2,2} \end{pmatrix} \\
 &= \begin{pmatrix} 1 & 0 & 0 \\ \frac{a_{1,0}}{a_{0,0}} & 1 & 0 \\ \frac{a_{2,0}}{a_{0,0}} & \frac{a'_{2,1}}{a'_{1,1}} & 1 \end{pmatrix} \times \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ 0 & a'_{1,1} & a'_{1,2} \\ 0 & 0 & a'_{2,2} - \frac{a'_{2,1}}{a'_{1,1}} a'_{1,2} \end{pmatrix}
 \end{aligned}$$

# Décomposition L U

Approche de la méthode : retour

- ▶ La soustraction d'un multiple d'un vecteur peut se traduire par la construction d'une matrice triangulaire inférieure
- ▶ Le coût de construction de cette matrice se réduit à des affectations.
- ▶ Nous avons ainsi dans le cadre de l'algorithme Naïf de Gauss une méthode de décomposition  $LU$

# Décomposition L U

## Algorithme Naïf

MethLU( $A, n$ )

**Entrées**  $A$  une matrice carrée de dimension  $n$

**Sorties**  $L$  et  $U$  deux matrices triangulaire  $L$  inférieure et  $U$  supérieure

**Corps**  $L :=$  Identité;  $U := A$ ;

**for**  $i = 0$  **to**  $n - 1$  **do**

**for**  $j = i + 1$  **to**  $n - 1$  **do**

$L[j, i] := (U[j, i] / U[i, i]);$

$U[j, i..n-1] := U[j, i..n-1] - L[j, i]^*$

$U[i, i..n-1];$

# Décomposition L U

## Algorithme Naïf : exemple

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 7 & 8 & 1 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 \\ 7/2 & 1 & 0 & 0 \\ 4 & 0 & 1 & 0 \\ 7/2 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 7 & 8 & 1 \\ 0 & -\frac{39}{2} & -22 & 3/2 \\ 0 & -22 & -22 & 5 \\ 0 & -\frac{39}{2} & -19 & 13/2 \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 \\ 7/2 & 1 & 0 & 0 \\ 4 & \frac{44}{39} & 1 & 0 \\ 7/2 & 1 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 7 & 8 & 1 \\ 0 & -\frac{39}{2} & -22 & 3/2 \\ 0 & 0 & \frac{110}{39} & \frac{43}{13} \\ 0 & 0 & 3 & 5 \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 \\ 7/2 & 1 & 0 & 0 \\ 4 & \frac{44}{39} & 1 & 0 \\ 7/2 & 1 & \frac{117}{110} & 1 \end{bmatrix}, \begin{bmatrix} 2 & 7 & 8 & 1 \\ 0 & -\frac{39}{2} & -22 & 3/2 \\ 0 & 0 & \frac{110}{39} & \frac{43}{13} \\ 0 & 0 & 0 & \frac{163}{110} \end{bmatrix}$$

## Décomposition L U

### Algorithme Naïf : Complexité

- ▶ Comme la construction de  $L$  est de coût négligeable (que des affectations), le coût de la construction  $LU$  est celui de la triangularisation de l'algorithme de Gauss :  $\frac{n(n-1)(2n-1)}{6}$  opérations.
- ▶ Ainsi si l'on veut résoudre  $A \times X = B_j$  pour  $j = 1, \dots, k$  alors on effectue une décomposition  $LU$  de  $A$  puis nous effectuons  $k$  résolutions triangulaires inférieures et supérieures, soit une complexité de  $\frac{n(n-1)(2n-1)}{6} + kn^2$  opérations.
- ▶ Pour revenir à notre problème de départ avec des matrices carrées de dimension  $n$ , cela revient à prendre  $k = n$ . Ainsi la complexité reste cubique de l'ordre de  $\frac{4n^3}{3}$



# Décomposition L U

## Avec recherche de Pivot

- ▶ L'approche naïve peut ne pas donner de solution si le pivot est nul, alors que le système peut être résolu.
- ▶ La méthode du pivot Gauss demande à échanger des lignes en cas de pivot nul.
- ▶ L'échange de ligne peut s'exprimer sous forme matricielle.

## Décomposition L U

### Echange de ligne

- ▶ Permuter deux lignes dans une matrice revient à multiplier cette matrice par la matrice identité où ces deux lignes ont été permutées.



$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{bmatrix} = \begin{bmatrix} 10 & 7 & 8 & 7 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 6 & 5 \\ 7 & 5 & 9 & 10 \end{bmatrix}$$

- ▶ Lors de l'exécution de l'algorithme de Gauss nous construisons une matrice de permutation comme composée de ces permutations de deux lignes.

# Décomposition L U

## Algorithme avec échange

PermutLU( $A, n$ )

**Entrées**  $A$  matrice carrée de dimension  $n$

**Sortie**  $P$  permutation,  $L$  et  $U$  telles que  $P \times A = L \times U$

**Corps**  $L := P := Id$  et  $U := A$

```

for  $i = 0$  to  $n - 1$  do  $max := abs(U[i, i]); k := i;$ 
    for  $j = i + 1$  to  $n - 1$  do
        if ( $max < abs(U[j, i])$ ) then  $k := j; max := abs(U[j, i]);$ 
    if ( $max \neq 0$ ) then
        for  $j = 0$  to  $n - 1$  do
             $T := U[i, j]; U[i, j] := U[k, j]; U[k, j] := T;$ 
             $T := P[i, j]; P[i, j] := P[k, j]; P[k, j] := T;$ 
        for  $j = 1$  to  $i - 1$  do
             $T := L[i, j]; L[i, j] := L[k, j]; L[k, j] := T;$ 
    for  $j = i + 1$  to  $n - 1$  do
         $L[j, i] := (U[j, i] / U[i, i]);$ 
         $U[j, i..n - 1] := U[j, i..n - 1] - (L[j, i]) * U[i, i..n - 1];$ 

```

# Décomposition L U

Algorithme avec échange : exemple (1)

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 7 & 8 & 1 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 7 & 8 & 1 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 7 & 8 & 1 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{7}{8} & 1 & 0 & 0 \\ 1/4 & 0 & 1 & 0 \\ \frac{7}{8} & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 8 & 6 & 10 & 9 \\ 0 & -1/4 & -11/4 & -\frac{23}{8} \\ 0 & 11/2 & 11/2 & -5/4 \\ 0 & -1/4 & 1/4 & \frac{17}{8} \end{bmatrix}$$

# Décomposition L U

Algorithme avec échange : exemple (2)

$$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 7 & 8 & 1 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1/4 & 1 & 0 & 0 \\ 7/8 & -1/22 & 1 & 0 \\ 7/8 & -1/22 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 8 & 6 & 10 & 9 \\ 0 & 11/2 & 11/2 & -5/4 \\ 0 & 0 & -5/2 & -\frac{129}{44} \\ 0 & 0 & 1/2 & \frac{91}{44} \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 7 & 8 & 1 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1/4 & 1 & 0 & 0 \\ 7/8 & -1/22 & 1 & 0 \\ 7/8 & -1/22 & -1/5 & 1 \end{bmatrix} \cdot \begin{bmatrix} 8 & 6 & 10 & 9 \\ 0 & \frac{11}{2} & \frac{11}{2} & -\frac{5}{4} \\ 0 & 0 & -\frac{5}{2} & -\frac{129}{44} \\ 0 & 0 & 0 & \frac{163}{110} \end{bmatrix}$$

# Décomposition L U

## Algorithme avec échange : Résolution

- ▶ Du fait que le résultat de la décomposition  $LU$  fait intervenir une permutation.  $P.A = L.U$
- ▶ Le système à résoudre devient  $P.A.X = P.B$
- ▶ On résout donc  $L.Y = P.B$
- ▶ Puis  $U.X = Y$

# Algorithmes pour l'inversion de matrice

## Sensibilisation aux erreurs d'arrondis

## Calcul du déterminant

### Notion de mineurs et de cofacteurs: Rappels

- ▶ Nous avons vu que :

$$\det(A) = \sum_{i=0}^{n-1} (-1)^{i+k} a_{i,k} \det(A_{i,k}) = \sum_{j=0}^{n-1} (-1)^{j+k} a_{k,j} \det(A_{k,j})$$

- ▶ Où le **mineur**  $A_{i,j}$  d'une matrice  $A$  de dimension  $n \times n$  est la matrice de dimension  $(n-1) \times (n-1)$  issue de la matrice  $A$  privée de sa  $i^{ieme}$  ligne et de sa  $j^{ieme}$  colonne.
- ▶  $(-1)^{i+j} \det(A_{i,j})$  est appelé le **cofacteur** de  $a_{i,j}$



## Calcul de l'inverse

Via les cofacteurs :

- Nous pouvons noter sous forme de produit scalaire l'expression du déterminant :

$$\det(A) = \left( \widehat{A}_{0,k} \quad \widehat{A}_{1,k} \quad \dots \quad \widehat{A}_{n-1,k} \right) \cdot \begin{pmatrix} a_{0,k} \\ a_{1,k} \\ \vdots \\ a_{n-1,k} \end{pmatrix}$$

$$\det(A) = \left( a_{k,0} \quad a_{k,1} \quad \dots \quad a_{k,n-1} \right) \cdot \begin{pmatrix} \widehat{A}_{k,0} \\ \widehat{A}_{k,1} \\ \vdots \\ \widehat{A}_{k,n-1} \end{pmatrix}$$

Avec  $\widehat{A}_{i,k} = (-1)^{i+k} \det(A_{i,k})$  le cofacteur.

## Calcul de l'inverse

Via les cofacteurs : (2)

- ▶ Nous remarquons, pour  $k' \neq k$ , que :  
(propriété des déterminants : forme bilinéaire alternée avec  $a_{i,k'} = a_{i,k}$ )

$$\left( \widehat{A_{0,k}} \quad \widehat{A_{1,k}} \quad \dots \quad \widehat{A_{n-1,k}} \right) \cdot \begin{pmatrix} a_{0,k'} \\ a_{1,k'} \\ \vdots \\ a_{n-1,k'} \end{pmatrix} = 0$$

## Calcul de l'inverse

Via les cofacteurs : (3)

► Nous obtenons ainsi :

$$\begin{aligned}
 & \begin{pmatrix} \widehat{A_{0,0}} & \widehat{A_{1,0}} & \dots & \widehat{A_{n-1,0}} \\ \widehat{A_{0,1}} & \widehat{A_{1,1}} & \dots & \widehat{A_{n-1,1}} \\ \vdots & \vdots & \dots & \vdots \\ \widehat{A_{0,n-1}} & \widehat{A_{1,n-1}} & \dots & \widehat{A_{n-1,n-1}} \end{pmatrix} \cdot \begin{pmatrix} a_{0,0} & a_{0,1} & \dots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \dots & a_{1,n-1} \\ \vdots & \vdots & \dots & \vdots \\ a_{n-1,0} & a_{n-1,1} & \dots & a_{n-1,n-1} \end{pmatrix} \\
 & = \begin{pmatrix} \det A & 0 & \dots & 0 \\ 0 & \det A & \dots & 0 \\ 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & \det A \end{pmatrix} \\
 & = \begin{pmatrix} a_{0,0} & a_{0,1} & \dots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \dots & a_{1,n-1} \\ \vdots & \vdots & \dots & \vdots \\ a_{n-1,0} & a_{n-1,1} & \dots & a_{n-1,n-1} \end{pmatrix} \cdot \begin{pmatrix} \widehat{A_{0,0}} & \widehat{A_{1,0}} & \dots & \widehat{A_{n-1,0}} \\ \widehat{A_{0,1}} & \widehat{A_{1,1}} & \dots & \widehat{A_{n-1,1}} \\ \vdots & \vdots & \dots & \vdots \\ \widehat{A_{0,n-1}} & \widehat{A_{1,n-1}} & \dots & \widehat{A_{n-1,n-1}} \end{pmatrix}
 \end{aligned}$$

## Calcul de l'inverse

Via les cofacteurs : (4)

- ▶ Nous obtenons ainsi lorsque le déterminant est non nul :

$$A^{-1} = \frac{1}{\det A} \begin{pmatrix} \widehat{A_{0,0}} & \widehat{A_{1,0}} & \dots & \widehat{A_{n-1,0}} \\ \widehat{A_{0,1}} & \widehat{A_{1,1}} & \dots & \widehat{A_{n-1,1}} \\ \vdots & \vdots & \dots & \vdots \\ \widehat{A_{0,n-1}} & \widehat{A_{1,n-1}} & \dots & \widehat{A_{n-1,n-1}} \end{pmatrix}$$

- ▶ Le calcul de l'inverse via cette approche fait appel à  $n^2$  calculs de déterminants de tailles  $n - 1$ .

# Calcul de l'inverse

## Résolution d'un système

- Nous pouvons aussi chercher l'inverse en résolvant le système suivant :

$$\begin{pmatrix} a_{0,0} & a_{0,1} & \dots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \dots & a_{1,n-1} \\ \vdots & \vdots & \dots & \vdots \\ a_{n-1,0} & a_{n-1,1} & \dots & a_{n-1,n-1} \end{pmatrix} \cdot \begin{pmatrix} x_{0,0} & x_{0,1} & \dots & x_{0,n-1} \\ x_{1,0} & x_{1,1} & \dots & x_{1,n-1} \\ \vdots & \vdots & \dots & \vdots \\ x_{n-1,0} & x_{n-1,1} & \dots & x_{n-1,n-1} \end{pmatrix} \\ = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix}$$

# Calcul de l'inverse

## Résolution d'un système

- ▶ Nous avons vu que la résolution d'un tel système peut être réalisée en deux étapes :
  1. Décomposition  $LU$  de la matrice  $A$  :  $P.A = L.U$
  2. Résolution de  $L.Y = P.Ident = P$
  3. Résolution de  $UX = Y$
- ▶ Nous avons vu que le coût de cette résolution est en  $\frac{4n^3}{3}$

# Calcul de l'inverse

## Résolution d'un système : algorithme

### InverseMat( $A, n$ )

**Entrées** une matrice  $A$  de dimension  $n \times n$

**Sortie**  $AI = A^{-1}$

**Corps**  $P, L, U := \text{PermutLU}(A, n);$

for  $i = 0$  to  $n - 1$  do

$Y[0..n - 1, i] := \text{ReducTriangL}(L, P[0..n - 1, i], n);$

for  $i = 0$  to  $n - 1$  do

$AI[0..n - 1, i] := \text{ReducTriangU}(U, Y[0..n - 1, i], n);$

# Calcul de l'inverse

## Résolution d'un système : Exemple

$$A := \begin{bmatrix} 2 & 7 & 8 & 1 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{bmatrix}$$

$$P, L, U := \text{PermutLU}(A, n);$$

$$P := \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, L := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1/4 & 1 & 0 & 0 \\ 7/8 & -1/22 & 1 & 0 \\ 7/8 & -1/22 & -1/5 & 1 \end{bmatrix}, U := \begin{bmatrix} 8 & 6 & 10 & 9 \\ 0 & 11/2 & 11/2 & -5/4 \\ 0 & 0 & -5/2 & -\frac{129}{44} \\ 0 & 0 & 0 & \frac{163}{110} \end{bmatrix}$$



## Calcul de l'inverse

### Résolution d'un système : Exemple (suite)

La réduction triangulaire inférieure donne :

$$Y := \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & -1/4 & 0 \\ 1/22 & 1 & -\frac{39}{44} & 0 \\ \frac{3}{55} & 1/5 & -\frac{117}{110} & 1 \end{bmatrix}$$

La réduction triangulaire supérieure permet d'obtenir l'inverse :

$$A^{-1} := \begin{bmatrix} -\frac{25}{163} & \frac{17}{163} & \frac{80}{163} & -\frac{78}{163} \\ \frac{41}{163} & \frac{96}{163} & -\frac{229}{163} & \frac{154}{163} \\ -\frac{10}{163} & -\frac{91}{163} & \frac{195}{163} & -\frac{129}{163} \\ \frac{6}{163} & \frac{22}{163} & -\frac{117}{163} & \frac{110}{163} \end{bmatrix}$$

La matrice des cofacteurs est (le déterminant est  $-163$ )

$$\hat{A}^t := \begin{bmatrix} 25 & -17 & -80 & 78 \\ -41 & -96 & 229 & -154 \\ 10 & 91 & -195 & 129 \\ -6 & -22 & 117 & -110 \end{bmatrix}$$

# Conditionnement

## Rappel

- ▶ Nous savons que le conditionnement se calcule :

$$\text{cond}(A) = \|A\| \cdot \|A^{-1}\| \text{ avec } \|A\| = \sup_{\|v\|=1} \|Av\|$$

Un système est bien conditionné si  $\text{cond}(A)$  est petit.

- ▶ La norme 1 est définie par :  $\|A\|_1 = \max_{j=0..n-1} \sum_{i=0}^{n-1} |a_{i,j}|$
- ▶ La norme  $\infty$  est définie par :  $\|A\|_\infty = \max_{i=0..n-1} \sum_{j=0}^{n-1} |a_{i,j}|$

# Conditionnement

## Exemple

- ▶ La matrice  $A$  précédente

$$A := \begin{bmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{bmatrix}$$

Donne pour la norme 1 de Maple un conditionnement de 4488

- ▶ Alors que la matrice  $B$

$$B := \begin{bmatrix} 1 & 7 & 2 & 1 \\ 7 & 5 & 1 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 1 \end{bmatrix}$$

a un conditionnement de 7.0.

# Conditionnement

## Exemple (2)

La résolution du système

$$\begin{bmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 32 \\ 23 \\ 33 \\ 31 \end{bmatrix}$$

Donne comme résultats intermédiaire et final :

$$\rightarrow Y := \begin{bmatrix} 32 \\ 7.4 \\ 6.75 \\ 0.1 \end{bmatrix} \rightarrow \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

# Conditionnement

## Exemple (3)

Si on effectue une légère perturbation sur le vecteur initial :

$$A := \begin{bmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} := \begin{bmatrix} 32.1 \\ 22.9 \\ 33.1 \\ 30.9 \end{bmatrix}$$

Nous constatons une forte perturbation sur le résultat final (pour rappel le conditionnement était de 4488)

$$Y := \begin{bmatrix} 32.1000000000000014 \\ 7.41999999999999993 \\ 6.57499999999999929 \\ -0.110000000000000001 \end{bmatrix} \rightarrow \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 9.19999999999999929 \\ -12.5999999999999996 \\ 4.50000000000000000 \\ -1.10000000000000009 \end{bmatrix}$$

# Conditionnement

## Exemple (4)

Considérons maintenant la résolution du système

$$\begin{bmatrix} 1 & 7 & 2 & 1 \\ 7 & 5 & 1 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} := \begin{bmatrix} 11 \\ 18 \\ 33 \\ 22 \end{bmatrix}$$

nous obtenons comme résultats intermédiaire et final :

$$\rightarrow Y := \begin{bmatrix} 33 \\ 6.875 \\ -10.6 \\ -6.98 \end{bmatrix} \rightarrow \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

# Conditionnement

## Exemple(5)

Si on effectue une  $l$  perturbation du même ordre que celui effectué sur le système précédent :

$$A := \begin{bmatrix} 1 & 7 & 2 & 1 \\ 7 & 5 & 1 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} := \begin{bmatrix} 11.1 \\ 17.9 \\ 33.1 \\ 21.9 \end{bmatrix}$$

Nous constatons une faible perturbation sur le résultat final (pour rappel le conditionnement était de 7)

$$Y := \begin{bmatrix} 33.1000000000000014 \\ 6.96250000000000036 \\ -10.7840000000000007 \\ -7.17512953399999986 \end{bmatrix} \rightarrow \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0.95504450979999997 \\ 1.01290801200000002 \\ 1.01364985199999991 \\ 1.02729970299999995 \end{bmatrix}$$

# Evaluation/Interpolation



## Evaluation d'un Polynôme

### Méthode Naïve

Considérons un polynôme de degré  $k - 1$  défini par ses  $k$  coefficients  $a_i$

$$A(X) = \sum_{i=0}^{k-1} a_i X^i$$

EvalPol( $A, \theta$ )

**Entrée** ( $a_0, \dots, a_{k-1}$ ) les coefficients,  $\theta$  la valeur du point

**Sortie** la valeur du polynôme  $A$  en  $\theta$  :  $V = A(\theta)$

**Corps**  $V \leftarrow a_0, C \leftarrow \theta$

**Pour**  $i = 1$  à  $k - 1$  **faire**

$$V \leftarrow a_i \times C + V$$

$$C \leftarrow C \times \theta$$

## Evaluation d'un Polynôme

### Méthode de Horner (premier ordre)

Considérons un polynôme de degré  $k - 1$  défini par ses  $k$  coefficients  $a_i$

$$A(X) = \sum_{i=0}^{k-1} a_i X^i$$

Nous avons

$$A(X) = a_0 + X(a_1 + X(a_2 + X(a_3 + \dots + X(a_{k-2} + X(a_{k-1})))) \dots)$$

## Evaluation d'un Polynôme

### Algorithme de Horner (premier ordre)

Horner( $A, \theta$ )

**Entrée**  $(a_0, \dots, a_{k-1})$  les coefficients du polynôme,  $\theta$  la valeur du point

**Sortie** la valeur du polynôme  $A$  en  $\theta$  :  $A(\theta)$

**Corps**  $V \leftarrow a_{k-1}$

**Pour**  $i = k - 2$  à  $0$  faire  $V \leftarrow a_i + \theta \times V$

# Evaluation d'un Polynôme

## Algorithme de Horner (premier ordre) Exemple

- ▶  $A(X) := 2 * X^4 + 5 * X^2 + 1$  en  $X = 3$
- ▶
  1.  $V \leftarrow 2$
  2.  $V \leftarrow 0 + 3 * 2$
  3.  $V \leftarrow 5 + 3 * 6$
  4.  $V \leftarrow 0 + 3 * 23$
  5.  $V \leftarrow 1 + 3 * 69 = 208$

## Evaluation d'un Polynôme

### Méthode de Horner (second ordre)

On sépare les indices pairs des indices impairs, nous supposons  $k$  pair.

$$\begin{aligned} A(X) &= \sum_{i=0}^{k-1} a_i X^i \\ &= \sum_{i=0}^{k/2-1} a_{2i} X^{2i} + X \cdot \sum_{i=0}^{k/2-1} a_{2i+1} X^{2i} \\ &= A_0(X^2) + X \cdot A_1(X^2) \end{aligned}$$

Approche parallélisable

Pratique si calcul de  $P(x)$  et de  $P(-x)$

## Evaluation d'un Polynôme

### Algorithme de Horner (second ordre)

Soit  $k - 1$  le degré, nous supposons que  $k = 2^n$

PreCalcul( $\theta, n$ )

**Entrée**  $\theta$  la valeur du point,  $k = 2^n$

**Sortie** le tableau  $C[i] = \theta^{2^i}$

**Corps**  $C[0] \leftarrow \theta$

**Pour**  $i = 1$  à  $n - 1$  **faire**  $C[i] \leftarrow C[i - 1] * C[i - 1]$

## Evaluation d'un Polynôme

### Algorithme de Horner (second ordre)

HornerBis( $A, C, i$ )

**Entrée**  $(a_0, \dots, a_{k-1})$  les coefficients du polynôme, le tableau  $C[i] = \theta^{2^i}$

**Sortie** la valeur du polynôme  $A$  en  $\theta^{2^i}$  :  $A(\theta^{2^i})$

**Corps** **Si**  $\deg A = 0$  **Alors**  $V \leftarrow A_0$

**Sinon**

$$\begin{cases} A_0(X) \leftarrow \sum_{i=0}^{\frac{k}{2}-1} a_{2i} X^i \\ A_1(X) \leftarrow \sum_{i=0}^{\frac{k}{2}-1} a_{2i+1} X^i \end{cases}$$

$$V \leftarrow \text{HornerBis}(A_0, C, i+1) + C[i].\text{HornerBis}(A_1, C, i+1)$$

**Appel initial** :  $\text{HornerBis}(A, C, 0)$

# Evaluation d'un Polynôme

## Complexité

- ▶ La complexité de l'algorithme naïf est de  $2(k - 1)$  produits et  $k - 1$  additions où  $k - 1$  représente le degré.
- ▶ La complexité de Horner premier ordre est de  $k - 1$  produits et  $k - 1$  additions où  $k - 1$  représente le degré.
- ▶ La complexité de Horner second ordre est de  $T(k) = 2T(k/2) + 1 = (k - 1)$  produits-additions avec  $T(1) = 0$  et  $(\log_2 k - 1)$  carrés pour la construction de la table.



## Evaluation d'une puissance $X^n$

### Méthode Naïve

L'exposant peut s'écrire  $n = \sum_{i=0}^{k-1} n_i 2^i$  d'où l'algorithme :

PuissNaif( $\theta, n$ )

**Entrée**  $\theta$  et un entier  $n$

**Sortie**  $p = \theta^n$

**Corps**  $p \leftarrow 1, C \leftarrow \theta$

**Tant que**  $n > 0$

**si**  $n \bmod 2 = 1$  **alors**  $p \leftarrow p * C$

$C \leftarrow C * C$

$n \leftarrow n/2$  (division entière)

## Evaluation d'une puissance $X^n$

### Méthode récursive

L'exposant peut s'écrire  $n = 2 * n_1 + n_0$  d'où l'algorithme :

PuissRec( $\theta, n$ )

**Entrée**  $\theta$  et un entier  $n$

**Sortie**  $p = \theta^n$

**Corps** **Si**  $n = 0$  **Alors**  $p \leftarrow 1$

**Sinon**  $p \leftarrow$  PuissRec( $\theta, \lfloor n/2 \rfloor$ )

$p \leftarrow p * p$

**si**  $n \bmod 2 = 1$  **alors**  $p \leftarrow p * \theta$

## Evaluation d'une puissance $X^n$

### Méthode itérative

Remarque similaire à l'approche de Horner :

$$n = \sum_{i=0}^{k-1} n_i 2^i = (\dots(n_{k-1} * 2 + n_{k-2}) * 2 + \dots + n_1) * 2 + n_0$$

PuissIt( $\theta, n$ )

**Entrée**  $\theta$  et un entier  $n = \sum_{i=0}^{k-1} n_i 2^i$

**Sortie**  $p = \theta^n$

**Corps**  $p \leftarrow 1$

**Pour**  $i = k - 1$  à 0 faire

$p \leftarrow p * p$

**si**  $n_i = 1$  **alors**  $p \leftarrow p * \theta$

## Evaluation d'une puissance $X^n$

### Complexité

- ▶ Soit un entier  $n$  l'exposant,  $n = \sum_{i=0}^{k-1} n_i 2^i$
- ▶ Les différents algorithmes proposés réclament  $k - 1$  mises au carré, et un nombre de produits égal au poids de Hamming de  $n$  (nombre de 1 dans l'écriture binaire)
- ▶ Une réécriture de l'exposant lorsque celui-ci est fixé, peut réduire cette complexité : sous forme de produit, ou de chaîne d'addition.

## Représentation d'un polynôme

- ▶ Un polynôme de degré  $k - 1$  peut être défini par la donnée :
  - ▶ de ses  $k$  coefficients  $a_i$

$$A(X) = \sum_{i=0}^{k-1} a_i X^i$$

- ▶ de ses valeurs en  $k$  points différents  $e_i$

$$\text{pour } i = 0, k \quad A(e_i) = \sum_{i=0}^{k-1} a_i e_i^i$$

les  $e_i$  sont choisis suivant deux critères : un calcul simple des  $A(e_i)$  et une taille respectable des  $A(e_i)$ .

# Reconstruction des coefficients

## Méthode de Lagrange

- ▶ Le principe est de construire une somme de  $k$  polynômes tels que le  $i$ -ième vaut  $P(e_i)$  en  $e_i$  et 0 pour tous les autres  $e_j$ ,  $j \neq i$ .

$$P(X) = \sum_{i=0}^{k-1} P(e_i) \frac{\prod_{j \neq i} (X - e_j)}{\prod_{j \neq i} (e_i - e_j)}$$

# Reconstruction des coefficients

## Méthode de Newton

- ▶ L'idée est d'écrire le polynôme comme dans un système de numération en augmentant le degré des polynômes sommés.

$$P(X) = \sum_{i=0}^{k-1} \hat{p}_i \prod_{j=0}^{i-1} (X - e_j) = \hat{p}_0 + \hat{p}_1(X - e_0) + \hat{p}_2(X - e_0)(X - e_1) + \dots$$

$$\left\{ \begin{array}{l} \hat{p}_0 = p'_0 \\ \hat{p}_1 = (p'_1 - \hat{p}_0)/(e_1 - e_0) \\ \dots \\ \hat{p}_i = (\dots (p'_i - \hat{p}_0)/(e_i - e_0) - \hat{p}_1)/(e_i - e_1) - \dots - \hat{p}_{i-1})/(e_i - e_{i-1}) \\ \dots \\ \hat{p}_{k-1} = (\dots (p'_{k-1} - \hat{p}_0)/(e_{k-1} - e_0) - \hat{p}_1)/(e_{k-1} - e_1) \dots - \hat{p}_{k-2})/(e_{k-1} - e_{k-2}) \end{array} \right.$$

Où,  $p'_i = P(e_i)$

## Produit de deux nombres

via les polynômes

- ▶ Soient  $A = \sum_{i=0}^{k-1} a_i \beta^i$  et  $B = \sum_{i=0}^{k-1} b_i \beta^i$  deux nombres en base  $\beta$
- ▶ Soient  $A(X) = \sum_{i=0}^{k-1} a_i X^i$  et  $B(X) = \sum_{i=0}^{k-1} b_i X^i$  les polynômes associés
- ▶ Calcul du produit  $P = A \times B$  :
  1. Evaluation du produit polynomial :  $P(X) = A(X) \times B(X)$
  2. Evaluation de la valeur:  $P(\beta) = A(\beta) \times B(\beta)$



# Produit de deux nombres

via les polynômes : remarques

- ▶ A l'étape 1, les coefficients  $p_i$  obtenus sont inférieurs à  $k \times \beta^2$
- ▶ A l'étape 2, le calcul de  $P(\beta)$  revient à réduire les  $p_i$  en propageant des retenues.

## Produit de deux nombres

### Algorithme Karatsuba (1)

- ▶ Choix des points  $e_0 = 0$ ,  $e_1 = -1$  et  $e_2 = \infty$

- ▶ Nous avons :

$$A = \sum_{i=0}^{k-1} a_i \beta^i = \left( \sum_{i=0}^{k/2-1} a_{k/2+i} \beta^i \right) \beta^{k/2} + \sum_{i=0}^{k/2-1} a_i \beta^i = A_1 \beta^{k/2} + A_0$$

- ▶ Point de vue polynomial :  $A(X) = A_1 X + A_0$

$$\begin{cases} A(0) = A_0 \\ A(-1) = A_0 - A_1 \\ A(\infty) = \lim_{X \rightarrow \infty} A_1 X \end{cases}$$

## Produit de deux nombres

### Algorithme Karatsuba (2)

- ▶ Valeurs du polynôme produit

$$\begin{cases} P(0) = A_0 B_0 \\ P(-1) = (A_0 - A_1)(B_0 - B_1) \\ P(\infty) = \lim_{X \rightarrow \infty} A_1 B_1 X^2 \end{cases}$$

- ▶ Application de Newton

$$\begin{cases} \hat{p}_0 = P(0) = A_0 B_0 \\ \hat{p}_1 = (P(-1) - \hat{p}_0)/(-1) = (A_1 - A_0)(B_0 - B_1) + A_0 B_0 \\ \hat{p}_\infty = \lim_{X \rightarrow \infty} ((P(\infty) - \hat{p}_0)/X - \hat{p}_1)/(X + 1) = A_1 B_1 \end{cases}$$

## Produit de deux nombres

### Algorithme Karatsuba (3)

► Reconstruction

$$\left\{ \begin{array}{l} P(X) = \hat{p}_0 + \hat{p}_1 X + \hat{p}_\infty X(X+1) \\ = A_0 B_0 \\ \quad + ((A_1 - A_0)(B_0 - B_1) + A_0 B_0 + A_1 B_1) X \\ \quad + A_1 B_1 X^2 \end{array} \right.$$

► Calcul

$$\left\{ \begin{array}{l} P(\beta^{k/2}) = A_0 B_0 \\ \quad + ((A_1 - A_0)(B_0 - B_1) + A_0 B_0 + A_1 B_1) \beta^{k/2} \\ \quad + A_1 B_1 \beta^k \end{array} \right.$$

## Produit de deux nombres

### Algorithme Karatsuba (4) : Complexité

- ▶ On note  $K(n)$  le nombre d'opérations élémentaires
- ▶ On a la formule de récurrence  $K(n) = 3K(n/2) + \alpha n$  en supposant les additions linéaires
- ▶ Nous obtenons,  $K(n) = O(n^{\log_2(3)})$

## Produit de deux nombres

### Algorithme Toom Cook (1)

L'approche de Karatsuba se généralise, par exemple avec une découpe en trois et donc 5 points différents

▶ Choix des points  $e_0 = 0$ ,  $e_1 = -1$ ,  $e_2 = 1$ ,  $e_3 = 2$  et  $e_4 = \infty$

▶ Nous avons :

$$A = A_2\beta^{2k/3} + A_1\beta^{k/3} + A_0$$

▶ Point de vue polynomial :  $A(X) = A_2X^2 + A_1X + A_0$

$$\left\{ \begin{array}{l} A(0) = A_0 \\ A(-1) = A_2 - A_1 + A_0 \\ A(1) = A_2 + A_1 + A_0 \\ A(2) = 4A_2 + 2A_1 + A_0 \\ A(\infty) = \lim_{X \rightarrow \infty} A_2X^2 \end{array} \right.$$

## Produit de deux nombres

### Algorithme Toom Cook (2)

- ▶ Comme pour Karatsuba nous appliquons Newton...

$$\left\{ \begin{array}{l} \hat{p}_0 = P(0) = A_0 B_0 \\ \hat{p}_1 = (P(-1) - \hat{p}_0)/(-1) \\ \hat{p}_2 = ((P(1) - \hat{p}_0)/(1) - \hat{p}_1)/(2) \\ \hat{p}_3 = (((P(2) - \hat{p}_0)/(2) - \hat{p}_1)/(3) - \hat{p}_2)/(1) \\ \hat{p}_4 = \lim_{X \rightarrow \infty} (((((P(\infty) - \hat{p}_0)/X - \hat{p}_1)/(X + 1) - \hat{p}_2)/(X - 1) - \hat{p}_3)/(X - 2)) \\ \quad = A_2 B_2 \end{array} \right.$$

- ▶ Ici nous remarquons une division par 3 qui montre un peu les limites de cette approche lorsque l'on généralise
- ▶ Reconstruction en calculant  $P(\beta^{k/3})$  avec :

$$P(X) = \hat{p}_0 + X(\hat{p}_1 + (X + 1)(\hat{p}_2 + (X - 1)(\hat{p}_3 + \hat{p}_4(X - 2))))$$

# Produit de deux nombres

## Algorithme Toom Cook (3)

- ▶ On note  $T_3(n)$  le nombre d'opérations élémentaires
- ▶ On a la formule de récurrence  $T_3(n) = 5T_3(n/3) + \alpha n$  en supposant les additions linéaires
- ▶ Nous obtenons,  $T_3(n) = O(n^{\log_3(5)})$



## Produit de deux nombres

### Algorithme Toom Cook (4)

- ▶ On généralise en découpant en  $k$
- ▶ On note  $T_k(n)$  le nombre d'opérations élémentaires
- ▶ On a la formule de récurrence  $T_k(n) = (2k - 1)T_k(n/k) + \alpha n$  en supposant les additions linéaires
- ▶ Nous obtenons,  $T_k(n) = O(n^{\log_k(2k-1)})$
- ▶ Nous pouvons donc dire que la multiplication peut se faire pour tout  $\epsilon$  en  $O(n^{1+\epsilon})$

## Itération de Newton

résolution de l'équation  $f(x) = 0$

Deux sites ont inspiré ce cours <sup>6</sup> <sup>7</sup>

---

<sup>6</sup><http://www.cse.illinois.edu/heath/scicomp/notes/chap05.pdf>

<sup>7</sup>[http://www.pi314.net/fr/algo\\_newton.php](http://www.pi314.net/fr/algo_newton.php)

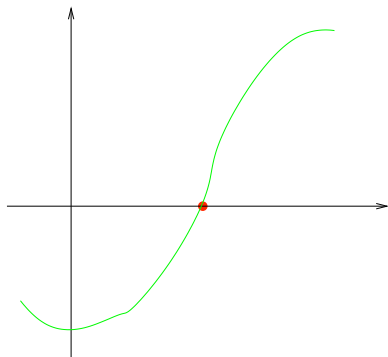
# Trouver une solution à $f(x) = 0$

## Présentation du problème

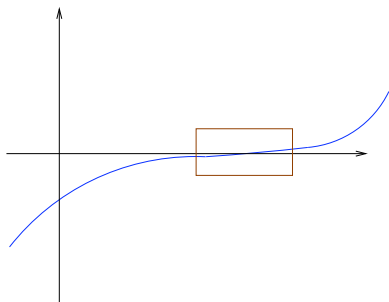
- ▶ Nous désirons trouver une solution de l'équation  $f(x) = 0$  sous la forme d'un nombre "virgule flottante".
- ▶ Une équation de ce type peut avoir aucune, une ou plusieurs solutions
- ▶ Nous supposerons qu'elle en a au moins une que l'on cherchera à déterminer

# Trouver une solution à $f(x) = 0$

Que cherche-t-on à résoudre ?



(a)  $|x - e| \approx 0$



(b)  $f(x) \approx 0$

## Trouver une solution à $f(x) = 0$

### Solution simple

- ▶ Une solution  $e$  est dite simple si  $f(e) = 0$  et  $f'(e) \neq 0$
- ▶ Le conditionnement est donné par  $\frac{1}{|f'(e)|}$
- ▶ Le taux de convergence pour  $x_k$  obtenue à la  $k$ -ième itération, est caractérisé par  $r$  tel que :

$$\lim_{k \rightarrow \infty} \frac{|x_{k+1} - e|}{|x_k - e|^r} = C$$

- ▶ À chaque itération nous obtenons  $r$  fois plus de précision.

## Méthode de bisection

Solution simple  $f(e) = 0$  avec  $e \in [a, b]$

- ▶ Nous cherchons à résoudre  $f(x) = 0$ , sachant que  $e \in [a, b]$  et que  $f(a) * f(b) < 0$  (autrement dit,  $f(a)$  et  $f(b)$  sont de signes différents)
- ▶ On considère le milieu  $m$  de l'intervalle :  $m = \frac{b+a}{2}$
- ▶ Si  $f(m)$  est du signe de  $f(a)$  alors on remplace  $a$  par  $m$ , sinon  $b$  par  $m$  (*nous considérons qu'il n'y a qu'une solution sur cet intervalle*)
- ▶ On réitère l'opération jusqu'à ce que la taille de l'intervalle corresponde à la précision souhaitée.

Animation :

[http://www.cse.illinois.edu/iem/nonlinear\\_eqns/Bisection/](http://www.cse.illinois.edu/iem/nonlinear_eqns/Bisection/)

## Méthode de bisection

Solution simple  $f(e) = 0$  avec  $e \in [a, b]$

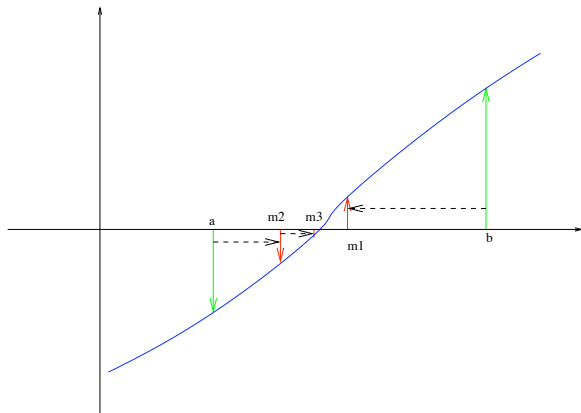


Figure : Méthode de la bisection avec  $m_i$ , ... les milieux successifs

## Méthode de bisection

Solution simple  $f(e) = 0$  avec  $e \in [a, b]$

Bissec( $f, a, b$ )

**Entrées** la fonction  $f$ , les extrémités  $a$  et  $b$ , la précision  $\varepsilon$

**Sortie**  $m$  tel que  $|m - e| < \varepsilon$

**Corps** Tant que  $((b - a) \geq \varepsilon)$  faire

$$m \leftarrow (a + b)/2$$

Si  $f(a) * f(m) > 0$  Alors

$$a \leftarrow m$$

Sinon

$$b \leftarrow m$$



## Méthode de bisection

Solution simple  $f(e) = 0$  avec  $e \in [a, b]$

- ▶ À chaque itération l'intervalle diminue de moitié
- ▶ Si on se place en base deux, cela revient à obtenir un nouveau chiffre de précision à chaque itération
- ▶ Le taux de convergence est dit linéaire ( $r = 1$  et  $C = 0.5$ )
- ▶ Le nombre d'itérations est de l'ordre de  $\lceil \log_2(\frac{b-a}{\epsilon}) \rceil$

## Méthode du point fixe

### Principe

- ▶ Soit une fonction  $g$ , on appelle point fixe, une valeur  $x$  telle que :  $g(x) = x$
- ▶ Schéma de recherche d'un point fixe, avec comme itération :

$$x_{k+1} = g(x_k)$$

- ▶ La résolution de l'équation  $f(x) = 0$  peut se faire via une approche itérative de point fixe. Pour une équation, il peut y avoir plusieurs approches. Mais toutes ne convergent pas.
- ▶ Exemple : pour  $f(x) = x^2 - x - 5$  la recherche de point fixe peut être faite via  $g(x) = x^2 - 5$  ou encore  $g(x) = \sqrt{x + 5} \dots$

# Méthode du point fixe

## Convergence

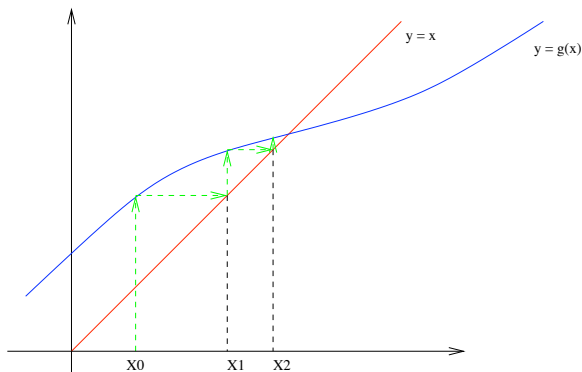


Figure : Méthode du point fixe avec  $X_i$ , ... les itérations successives

# Méthode du point fixe

## Divergence

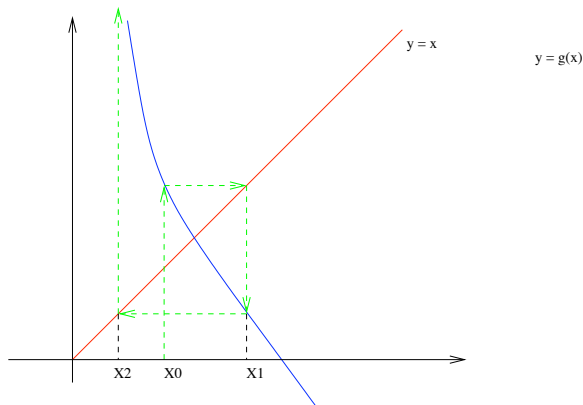


Figure : Méthode du point fixe avec  $X_i$ , ... les itérations successives

## Méthode du point fixe

### Conditions de convergence

- ▶ Soit une fonction  $g$ , telle que :  $g(e) = e$ , Si  $|g'(e)| < 1$  Alors il existe un intervalle sur lequel l'itération  $x_{k+1} = g(x_k)$  converge.
- ▶ Si  $|g'(e)| > 1$  Alors ce schéma diverge.
- ▶ La convergence asymptotique est en général linéaire avec  $C = |g'(e)|$
- ▶ De plus si  $g'(e) = 0$  alors la convergence est au moins quadratique.

Animation :

[http://www.cse.illinois.edu/iem/nonlinear\\_eqns/FixedPoint/](http://www.cse.illinois.edu/iem/nonlinear_eqns/FixedPoint/)

## Méthode de Newton

### Principe de la tangente

- ▶ Nous considérons la fonction tangente  $h_k$  en  $x_k$ , telle que :

$$h_k(x) = f'(x_k) \cdot x + f(x_k) - f'(x_k) \cdot x_k$$

(la tangente  $h_k$  est une approximation de  $f$  au voisinage de  $x_k$  )

- ▶ La fonction  $h_k$  est affine en  $x$ , et a pour racine

$$x_k - \frac{f(x_k)}{f'(x_k)}$$

(la tangente coupe l'axe des abscisses en ce point)

- ▶ L'**itération de Newton** est définie par

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

# Méthode de Newton

## Principe de la tangente

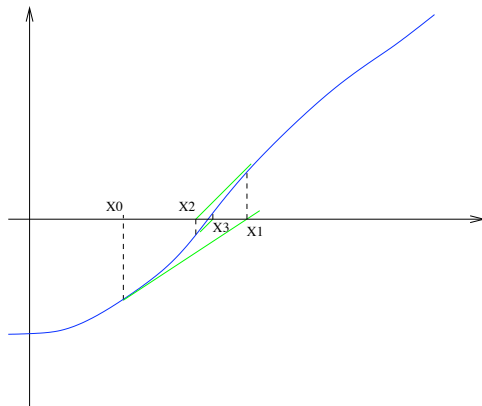


Figure : Méthode de Newton via les tangentes

## Méthode de Newton

### Approche du point fixe

- ▶ Ainsi résoudre  $f(x) = 0$  peut se réduire en un problème de point fixe avec la fonction  $g$  telle que

$$g(x) = x - \frac{f(x)}{f'(x)}$$

- ▶ La convergence est assurée lorsque  $|g'(e)| < 1$  avec 
$$g'(x) = \frac{f(x) \cdot f''(x)}{(f'(x))^2}$$
- ▶ Comme  $e$  est racine simple de  $f(x)$ , nous avons  $g'(e) = 0$ , ce qui assure une **convergence quadratique** à condition de *démarrer sur un petit voisinage de  $e$* .

Animation : <http://www.cse.illinois.edu/iem/nonlinear\protect\unhbox\voidb@x\kern.06em\vbox{\hrulewidth.3em}eqns/Newton/>



## Méthode de Newton

### Exemple de l'inversion

- ▶ Pour calculer l'inverse de  $a$ , ici  $|a| > 1$ , nous considérons  $f(x) = \frac{1}{x} - a$  qui permet d'approcher  $\frac{1}{a}$  avec l'itération

$$x_{k+1} = 2.x_k - a.x_k^2 = x_k.(2 - a.x_k)$$

- ▶ Nous avons :  $x_{k+1} - \frac{1}{a} = -a.(x_k - \frac{1}{a})^2$  assurant une convergence quadratique
- ▶ Si par exemple  $|x_1 - \frac{1}{a}| < 1/2$  alors  $|x_{k+1} - \frac{1}{a}| < 1/2^n$  avec  $n = 2^k$

## Méthode de Newton

### Exemple de la racine carrée

- ▶ Pour calculer la racine carrée de  $a$  nous pouvons résoudre l'équation  $f(x) = 0$  avec  $f(x) = a - x^2$  mais dans ce cas l'itération de Newton devient

$$x_{k+1} = x_k + \frac{a - x_k^2}{2 \cdot x_k}$$

ce qui demande une division à chaque itération.

- ▶ Nous préférons  $f(x) = \frac{1}{x^2} - a$  qui permet d'approcher  $\frac{1}{\sqrt{a}}$  avec l'itération

$$x_{k+1} = x_k + x_k \cdot \frac{1 - a \cdot x_k^2}{2}$$

$\sqrt{a}$  est obtenue en multipliant le résultat par  $a$

## Théorème de Sturm

Nombre de solutions d'une équation  $P(x) = 0$

## Suite de Sturm associée à un polynôme $P$ de degré $n$ à racines réelles simples

Définition (suite de Sturm d'un polynôme)

$$P_0 \leftarrow P$$

$$P_1 \leftarrow P'$$

$$P_{k+2} \leftarrow -(P_k \bmod P_{k+1})$$

*Suite de  $n + 1$  polynômes de degré décroissant ( $P_i$  est de degré  $n - i$ .)*

Définition (Variation)

$V(x)$ : le nombre de changement de signes de la suite  $P_0(x), P_1(x), \dots, P_n(x)$  sans tenir compte des valeurs éventuellement nulles

Théorème de Sturm

Le nombre de racines réelles de  $P$  dans l'intervalle  $[a, b]$  est  $V(a) - V(b)$ .

## Exemple

On veut savoir combien  $P = X^3 - 3X^2 - X + 3$  a de racines dans l'intervalle  $[0, 4]$ . On détermine la suite de Sturm de  $P$ :

$$P_0(X) = X^3 - 3X^2 - X + 3$$

$$P_1(X) = 3X^2 - 6X - 1$$

$$P_2(X) = 8/3X - 8/3$$

$$P_3(X) = 4$$

On calcule:

▶  $P_0(0) = 3, P_1(0) = -1, P_2(0) = -8/3, P_3(0) = 4$  d'où  $V(0) = 2$

▶  $P_0(4) = 15, P_1(4) = 23, P_2(4) = 8, P_3(4) = 4$  d'où  $V(4) = 0$

et il y a  $V(0) - V(4) = 2$  racines entre 0 et 4.

## Exemple

Si on veut affiner, on calcule la variation au point milieu 2:

$P_0(2) = -3, P_1(2) = -1, P_2(2) = 8/3, P_3(2) = 4$  d'où  $V(2) = 1$

et il y a

- ▶  $V(0) - V(2) = 1$  racine entre 0 et 2
- ▶  $V(2) - V(4) = 1$  racine entre 2 et 4

On a isolé les racines entre 0 et 4, on peut alors appliquer l'itération de Newton pour trouver chacune de ces deux racines rapidement.