Représentations et Méthodes Numériques

2I011

UPMC (Paris 6)
France

2017

Équipe pédagogique et Évaluation

Cours:

- Christoph Lauter – `christoph.lauter@lip6.fr`

Travaux Dirigés et Travaux sur Machines:

- **Lundi:** M. Mezzarobba et C. Buron
- **Mardi:** Y. Rotella et J. Marrez
- **Vendredi:** Th. Espitau et V. Zucca
- **Groupe PIMA:** J. Berthomieu et V. Zucca

Évaluation:

- Examen 60%
- Partiel 30%
- TME 10%

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, Geométrie algorithmique, triangulation...
- l'intelligence artificielle
  - Google (pagerank, valeurs propres,...)
  - fouille de données, ...

Méthodes numériques : est-ce important ?

- 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 du à une mauvaise implantation de la division.

Programme

- Les entiers et l'Ordinateur
- Division euclidienne - congruences - algorithme d'Euclide - CRT
- Représentation des flottants - algorithmes élémentaires
- Arithmétique d'intervalles
- 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 $n \in \mathbb{N}$, $\sigma(n) \neq 0$. [1]
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 $P = \mathbb{N}$. [2]
   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 $\mathbb{N}$.

[1] cet item et le suivant permettent de définir la notion de prédécesseur, noté $\rho(n)$ avec $\sigma(\rho(n)) = \rho(\sigma(n)) = n$

[2] 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 $x + y' = x' + y$.[3]
- Les classes d'équivalence de $\mathbb{N} \times \mathbb{N}$ suivant $\mathcal{R}$ forment un ensemble noté $\mathbb{Z}$ **des entiers relatifs**. [4]
- 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 $(0, x)$

[3] En fait (x, y) est équivalent à $(x - y, 0)$ si $x \geq y$ et à $(0, y - x)$ sinon
[4] (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 mod a)

L'écriture des nombres

- Système de représentation dit de position (Gerbert d'Aurillac (940-1003), pape Sylvestre II)
- Choix d'une base β
  et de **chiffres** (alphabet) $\mathcal{C} = \{0, 1, 2, \ldots, \beta - 1\}$
- Le nombre n s'écrit : $n_{k-1} \ldots n_3 n_2 n_1 n_0 = \sum_{i=0}^{k-1} n_i \beta^i$ 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 β:

$B_0 \leftarrow 1$

$i \leftarrow 0$

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 β:

$i \leftarrow 0$

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 β, $A = a_{k-1}...a_1 a_0$ et $B = b_{k-1}...b_1 b_0$

Sorties $S = s_k s_{k-1}...s_1 s_0$

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, $A = a_{k-1}...a_1 a_0$ et $B = b_{k-1}...b_1 b_0$

Sorties $S = s_k s_{k-1}...s_1 s_0$

Corps $c \leftarrow 0$

Pour $i = 0$ **à** $k - 1$ **faire**

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

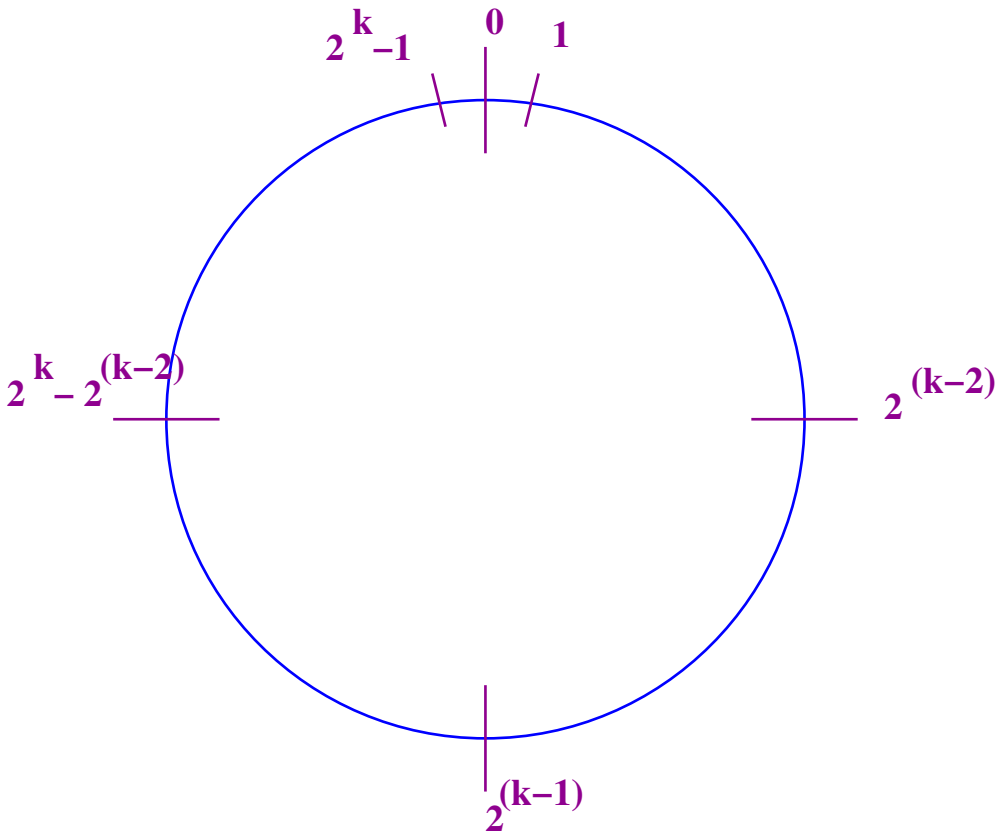
$c \leftarrow a_i.b_i \oplus a_i.c \oplus b_i.c$ (. et)

$s_k \leftarrow c$

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).

Les entiers de l'ordinateur



Premiers effets de bord

$$
\begin{aligned}
10^8 &\rightarrow 100000000 \\
10^9 &\rightarrow 1000000000 \\
10^{10} &\rightarrow 1410065408 \\
10^{11} &\rightarrow 1215752192 \\
10^{12} &\rightarrow 3567587328
\end{aligned}
$$

$$
\begin{aligned}
2^{30} &\rightarrow 1073741824 \\
2^{31} &\rightarrow 2147483648 \\
2^{32} &\rightarrow 0 \\
2^{33} &\rightarrow 0
\end{aligned}
$$

Algorithme d'addition avec contrôle

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

Sorties $S = s_{k-1}...s_1 s_0$ 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)

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

$OF \leftarrow c$

Les entiers non-signés en C

- En C, les entiers `int` sont signés par défaut
- On déclare un entier non-signé comme `unsigned int`
- Un entier C non-signé
  - peut représenter les valeurs entre 0 et $2^k - 1$,
  - respecte une arithmétique modulaire $\mod 2^k$
- Sauf qu'on ne connaît pas k pour `unsigned int`.
- Alors:
  - mettre `#include <stdint.h>` et
  - utiliser `uint8_t`, `uint16_t`, `uint32_t` ou `uint64_t`
- Attention: les littéraux comme 1664 sont signés!
  $\Rightarrow$ Utiliser 1664u à la place.

Les entiers non-signés en C

- Dans la machine, les entiers sont représentés en binaire
- c-à-d pour 42, on a bien l'octet 00101010 quelque part.

- Mais: on programme en C, donc on a le droit d'écrire 42 en décimal.
- C'est le compilateur qui se charge de la conversion.

- En revanche, on ne peut accéder au binaire qu'avec des opérations de masque et décalage...

Les masques

En C, pour deux valeurs `uint32_t a, b`

- `a & b` est le ET logique de tous les bits de `a` et de `b`

	`a`	=	42	= 00101010
	`b`	=	22	= 00010110
`a & b`		=	2	= 00000010

- `a | b` est le OU logique de tous les bits de `a` et de `b`

	`a`	=	42	= 00101010
	`b`	=	22	= 00010110
`a | b`		=	62	= 00111110

- Avec un tel masque & on peut donc accéder aux différents bits d'un entier en C:
  - `a & 1u` est non-nul ssi `a` a son bit à droite à 1
  - `a & 2u` est non-nul ssi `a` a son bit de poids 1 à 1
  - `a & 4u` est non-nul ssi `a` a son bit de poids 2 à 1
  - ...

Les décalages

En C, pour une valeur `uint32_t a` et un entier `b`

- `a << b` a les mêmes bits que `a` décalés de b bits à gauche

`a`	$=$	42	$=$	00101010
`a << 2`	$=$	168	$=$	10101000

- `a >> b` a les mêmes bits que `a` décalés de b bits à droite

`a`	$=$	68	$=$	01000100
`a >> 2`	$=$	17	$=$	00010001

- Les bits *en trop* à gauche ou à droite *disparaissent*.

- `1u << k` a un seul bit à 1 à la k-ième position

$\Rightarrow$ `a & (1u << k)` est non-nul ssi `a` a son bit de poids k à 1.

Les décalages et les masques

- De la même façon qu'on peut abréger `a = a + b` en `a += b`,
  on peut abréger `a = a & b` en `a &= b`,
  `a = a | b` en `a |= b`,
  `a = a >> b` en `a >>= b` et
  `a = a << b` en `a <<= b`.

Lecture de code C

Que fait ce code?

```
void func(uint32_t *q, uint32_t *r,
          uint32_t n, uint32_t d) {
  int i;
  *q = 0;
  *r = 0;
  for (i=31;i>=0;i--) {
    *r <<= 1;
    *r |= (n >> i) & 1;
    if (*r >= d) {
      *r -= d;
      *q |= 1 << i;
    }
  }
}
```

Les entiers signés

Comment représenter des entiers signés dans un ordinateur?

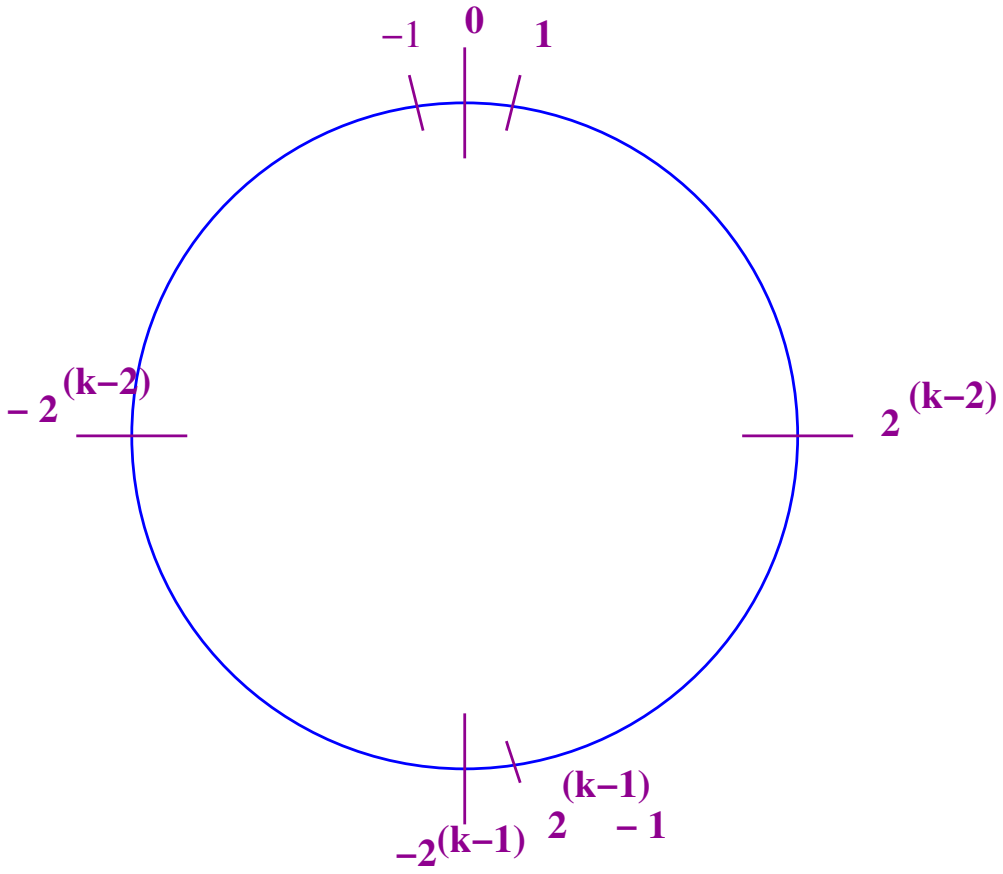
- Représentation *signe* (1 bit) + *valeur absolue*
  - Inconvénients
    - Deux représentations de zéro: -0 et $+0$
    - Algorithmes pour $+$ et $-$ compliqués:
      $\rightarrow$ une addition peut être une soustraction cachée.
- Représentation avec un biais
  - Idée:
    - décaler $[-2^{k-1}; 2^{k-1}-1]$ de 2^{k-1} en $[0; 2^k-1]$
    - représenter la valeur décalée (biaisée).
  - Inconvénients:
    - Signe plus difficile à connaître
    - Algorithmes pour $\times$ et $/$ compliquées:
      $\rightarrow$ il faut extraire le signe, la valeur absolue etc.
    - Les valeur positives n'ont pas la même suite de bit qu'en représentation non-signée.

Les entiers signés

Comment représenter des entiers signés dans un ordinateur?

- Représentation en complément à la base
  - Idée:
    - Laisser les entiers positifs tels quels
      $\rightarrow$ le bit a gauche est zéro si le nombre est suffisamment petit
    - Rajouter un biais de 2^k seulement au nombres négatifs, pour qu'il deviennent positifs
      $\rightarrow$ marquer le rajout de ce biais par un 1 sur le bit à gauche.
  - Donc:
    - $x \in [0; 2^{k-1} - 1]$: encoder x tel quel sur k bits
    - $x \in [-2^{k-1}; -1]$: encoder $2^k + x$ sur k bits; le k-ième bit est 1.
  - Avantages:
    - Accès au signe facile: prendre le bit à gauche
    - Une représentation unique
    - Rien n'est à changer dans les algorithmes pour $+, -, <$.
    - La multiplication est facile à gérer.
  - C'est la représentation utilisée dans les processeurs actuels.

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

$$
\begin{aligned}
10^8 &\rightarrow 100000000 \\
10^9 &\rightarrow 1000000000 \\
10^{10} &\rightarrow 1410065408 \\
10^{11} &\rightarrow 1215752192 \\
10^{12} &\rightarrow -727379968 \\
\\
2^{30} &\rightarrow 1073741824 \\
2^{31} &\rightarrow -2147483648 \\
2^{32} &\rightarrow 0 \\
2^{33} &\rightarrow 0
\end{aligned}
$$

Addition et soustraction en complément à la base

$$\begin{array}{rr} & 00001100 \\ + & 01001001 \\ \hline & 01010101 \end{array} \qquad \begin{array}{rr} & 10001100 \\ + & 01001001 \\ \hline & 11010101 \end{array} \qquad \begin{array}{rr} & 01001100 \\ + & 01001001 \\ \hline & 10010101 \end{array}$$

$$\begin{array}{rr} & 10001100 \\ + & 11001001 \\ \hline 1 & 01010101 \end{array} \qquad \begin{array}{rr} & 10001100 \\ + & 11110100 \\ \hline 1 & 10000000 \end{array}$$

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 $2F\ B1\ 12\ 0A$

Les entiers signés en C

- L'entier classique `int` est signé
- On n'en connaît pas la taille (pas facilement)
- On préférera `int8_t`, `int16_t`, `int32_t` et `int64_t`
- Un entier signé en C sur k bits couvre la plage $[-2^{k-1}; 2^{k-1} - 1]$
- La norme C ne dit rien sur une arithmétique signée "modulaire"; un dépassement de capacité a des comportements non-prévisibles.

Les entiers signés en C (cont'd)

- Une affectation non-signé $\rightarrow$ signé n'est définie que si la valeur tient sur la plage des entiers signés.
- Une affectation signé $\rightarrow$ non-signé n'est définie que si la valeur est positive.
- Les masques sur les signés sont mal spécifiés.
- Pour accéder à la représentation en bits d'un signé a, on peut le convertir en non-signé avec la même représentation en bits:

```
int32_t a; uint32_t aBits;
a = -1664;
aBits = *((uint32_t *) &a);
```

Ensuite, on peut appliquer les techniques vus pour accéder aux bits.

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 β, $A = a_{k-1}...a_1a_0$ et $B = b_{k-1}...b_1b_0$

Sorties $P = p_{2k-1}...p_1p_0 = A \times B$

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 β^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, $A = a_{k-1}...a_1 a_0$ et $B = b_{k-1}...b_1 b_0$

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

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 : $A = A_1\beta^{k/2} + A_0$ 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 β, $A = a_{k-1}...a_1 a_0$ et $B = b_{k-1}...b_1 b_0$

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

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

Sinon

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

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

$B_1 \leftarrow b_{k-1}...b_{t1}$ et $B_0 \leftarrow b_{t1-1}...a_0$ et $Sb \leftarrow 1$

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

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

$T \leftarrow$Karatsuba$(A_1, B_1, t2)$

$U \leftarrow$Karatsuba$(A_0, B_0, t1)$

$V \leftarrow$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
- $T(k) = 3 \times T(k/2) + \alpha k$ (αk 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

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

24571	53	Alignement
4×5300	463	diviseur $\times$ puissance de la base
-21200		(donne la position du chiffre du quotient)
3371		
6×530		choix d'un chiffre du quotient
-3180		soustraction du produit
191		chiffre quotient $\times$ diviseur
3×53		passage au chiffre suivant
-159		$24571 = 463 \times 53 + 32$
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 β est la base.
  Autrement dit, le quotient se réduit à un chiffre.
- $a = a_n a_{n-1} a_{n_1} \ldots a_1 a_0$ et $d = d_{n-1} d_{n-2} \ldots d_1 d_0$
- 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 $a_{n-1} \dots a_1 a_0$ 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**

$q \leftarrow q - 1$

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

UPMC SORBONNE UNIVERSITÉS

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

UPMC SORBONNE UNIVERSITÉS

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

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

$331 - 6 \times 590 = -23$

$331 - 5 \times 590 = 36$

Le PGCD

Rappels de math

d est un diviseur de *a* , s'il existe α 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 $\text{pcgd}(a, b) = \text{pgcd}(a - b, b)$

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

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 soustration

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 $pgcd(U, V) = pgcd(U - V, V)$

Terminaison : si $V = 0$ alors $U = pgcd(U, V)$

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

Algoritme 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 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 $F_n = \frac{1}{\sqrt{5}}(\phi^n - \phi'^n)$
- pgcd(F_n, F_{n-1}) demande n itérations, soit une complexité par rapport à la taille de F_n en base ϕ ($\log_\phi N$).

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

Algoritme à binaire (parité): Exemple

U	V	U - V
1479	699	780 $(/4 \rightarrow 195)$
699	195	504 $(/8 \rightarrow 63)$
195	63	132 $(/4 \rightarrow 33)$
63	33	30 $(/2 \rightarrow 15)$
33	15	18 $(/2 \rightarrow 9)$
15	9	6 $(/2 \rightarrow 3)$
9	3	6 $(/2 \rightarrow 3)$
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'un 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 :

$$\begin{aligned} (u_1, u_2, u_3) &\leftarrow (1, 0, a) \\ (v_1, v_2, v_3) &\leftarrow (0, 1, b) \end{aligned}$$

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

$$\begin{aligned} (t_1, t_2, t_3) &\leftarrow (u_1, u_2, u_3) - q \times (v_1, v_2, v_3) \\ (u_1, u_2, u_3) &\leftarrow (v_1, v_2, v_3) \\ (v_1, v_2, v_3) &\leftarrow (t_1, t_2, t_3) \end{aligned}$$

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

Itération : tant que $v_3 \neq 0$: $q = \lfloor u_3 \div v_3 \rfloor$
$$\begin{array}{rcl} (t_1, t_3) & \leftarrow & (u_1, u_3) - q \times (v_1, v_3) \\ (u_1, u_3) & \leftarrow & (v_1, v_3) \\ (v_1, v_3) & \leftarrow & (t_1, t_3) \end{array}$$

Théorème chinois des restes

Motivation

- Étant donné des entiers (*moduli*) $m_1, \ldots, m_n$ premiers entre eux et des restes associés $a_1, \ldots, a_n$, on cherche à resoudre le système de congruences

$$\begin{aligned} x &\equiv a_1 \mod m_1 \\ x &\equiv a_2 \mod m_2 \\ &\vdots \\ x &\equiv a_n \mod m_n \end{aligned}$$

  où x est l'inconnue.
- Problème important connu depuis le IIIe siècle de notre ère
  - Systèmes de numération modulaire
  - Cryptographie
  - Automates de vente etc.

Théorème chinois des restes

Algorithme

Entrées $m_1, \ldots, m_n$ des moduli premiers entre eux
$a_1, \ldots, a_n$ des entiers restes

Sortie x resolvant le système de congruences

Corps

$M \leftarrow m_1$

$x \leftarrow a_1$

Pour $i = 2, \ldots, n$ **faire**

Euclide étendu: calculer (u, v) tq.

$u\, m_i + v\, M = 1$

$x \leftarrow u\, m_i\, x + v\, M\, a_i$

$M \leftarrow m_i\, M$

$x \leftarrow x \mod M$

Théorème chinois des restes

Application

Une jeune fille portait un panier rempli d'œufs. Un chevalier passa avec son cheval et toucha le panier, qui tomba par terre et tous les œufs se cassèrent. Il voulut dédommager la fille et il lui demanda combien d'œufs elle avait eu.
Elle ne sut plus dire leur nombre mais elle se rappela que quand elle les avait comptés par paires, un œuf était resté seul, que quand elle avait compté par triplets, deux étaient restés et quand elle les avait arrangés par groupes de cinq, quatre étaient restés. Finalement, quand elle les avait comptés par groupes de sept, aucun œuf n'était resté à côté.
Alors le chevalier répondit: Maintenant je sais combien d'œufs il y avait.

Brahmagupta, VIIe siècle

Théorème chinois des restes

Application

Une jeune fille portait un panier rempli d'œufs. Un chevalier passa avec son cheval et toucha le panier, qui tomba par terre et tous les œufs se cassèrent. Il voulut dédommager la fille et il lui demanda combien d'œufs elle avait eu.
Elle ne sut plus dire leur nombre mais elle se rappela que quand elle les avait comptés par paires, un œuf était resté seul, que quand elle avait compté par triplets, deux étaient restés et quand elle les avait arrangés par groupes de cinq, quatre étaient restés. Finalement, quand elle les avait comptés par groupes de sept, aucun œuf n'était resté à côté.
Alors le chevalier répondit: Maintenant je sais combien d'œufs il y avait.

Brahmagupta, VIIe siècle

$$\begin{aligned} x &\equiv 1 \mod 2 \\ x &\equiv 2 \mod 3 \\ x &\equiv 4 \mod 5 \\ x &\equiv 0 \mod 7 \end{aligned}$$

Théorème chinois des restes

Application – à vous

On organise les calculs dans un tableau comme suit:

i	m_i	a_i	M	u	v	x	$x \mod m_i$
1	2	1	2			2	1
2							
3							
4							

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'addtion.

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 ϵ, 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 $(u_n - v_n)_{n\in\mathbb{N}}$ convergent 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 β 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.
$\frac{237}{315} = 0.752380952380\overline{952380}$

Nombres réels La partie fractionnaire peut être apériodique.

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

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

$$e = 1 + \sum_{n=1}^{\infty} \frac{1}{n!} = 2.718281828459045235 36\ldots$$

$$\pi = 4\sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} = 3.14159265358979323 8\ldots$$

Ecriture 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 $e \neq 2047(11111111111)$

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

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

- si, exp biaisé $= 0$:

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

- Si $e = 255(11111111)$ (ou $e = 2047(11111111111)$): 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

Exemples

- $1 = 0\ 01111111\ 00000000000000000000000$
- $2 = 0\ 10000000\ 00000000000000000000000$
- $3 = 0\ 10000000\ 10000000000000000000000$
- $9 = 0\ 10000010\ 00100000000000000000000$
- $1/2 = 0\ 01111110\ 00000000000000000000000$
- $1/3 = 0\ 01111101\ 01010101010101010101010_{101010101\ldots}$
- $1/10 = 0\ 01111011\ 10110011001100110011001_{100110011\ldots}$

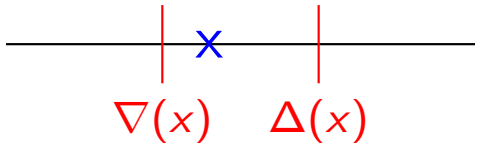
Valeurs extrêmes

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	7FEFFFFF FFFFFFFF	$1.7976931348623157e+308$
Normalisé positif minimum	00100000 00000000	$2.2250738585072014e-308$
Dénormalisé maximum	000FFFFF FFFFFFFF	$2.2250738585072009e-308$
Dénormalisé positif minimum	00000000 00000001	$4.9406564584124654e-324$
$-\infty$	FFF00000 00000000	$-\infty$
$+\infty$	7FF00000 00000000	$+\infty$
NAN	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 $\diamond(x \odot_{exact} y)$. 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, a 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 $=$ (= en C)
- inégalité $\neq$ (!= en C)
- supérieur $>$ (> en C)
- inférieur $<$ (< en C)
- incomparable ? (pas facilement disponible en C)

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

Dans le cas de comparaisons impliquant un NaN, les comparaisons $=$, $\neq$, $<$ et $>$ retournent faux, ? retourne vrai.
Dans le cas de l'égalité : si $x =$ NaN alors $x = x$ retourne faux, $x \neq x$ retourne faux mais $!(x = x)$ retourne vrai.

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 ∞: 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 ± 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 $9 + 1/10$
   $9 = 0\ 10000010\ 00100000000000000000000$
   $1/10 = 0\ 01111011\ 10110011001100110011001$
   $1/10 = 0\ 10000010\ 0000001\ 10110011001100110011001$
2. **Addition mantisses**
   $9 = 0\ 10000010\ 00100000000000000000000$
   $1/10 = 0\ 10000010\ 0000001101100110011001100110011001$
   $9 + 1/10 = 0\ 10000010\ 0010001101100110011001100110011001$
3. **Normalisation** suivant l'arrondi (ici inférieur)
   $9 + 1/10 = 0\ 10000010\ 00100011011001100110011$

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
  $9.999999999 + ((0.0000000004 + 0.0000000003) + 0.0000000003) = 10.00000000$

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
  $\frac{1}{3} = 3.333333333 * 10^{-1}$
  $3 * \frac{1}{3} = 9.999999999 * 10^{-1}$
  $1 - 3 * \frac{1}{3} = 1 * 10^{-10}$

Analyse d'erreur

Erreurs absolue et relative

- Souvent, dans l'analyse d'erreur, on note
  - y la quantité exacte, inconnue,
  - $\hat{y}$ la quantité inexacte, entachée d'erreur, calculée
- Il faut estimer et borner l'erreur entre $\hat{y}$ et y.
- On considère d'abord **l'erreur absolue** $\delta = \hat{y} - y$.
- Se tromper d'1 Euro quand on en a 1000000, c'est moins grave que quand on en a 10.
- On considère donc aussi **l'erreur relative**

$$\epsilon = \frac{\hat{y} - y}{y} = \frac{\hat{y}}{y} - 1.$$

Analyse d'erreur

Erreur globale

- En arithmétique flottante, toute opération peut provoquer une erreur.
- Ces erreurs élémentaires se combinent pour former une erreur globale.
- L'erreur globale est définie comme la différence (relative) du resultat qu'on aurait obtenu si tout avait été exact et le resultat obtenu par calcul flottant.
- La combinaison des erreurs élémentaires n'est pas simple (linéaire). Elles s'amplifient et se compensent les unes les autres.

Arithmétique d'intervalles

Au tableau, à la craie...

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 $(B_1, ..., B_n)$ 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 $V \in E$,

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

Vecteur colonne et vecteur ligne

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

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

A deux vecteurs associe un scalaire

Opérations sur les Vecteurs

Produit Scalaire : Algorithme

ProdScal(V, W, n)

Entrées $V \begin{pmatrix} v_1 & v_2 & \dots & v_n \end{pmatrix}$ et $W \begin{pmatrix} w_1 & w_2 & \dots & w_n \end{pmatrix}$
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, $(e_1, \ldots, e_n)$ une base de E et $(f_1, \ldots, f_m)$ base de F.

On note $\begin{pmatrix} a_{1,1} & a_{1,2} & \ldots & a_{1,n} \\ a_{2,1} & a_{2,2} & \ldots & a_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m,1} & a_{m,2} & \ldots & a_{m,n} \end{pmatrix}$ 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,

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

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

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

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

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

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

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

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

$$\begin{cases} P_{1,1} = T_1 + T_4 - T_5 + T_7 \\ P_{1,2} = T_3 + T_5 \\ P_{2,1} = T_2 + T_4 \\ P_{2,2} = T_1 - T_2 + T_3 + T_6 \end{cases}$$

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
- $S(n) = 7 \times S(\frac{n}{2}) + \alpha \times \left(\frac{n}{2}\right)^2$
- 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 : Réservation d'un espace de taille $m \times n$

Sur un tableau de vecteurs : Réservation d'un tableau d'adresses de taille m

Pour chaque adresse réservation 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 :

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

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

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

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

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

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

De la même façon que précédemment nous calculons,
$x_{n-2} = \frac{b'_{n-2}}{a_{n-2,n-2}}$, 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

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

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

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

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

Résolution d'un système linéaire

Cas d'une matrice triangulaire (4)

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

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

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

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

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

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

$$\rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} -\frac{97}{200} \\ -\frac{11}{20} \\ 3/4 \\ 1/2 \end{bmatrix}$$

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**

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

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

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 $A_{i,i}$ soit non nul. Il faut donc le vérifier au cours de l'algorithme.
  $\Rightarrow$ Pour ceci on échange avec une ligne non encore prise en compte comme pivot, de façon à avoir un pivot non nul.
  $\Rightarrow$ 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 $T_{i..n-1} := A_{i,i..n-1}; A_{i,i..n-1} := A_{j,i..n-1}; A_{j,i..n-1} := T_{i..n-1}$

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

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

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

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

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

if $(max < |A[j, i]|)$ 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**

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

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

retourne A et B

Triangularisation du système

Algorithme de Gauss avec choix sur la colonne(2)

$$\left[\begin{array}{cccc} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{array}\right], \left[\begin{array}{c} 32 \\ 23 \\ 33 \\ 31 \end{array}\right] \rightarrow \left[\begin{array}{cccc} 10 & 7 & 8 & 7 \\ 0 & 1/10 & 2/5 & 1/10 \\ 0 & 2/5 & \frac{18}{5} & \frac{17}{5} \\ 0 & 1/10 & \frac{17}{5} & \frac{51}{10} \end{array}\right], \left[\begin{array}{c} 32 \\ 3/5 \\ \frac{37}{5} \\ \frac{43}{5} \end{array}\right]$$

$$\left[\begin{array}{cccc} 10 & 7 & 8 & 7 \\ 0 & 2/5 & \frac{18}{5} & \frac{17}{5} \\ 0 & 1/10 & 2/5 & 1/10 \\ 0 & 1/10 & \frac{17}{5} & \frac{51}{10} \end{array}\right], \left[\begin{array}{c} 32 \\ \frac{37}{5} \\ 3/5 \\ \frac{43}{5} \end{array}\right] \rightarrow \left[\begin{array}{cccc} 10 & 7 & 8 & 7 \\ 0 & 2/5 & \frac{18}{5} & \frac{17}{5} \\ 0 & 0 & -1/2 & -3/4 \\ 0 & 0 & 5/2 & \frac{17}{4} \end{array}\right], \left[\begin{array}{c} 32 \\ \frac{37}{5} \\ -5/4 \\ \frac{27}{4} \end{array}\right]$$

$$\left[\begin{array}{cccc} 10 & 7 & 8 & 7 \\ 0 & 2/5 & \frac{18}{5} & \frac{17}{5} \\ 0 & 0 & 5/2 & \frac{17}{4} \\ 0 & 0 & -1/2 & -3/4 \end{array}\right], \left[\begin{array}{c} 32 \\ \frac{37}{5} \\ \frac{27}{4} \\ -5/4 \end{array}\right] \rightarrow \left[\begin{array}{cccc} 10 & 7 & 8 & 7 \\ 0 & 2/5 & \frac{18}{5} & \frac{17}{5} \\ 0 & 0 & 5/2 & \frac{17}{4} \\ 0 & 0 & 0 & 1/10 \end{array}\right], \left[\begin{array}{c} 32 \\ \frac{37}{5} \\ \frac{27}{4} \\ 1/10 \end{array}\right]$$

Résolution d'un système

Conditionnement [5]

Système initial :

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

Réduction de Gauss :

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

Résolution du système:

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

[5] (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:

$$A := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, B := \begin{bmatrix} 9.200000000 \\ -12.60000000 \\ 4.500000000 \\ -1.100000000 \end{bmatrix}$$

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 :

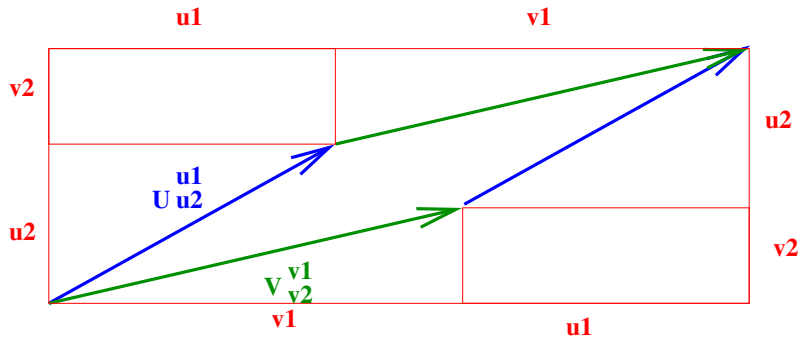
$$\text{cond}(A) = ||A||.||A^{-1}|| \text{ avec } ||A|| = \sup_{||v||=1} ||Av||$$

- 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

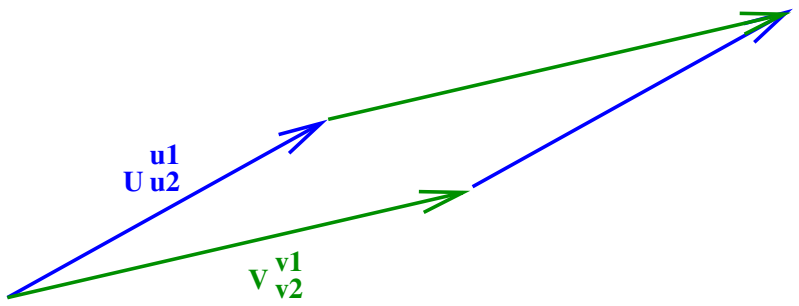


- Deux vecteurs en dimension 2 $V \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$ et $U \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$
- Surface définie par ces deux vecteurs : $S = |v_1 u_2 - v_2 u_1|$
- Surface signée :

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

Déterminant et surface

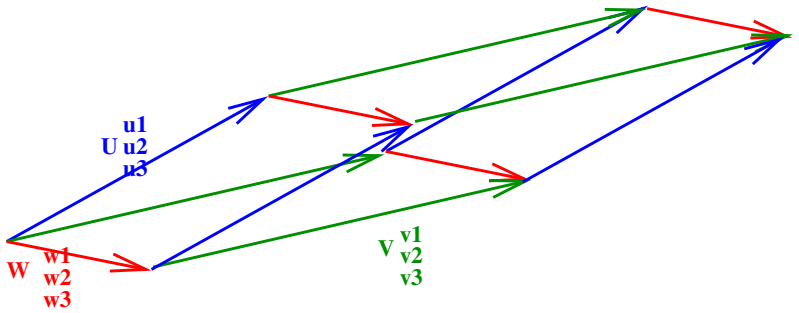
Interprétation trigonométrique



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

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

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

Déterminant de n vecteurs

E espace vectoriel de dimension n

- Le **déterminant de n vecteurs** $(V_1, ..., V_n)$ peut être définie comme une **forme n-linéaire alternée** det de E^n dans K telle que : $\det\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 1 \end{pmatrix} = 1$
- **Forme n-linéaire** :

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

- **Alternée** :
  $\det(V_1, ..., V_i, .., V_j, ..., V_n) + \det(V_1, ..., V_j, ..., V_i, ..., V_n) = 0$
  D'où, $\det(V_1, ..., V_i, .., V_i, ..., V_n) = 0$
- **Déterminant d'une matrice** : $\det(A)$, c'est le déterminant des vecteurs composant la matrice.

UPMC SORBONNE UNIVERSITÉS

Petits rappels

Identité et transposée

- **Identité** : la matrice $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ 0 & 0 & 0 & 1 \end{pmatrix}$ 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 :

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

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 :

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

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

- *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^{ieme} ligne et de sa j^{ieme} 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**

$B :=$Mineur$(A, i, 0, n)$;

$det := det + (-1)^{i \bmod 2} * a_{i,0} *$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'un 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** :

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

- 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) mod 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

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

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

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

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,

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

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

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

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 :

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{pmatrix}$$

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

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

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

- ReducTriangL(L, B, n) représente $\frac{n^2-n}{2}$ opérations (produits/divisions)
- 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 avions observé que:

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{pmatrix}$$

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

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

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

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

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 $j = 1, ..., k$ alors on effectue une décompostion LU de A puis nous effectuons k résolutions triangulaires inférieures et supérieures, soient 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)

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

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

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 :

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

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

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 $a_{i,k'} = a_{i,k}$)

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

Calcul de l'inverse

Via les cofacteurs : (3)

- Nous obtenons ainsi :

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

$$= \begin{pmatrix} \det A & 0 & \dots & 0 \\ 0 & \det A & \dots & 0 \\ 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & \det A \end{pmatrix}$$

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

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 :

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

$$= \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

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 $P, L, U :=$PermutLU(A, n);

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

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

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

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

Calcul de l'inverse

Résolution d'un système : Exemple

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

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

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

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)

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

Conditionnement

Rappel

- Nous savons que le conditionnement se calcule :

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

  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 ∞ 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 :

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

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 :

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

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

$$Y := \begin{bmatrix} 33.1000000000000014 \\ 6.96250000000000036 \\ -10.7840000000000007 \\ -7.17512953399999986 \end{bmatrix} \rightarrow \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0.955044509799999997 \\ 1.01290801200000002 \\ 1.01364985199999991 \\ 1.02729970299999995 \end{bmatrix}$$

Evaluation/Interpolation

Au tableau, à la craie...

Itération de Newton

résolution de l'équation $f(x) = 0$

Deux sites ont inspiré ce cours [6] [7]

[6] http://www.cse.illinois.edu/heath/scicomp/notes/chap05.pdf
[7] http://www.pi314.net/fr/algo_newton.php

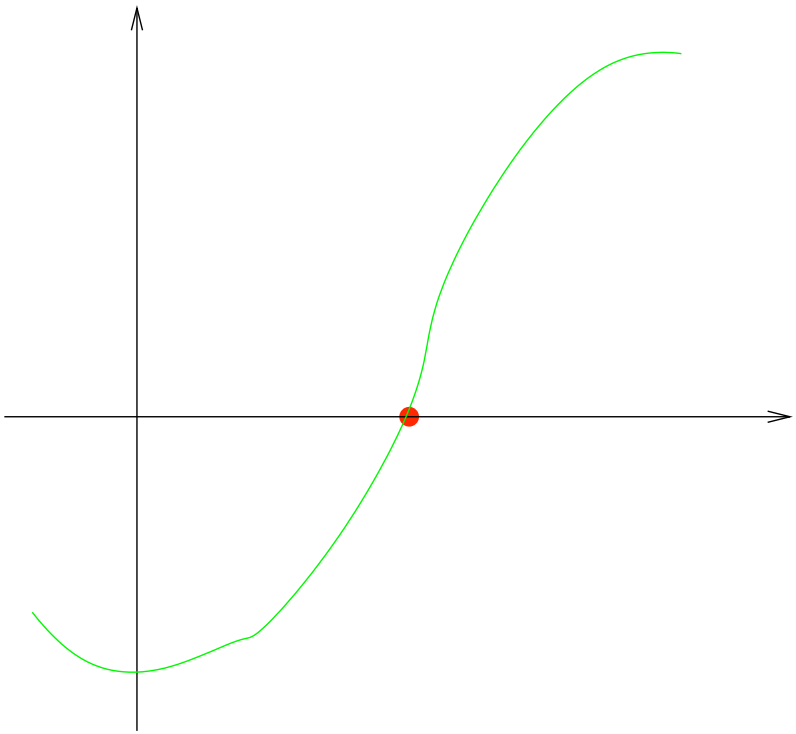
Trouver une solution à $f(x) = 0$

Présentation du problème

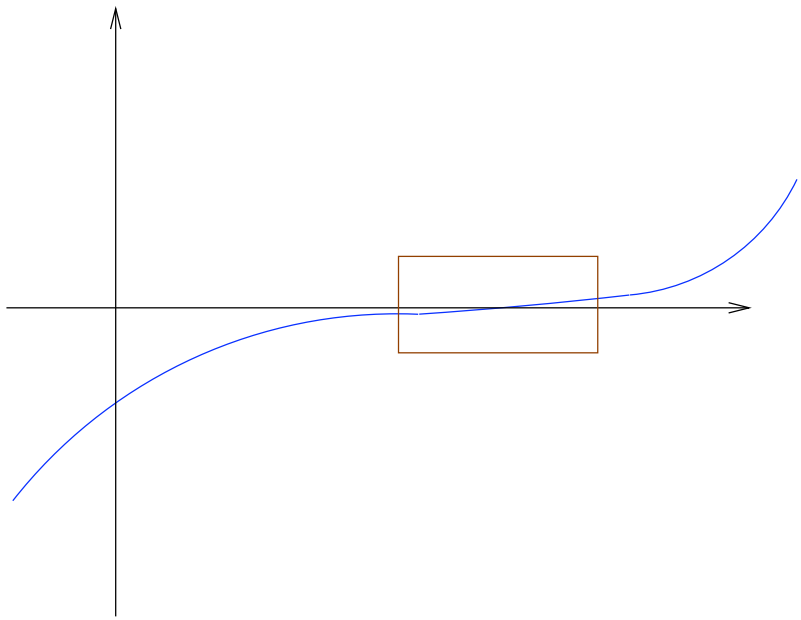
- Nous désirons trouver une solution de l'équation $f(x) = 0$ sous la forme d'un nombre "virgule flottante".
- Une équation de ce type peut avoir aucune, une ou plusieurs solutions
- Nous supposerons qu'elle en a au moins une que l'on cherchera à déterminer

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 :

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

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

Méthode de bissection

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/`

Méthode de bissection

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

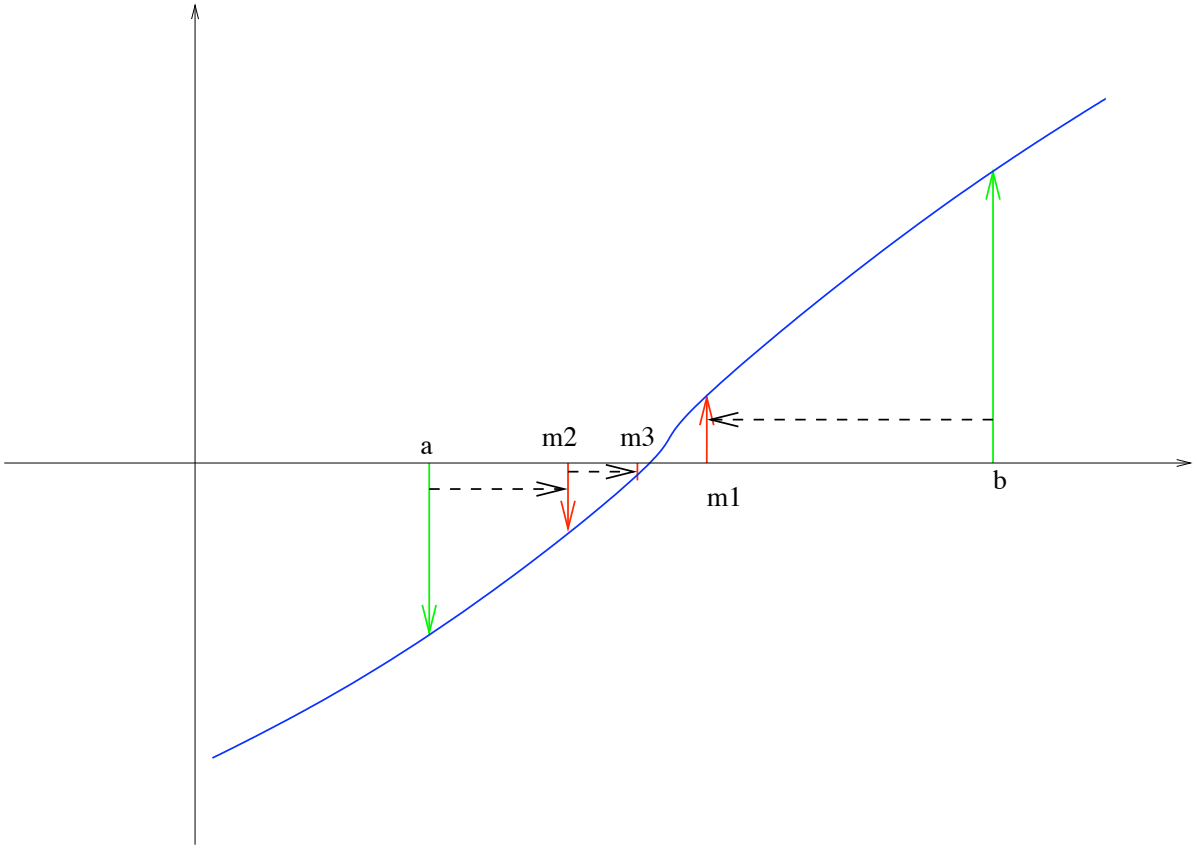


Figure: Méthode de la bissection avec *mi*, ... les milieux successifs

Méthode de bissection

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 ε

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 bissection

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 $\left\lceil \log_2(\frac{b-a}{\varepsilon}) \right\rceil$

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 $g(x) = \sqrt{x + 5}$...

Méthode du point fixe

Convergence

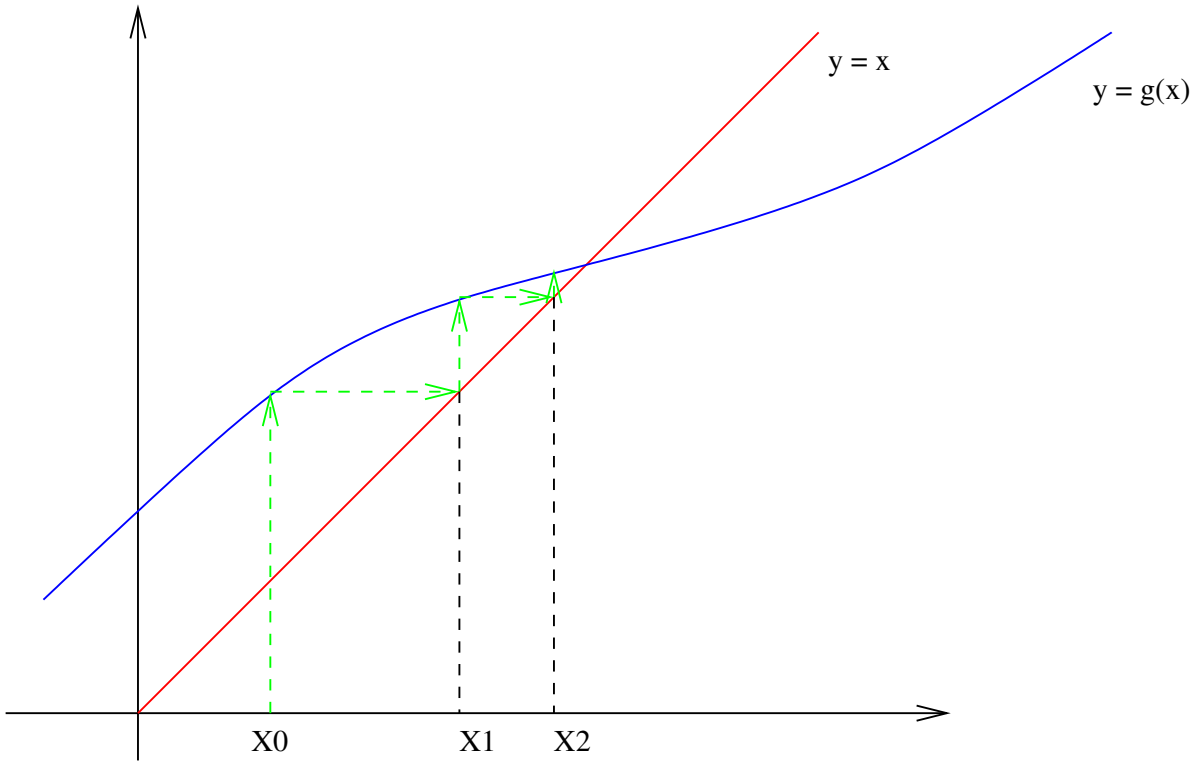


Figure: Méthode du point fixe avec Xi, ... les itérations successives

Méthode du point fixe

Divergence

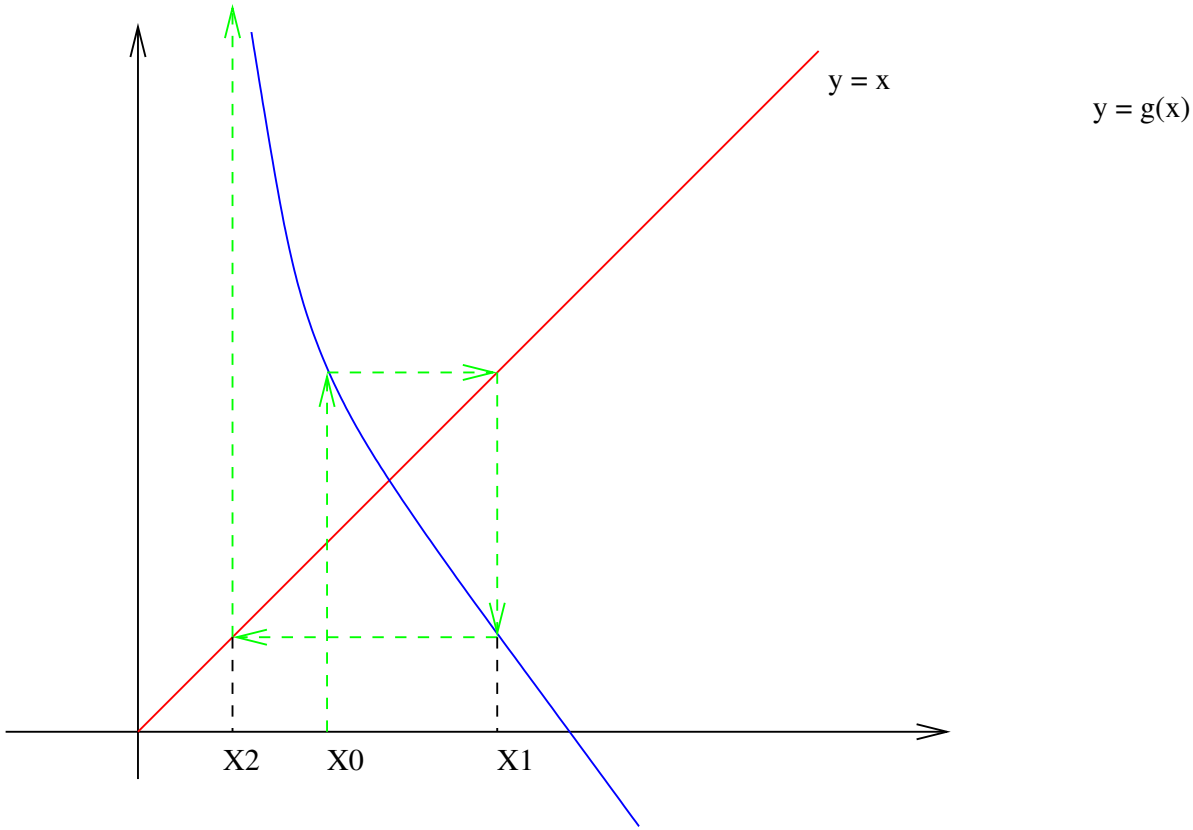


Figure: Méthode du point fixe avec *Xi*, ... les itérations successives

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/`

Méthode de Newton

Principe de la tangente

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

$$h_k(x) = f'(x_k).x + f(x_k) - f'(x_k).x_k$$

(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

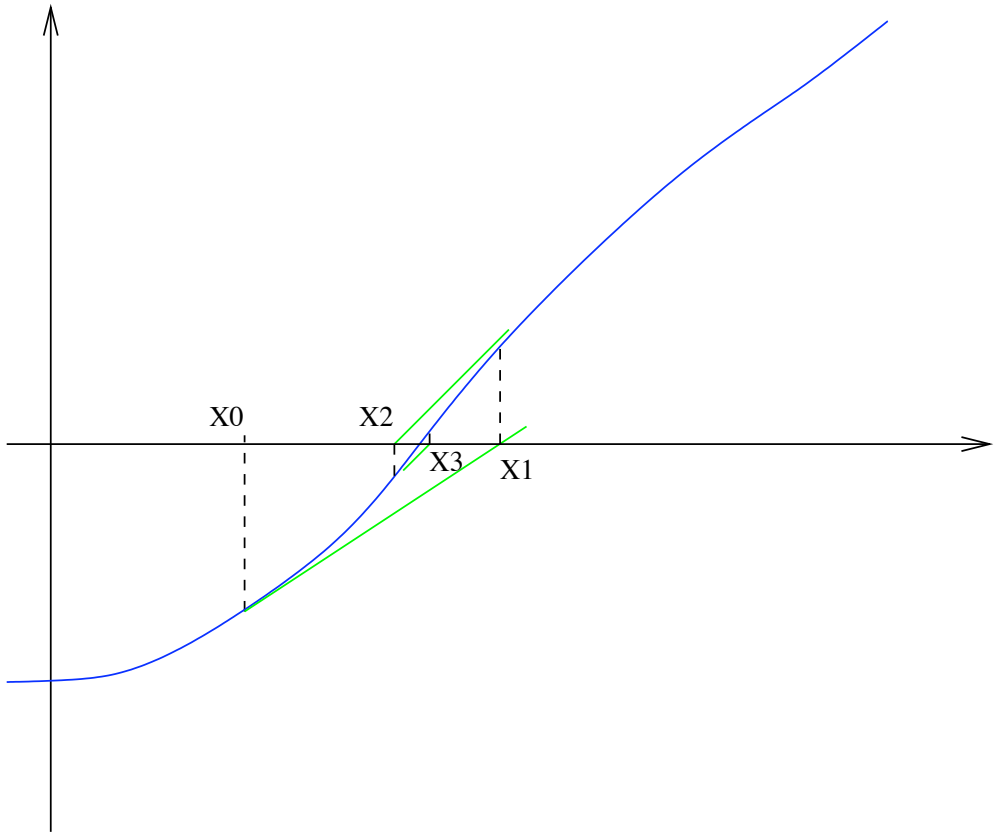


Figure: Méthode de Newton via les tangentes

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 $g'(x) = \frac{f(x).f''(x)}{(f'(x))^2}$
- Comme e est racine simple de $f(x)$, nous avons $g'(e) = 0$, ce qui assure une **convergence quadratique** *à condition de démarrer sur une petit voisinage de e*.

Animation :

`http://www.cse.illinois.edu/iem/nonlinear_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

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

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

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

$\sqrt{a}$ 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)

$P_0 \leftarrow P$

$P_1 \leftarrow P'$

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

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 $P_o(x), P_1(x), \ldots, P_n(x)$ 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.