Algorithmique numérique

Méthodes de calcul numérique

Gilles Marait & The IS104 team

12/04/2024

Bordeaux INP AQUITAINE

Plan

Bordeaux INP AQUITAINE

Plan

Bordeaux INP AQUITAINE

Présentation du module

- Algorithmique : science des algorithmes.
- Numérique : données pouvant être représentées par des nombres.
- Algorithmique numérique : conception d'algorithmes dont l'exécution passe par l'utilisation de variables numériques.

Modélisation d'un phénomène *réel* :

$\rightarrow$ équations du monde physique (équations de Maxwell en électromagnétique, équation de Boltzmann en thermodynamique, équations de Navier-Stokes en mécanique des fluides...) et leurs applications en biologie, électronique...

$\rightarrow$ équations le plus souvent à variables continues.

Présentation du module

Domaine du HPC : *High Performance Computing*, ou calcul haute-performance.
Considérer l'exécution d'algorithmes particulièrement complexes sur des superordinateurs (*supercomputers*).

Occigen, superordinateur du CINES (Montpellier)

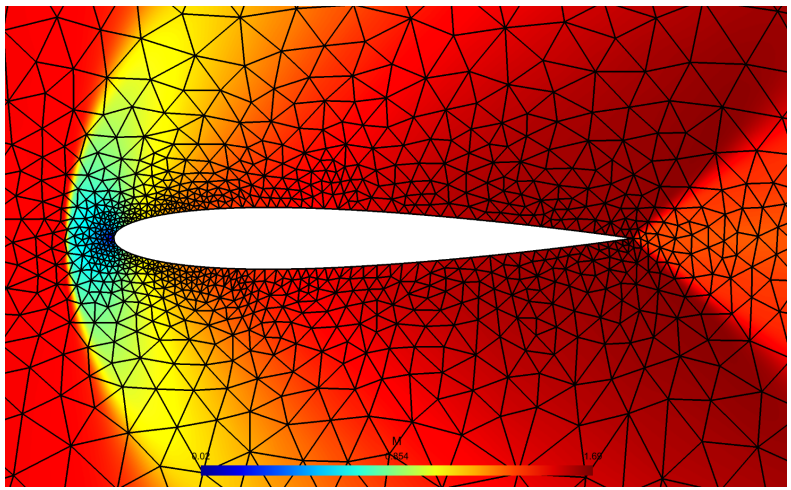
Voir le top 500 (www.top500.org), liste des 500 superordinateurs les plus performants. Actuellement (classement novembre 2020), le record est à 442 010 *Tflops*, soit $4,42 \times 10^{17}$ opérations sur des réels "flottants" par seconde.

Bordeaux INP AQUITAINE

Présentation du module

L'objet de ce domaine est de résoudre un problème physique le plus précisément et le plus efficacement possible.

Pour le cela, l'espace et le temps doivent être discrétisés, et des méthodes de calcul numérique adaptées et optimisées sont utilisées pour résoudre le problème.



Écoulement supersonic (équipe CARDAMOM, inria Bordeaux)

Parmi les domaines d'application, on peut noter :

aéronautique	astrophysique	médecine
finance	météorologie	physique des matériaux
acoustique	physique nucléaire	*machine learning*, *IA*. . .

Présentation du module

On va s'intéresser à la façon de passer d'un modèle mathématique (basé sur des équations) à un résultat numérique qui soit le plus effectif possible (et éventuellement exact, bien que cela ne soit pas forcément atteignable).

Étapes du travail (et critères d'évaluation) :

- conception d'un algorithme de résolution du problème
- conception d'un programme à partir de cet algorithme
- étude de la validité des calculs effectués : avoir un regard critique sur le résultat
- étude de la complexité des calculs
- éventuellement retour à la conception pour optimiser

Plus généralement, il s'agit d'adopter une démarche scientifique sur la résolution du problème, sur le plan théorique et technique.

Présentation du module

I Méthodes de calcul numérique
II Résolution de systèmes linéaires
III Calcul des éléments propres
IV Équations non linéaires
V Méthodes d'interpolation et d'intégration
VI Équations différentielles

Plan

Les chiffres, c'est pas une science exacte, figurez-vous.

Chevalier Karadoc - Kaamelott

Représentation des nombres en machine

Pour le calcul numérique, on a besoin de manipuler des nombres entiers et réels.

Quelles contraintes a-t-on sur les nombres et les opérations qu'on manipule numériquement par rapport aux nombres et opérations mathématiques ?

Représentation des nombres en machine

Dans les composants électroniques, la présences de 2 tensions distinctes permet le stockage de nombres binaires. Une tension représentera le 1, et l'autre le 0.

Remarque

Dans la nature, on peut penser à l'ADN et l'ARN qui stockent l'information sous forme de chaîne de 4 bases nucléiques (A, C, G, T / U).

Représentation des nombres en machine

Une suite de n bits peut représenter 2^n nombres différents.

bits de poids fort $\rightarrow$ 01001101 $\leftarrow$ bits de poids faible

Octet

On regroupe habituellement les chiffre binaire (bit) par 8, pour obtenir un octet (*byte* en anglais).

Un octet peut donc représenter $2^8 = 256$ nombres entiers.

Nombres entier non signés

Pour stocker un nombre entier non signé, on va simplement choisir sa représentation mathématiques en base 2. Avec n bits, on peut donc coder les nombres allant de 0 à $2^n - 1$.

Sur un octet :

Nombre décimal	Codage binaire
0	0000 0000
1	0000 0001
2	0000 0010
...	...
255	1111 1111

Nombres entiers signés

On va coder sur n bits les entiers de -2^{n-1} à $2^{n-1}-1$.
Afin de conserver la même opération d'addition en binaire (modulo 2^n), on utilise la technique du complément à 2.

Complément à 2

Pour stocker $-k$ où k est un entier positif, on procède ainsi :

- Mettre 0 sur le bit de poids fort, puis prendre k sur $n-1$ bits suivants
- opération miroir sur les bits
- ajout de 1

Remarque

Le bit de poids fort est un entier signé et donc toujours un 1 s'il est négatif, et 0 s'il est positif.

Nombres entiers signés

Exemple avec $n = 6$ et $k = 15$, pour représenter -15

Étape	bits	Entier signé	Entier non signé
k sur $n-1$ bits	(0)01111	15	15
Opéation miroir	110000	-16	$= 48(64)$
Ajout de 1	110001	-15	$= 49(64)$

Donc -15 est serait stocké 110001 en machine sur 6 bits.
Cette représentation interprétée comme un entier signé vaut :

$$-15(2^n) = -15(64) = 49$$

Nombres entiers signés

L'avantage de ce codage est que l'addition est la même pour entiers signés et non signés (modulo 2^n).
Exemple, avec $n = 6$:

	bits	entier non signé	entier signé
	010110	22	22
+	110001	49	-15
=	000111	7 (64)	7 (64)

Nombres réels

Plus difficile : comment représenter les nombres réels en machine ?
Mathématiquement, $\mathbb{R}$ est continu : entre 2 réels x et y tels que $x < y$, on peut toujours trouver un réel z tel que $x < z < y$.

Il va être impossible de reproduire cette propriété avec des nombres stockés comme des entiers sur la machine. Quelle méthode a été choisie pour nos ordinateurs, pour passer du continu au discret ? Quelles sont ses limites ?

Référence

What every computer scientist should know about floating-point arithmetic, David Goldberg

Nombres réels

Norme IEEE754 (1985) pour les nombres "flottants".
On décompose le réel x de la manière suivante :

$$x = s.(x_1, x_2 x_3 \ldots x_{p-1} x_p).2^{e_1 e_2 \ldots e_q}$$

- s : bit de signe (si le réel est signé)
- $x_1, x_2 x_3 \ldots x_{p-1} x_p$: la mantisse de p bits
- $e_1 e_2 \ldots e_q$: un exposant sur q bits

Type (C)	$s+p+q$	$(s+)p$	q
float	32	24	8
double	64	53	11
long double	80	65	15

Nombres réels

Il existe quelques nombres spéciaux :

- $\pm\infty$: e tout à 1, x tout à 0
- NaN (*Not a Number*) : e tout à 1, x non tous nuls

Avec des régles spéciales.
Par exemple pour tout y :

$$NaN\ op\ y = NaN$$

avec $op = +, -, *, /$

Exemple de non représentablilité des nombres

En C

```
float f = 1e8 + 1;
printf("%f", f);
```

On obtient :

```
100000000.000000
```

Car $10^8 > 2^{24}$ et la mantisse est sur $p = 24$ bits.

Représentation des nombres réels

Soit $x \in \mathbb{R}$. On note $rp(x)$ la meilleure représentation machine de x par un flottant.

On appelle p le nombre de bits de la mantisse la précision.
Si x s'écrit en binaire :

$$x_1, x_2 \ldots x_i x_{i+1} \ldots$$

(On néglige l'exposant car on étudie l'erreur relative, $x_1 = 1$).

$$rp(x) = \begin{cases} x_1, x_2 \ldots x_p & \text{si } x_{p+1} = 0 \\ x_1, x_2 \ldots x_p + 2^{-p} & \text{si } x_{p+1} = 1 \end{cases}$$

On note $\mathbb{F}$ l'ensemble (fini) des nombres ayant une représentation exacte en flottants. Si $x \in \mathbb{F}$, alors $x = rp(x)$.

Bordeaux INP AQUITAINE

Erreur relative et précision machine

Erreur bornée relative à x

$$\left|\frac{x - rp(x)}{x}\right| \leq \frac{2^{-(p+1)}}{2^{-1}} = 2^{-p}$$

On appelle $\epsilon = 2^{-p}$ la précision machine.
On peut notamment écrire la représentation de x :

$$\boxed{rp(x) = (1 + \theta)x}$$

avec $|\theta| \leq \epsilon$.

Remarque

En base b, on aurait :

$$\left|\frac{x - rp(x)}{x}\right| \leq \frac{b^{-p}/2}{b^{-1}} = \frac{b^{-p+1}}{2} = \epsilon_b$$

Précision machine des différents types

La précision machine dépend donc p, la taille de la mantisse. Pour les différents types usuels, on a :

Type (C)	Nombre de bits	p	ϵ
float	32	23	$2^{-23} \approx 10^{-6.7}$
double	64	52	$2^{-52} \approx 10^{-15.6}$
long double	80	64	$2^{-64} \approx 10^{-19.3}$

Plan

Bordeaux INP AQUITAINE

Plan

Problèmes d'approximation

Une fois prise en compte la précision limitée des nombres réels en machine, il reste à étudier les erreurs générées par les opérations machine.

Opérations élémentaires

On note $+_m$ et $\times_m$ respectivement l'addition et la multiplication machine.
$\forall (x, y) \in \mathbb{F}^2$, on a :

$$x +_m y = rp(x + y) = (x + y)(1 + \theta_1)$$

$$x \times_m y = rp(x \times y) = (xy)(1 + \theta_2)$$

avec $|\theta_1|$ et $|\theta_2| \leq \epsilon$

Problèmes d'approximation

Exemple

Avec $b = 10$, $p = 4$, $\epsilon = 5.10^{-4}$:

$$\begin{array}{llllll} 0.1056 & +_m & 0.4985\times10^{-5} & = & 0.1056 & \text{err. rel. } 4.7\times10^{-4} \\ 0.1201 & +_m & 0.1495\times10^{-1} & = & 0.1351 & \text{err. rel. } 3.7\times10^{-4} \\ 0.3456 & \times_m & 0.2921 & = & 0.1009 & \text{err. rel. } 4.9\times10^{-4} \end{array}$$

En général, on obtient des erreurs relatives de l'ordre de ϵ.

Problèmes d'approximation (addition)

Exemple

Pour l'addition, on voit facilement le phénomène :

$$\begin{array}{cccccccccccc} & & 1 & 1 & 0 & 1 & 1 & 0 & & & & \\ + & & & & & & 1 & 1 & 0 & 0 & 0 & 1 \\ \hline & 1 & 1 & 1 & 0 & 0 & 1 & X & X & X & X \end{array}$$

Les 4 digits X sont perdus. Plus généralement, avec

$$x = 0.x_1 \ldots x_p \times 2^e$$

$$y = 0.y_1 \ldots y_p \times 2^f$$

en faisant $x +_m y$, on perd les $min(|e - f|, p)$ dernières décimales.

Problèmes d'approximation (addition)

Exemple

L'addition est commutative, mais plus associative. Exemple avec 6 décimales :

$$a = 0.233712 \times 10^{-6}$$
$$b = 0.336784 \times 10^{0} \qquad a +_m b = 0.36784 \times 10^{0}$$
$$c = -0.336778 \times 10^{0} \qquad b +_m c = 0.60000 \times 10^{-5}$$

$$a + b + c = 0.6233712 \times 10^{-5} \quad \text{(Résultat exact)}$$
$$(a +_m b) +_m c = 0.60000 \times 10^{-5} \quad \text{Err. rel. } 3.7 \times 10^{-2}$$
$$a +_m (b +_m c) = 0.62337 \times 10^{-5} \quad \text{Err. rel. } 1.9 \times 10^{-6}$$

Pour l'addition, l'ordre des opérations peut donc être important !

Bordeaux INP AQUITAINE

Problèmes d'approximation (soustraction)

Exemple

La soustraction de quantités équivalentes est aussi un cas difficile.

	1	0	0	0	0	1	0	0	
−		1	1	1	1	0	1	1	1
=	(0)	0	0	0	1	0	0	0	1
	X	X	X	X					

Les 4 digits X sont perdus lors de l'annulation et le résultat n'est plus que sur 5 bits.

Propagation des erreurs de calcul

Un réel x est connu avec une certaine précision Δx.

A moins d'être représentable de manière exacte (auquel cas $\Delta x = 0$), la valeur $x + \Delta x \in \mathbb{F}$ qu'on manipule est le résultat d'un calcul précédemment fait avec une précision donnée.

Voyons comment ces erreurs ce propagent lorsqu'on poursuit les calculs avec $+_m$ et $\times_m$ sur ces valeurs.

Rappel

$$x +_m y = rp(x + y) = (x + y)(1 + \theta_1)$$

$$x \times_m y = rp(x \times y) = (xy)(1 + \theta_2)$$

$$|\theta_1| \text{ et } |\theta_2| \leq \epsilon$$

Dans les calculs suivants, on négligera les termes du second ordre (en $\theta \Delta x$).

Propagation des erreurs de calcul (addition)

Addition

$$\begin{aligned}(x+\Delta x)+_m(y+\Delta y) &= (x+y+\Delta x+\Delta y)(1+\theta)\\ &= (x+y)+(\Delta x+\Delta y+\theta(x+y))\end{aligned}$$

- Erreur global : $\Delta(x+y) = \Delta x+\Delta y+\theta(x+y)$
- Erreur relative : $\frac{\Delta(x+y)}{x+y} = \frac{\Delta x+\Delta y}{x+y}+\theta$

Observations

- Les erreurs globales s'ajoutent
- L' erreur relative est grande si $x \approx -y$

Propagation des (multiplication)

Multiplication

$$\begin{aligned}(x+\Delta x)\times_m(y+\Delta y) &= (xy+x\Delta y+y\Delta x)(1+\theta)\\ &= (xy)+(x\Delta y+y\Delta x+xy\theta)\end{aligned}$$

- Erreur global : $\Delta(x\times y)=x\Delta y+y\Delta x+xy\theta$
- Erreur relative : $\frac{\Delta(x\times y)}{xy}=\frac{\Delta x}{x}+\frac{\Delta y}{y}+\theta$

Observations

Cette fois-ci, ce sont les erreurs relatives qui s'ajoutent.

Plan

Notion de conditionnement

Les erreurs introduites par la machine sont rarement limitantes. En double précision, on a $\epsilon \approx 10^{-16}$, ce qui est souvent largement plus précis que les données en entrée et la précision attendue en sortie !
Il est d'ailleurs souvent illusoir de pouvoir espérer atteindre une telle précision pour certains problèmes. La notion de conditionnement permet de quantifier cette propriété.

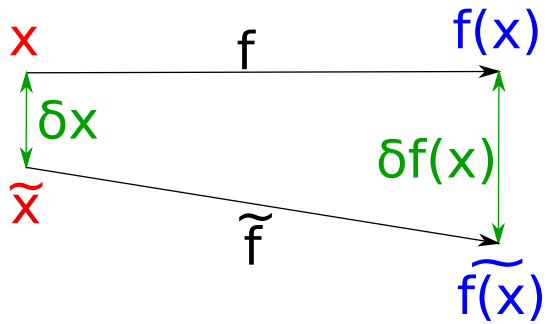
Remarque

Depuis quelques années, on utilise de plus en plus des flottant "demi-précision" sur 16 bits (norme IEEE 754-2008), et on fabrique du matériel spécifique pour les manipuler, afin de gagner en temps et énergie. Ces flottants sont particulièrement utiliés dans le machine learning et les réseaux de neurones, où la précision requise est faible.

Précision	Nombre de bits	p	ϵ
Demi	16	10	$2^{-10} \approx 10^{-3}$
Simple	32	23	$2^{-23} \approx 10^{-6.7}$
Double	64	52	$2^{-52} \approx 10^{-15.6}$
Étendue	80	64	$2^{-64} \approx 10^{-19.3}$

Bordeaux INP AQUITAINE

Notion de conditionnement



On veut résoudre un problème mathématiques avec x en entrée et $f(x)$ en sortie.

x n'étant généralement pas exact (par exemple venant d'une mesure physique à précision limitée, ou tout simplement sans représentation exacte en machine), on utilisera $\tilde{x} = x + \delta x$ en entrée de notre algorithme.

L'algorithme codé $\tilde{f}$ nous donne un résultat en sortie $f(\tilde{x})$.
Dans quelle mesure peut-on espérer avoir $f(\tilde{x})$ proche de $f(x)$?

Bordeaux INP AQUITAINE

Notion de conditionnement

On dit qu'un problème est bien conditionné (*well-conditioned*) si on peut espérer un résultat $f(\tilde{x})$ proche de $f(x)$ à partir d'une imprécision sur l'entrée x. Ci ce n'est pas le cas, on parle de problème mal conditionné (*ill-conditioned*).

Exemple de problème mal conditionné : la météo, les problèmes physiques chaotiques (problème à 3 corps. . .)

Notion de conditionnement

Plus formellement, en considérant δx infinitésimal et en écrivant $\delta f = f(x + \delta x) - f(x)$, on définit :

Le conditionnement absolu

$$\hat{\kappa} = \sup_{\delta x} \frac{||\delta f||}{||\delta x||}$$

Le conditionnement relatif

$$\kappa(x) = \sup_{\delta x} \frac{||\delta f(x)||/||f(x)||}{||\delta x||/||x||}$$

On utilisera celui-ci lorsqu'on parle de conditionnement en général.

Référence

Numerical linear algebra Lloyd N. Trefethen; David Bau, III

Conditionnement pour une fonction C^1

Prenons par exemple le cas de f, une fonction C^1.

$$\delta f(x) = f(x) - f(x + \delta x)$$

donc

$$\frac{\delta f(x)/f(x)}{\delta x/x} = \frac{x}{f(x)} \times \frac{f(x) - f(x + \delta x)}{\delta x}$$

Ce qui tend, quand $\delta x \to 0$, vers $\frac{xf'(x)}{f(x)}$
On obtient :

$$\boxed{\kappa(x) = \left| \frac{xf'(x)}{f(x)} \right|}$$

Propagation des erreurs pour une fonction C^1

En reprenant le calcul de propagation des erreurs, cette fois sur une fonction f C^1, en appelant f_m le f machine :

$$\begin{aligned} f_m(x+\Delta x) &= (f(x)+\Delta x f'(x)+o(\Delta x))(1+\theta) \\ &= f(x)+\Delta x f'(x)+\theta f(x) \end{aligned}$$

- Erreur global : $\Delta(f(x)) = \Delta x f'(x) + \theta f(x)$
- Erreur relative :

$$\frac{\Delta f(x)}{f(x)} = \frac{\Delta x f'(x)}{f(x)} + \theta = \underbrace{\frac{\Delta x}{x}}_{\text{err. rel.}} \times \underbrace{\frac{x f'(x)}{f(x)}}_{\kappa(x)} + \theta$$

L'erreur relative sur x est donc multipliée par le conditionnement en x.

Plan

Système linéaire de Wilson

Exemple : système linéaire de Wilson

Regardons le problème de la résolution du système linéaire $AX = Y$ suivant.

$$\underbrace{\begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix}}_{A} \cdot \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}}_{X} = \underbrace{\begin{pmatrix} 32 \\ 23 \\ 33 \\ 31 \end{pmatrix}}_{Y} \qquad \underbrace{\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}}_{A^{-1}Y}$$

$$\underbrace{\begin{pmatrix} 25 & -41 & 10 & -6 \\ -41 & 68 & -17 & 10 \\ 10 & -17 & 5 & -3 \\ -6 & 10 & -3 & 2 \end{pmatrix}}_{A^{-1}}$$

Pas de propriété particulière à première vue : A est inversible, de déterminant 1, sans coefficients trop importants dans l'inverse.

Système linéaire de Wilson

Exemple : système linéaire de Wilson

Si on modifie légèrement le membre droit Y :

$$\underbrace{\begin{pmatrix}32\\23\\33\\31\end{pmatrix}}_{Y} \rightarrow \underbrace{\begin{pmatrix}32.1\\22.9\\33.1\\30.9\end{pmatrix}}_{\tilde{Y}} \text{ on obtient : } \underbrace{\begin{pmatrix}9.2\\-12.6\\4.5\\1.1\end{pmatrix}}_{A^{-1}\tilde{Y}} \text{, très différent de } \underbrace{\begin{pmatrix}1\\1\\1\\1\end{pmatrix}}_{A^{-1}Y}$$

Système linéaire de Wilson

Exemple : système linéaire de Wilson

De même, si on modifie légèrement la matrice A :

$$\underbrace{\begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix}}_{A} \rightarrow \underbrace{\begin{pmatrix} 10 & 7 & 8.1 & 7.2 \\ 7.08 & 5.04 & 6 & 5 \\ 8 & 5.98 & 9.89 & 9 \\ 6.99 & 4.99 & 9 & 9.98 \end{pmatrix}}_{\tilde{A}}$$

on obtient : $\underbrace{\begin{pmatrix} -81 \\ 137 \\ -34 \\ 22 \end{pmatrix}}_{\tilde{A}^{-1}Y}$, vraiment très différent de $\underbrace{\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}}_{A^{-1}Y}$

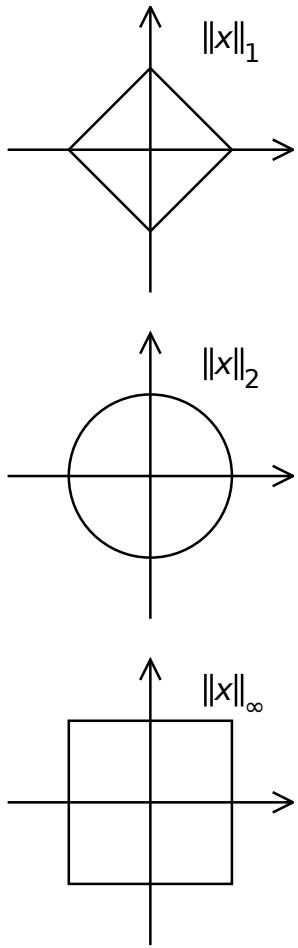
Rappel sur les normes vectorielles

Avec un vecteur $x = \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix}$

- $||x||_1 = |x_1| + |x_2| + ... + |x_n|$
- $||x||_2 = \sqrt{x_1^2 + x_2^2 + ... + x_n^2}$
- $\forall p \in \mathbb{N}^* : ||x||_p = \sqrt[p]{|x_1|^p + |x_2|^p + ... + |x_n|^p}$
- $||x||_\infty = \lim_{p \to \infty} ||x||_p = max(|x_1|, |x_2|, ..., |x_n|)$

Rappel sur les normes vectorielles

Visuellement, l'ensemble des vecteurs de norme 1 dans les différentes normes :



Source : wikipedia, article *Norm (mathematics)*

Norme matricielle subordonnée

On définit la norme matricielle subordonnée à une norme vectorielle $||.||$. Pour une matrice A :

$$|||A||| = sup\{||Ax||, ||x|| = 1\} = sup_{x \neq 0} \left\{ \frac{||Ax||}{||x||} \right\}$$

Autrement dit, $|||A|||$ est la norme de $||Ax||$ la plus grande possible en prenant x de norme 1.

Norme matricielle subordonnée

En particulier, cette norme a la bonne propriété, pour toutes matrices A et B pouvant être multipliées :

$$\boxed{|||AB||| \leq |||A|||.|||B|||}$$

Preuve :
Soit x un vecteur non nul. On peut remarquer à partir de la définition $|||A|||.||x|| \geq ||Ax||$, d'où :

$$||(AB)x|| = ||A(Bx)|| \leq |||A|||.||Bx|| \leq |||A|||.|||B|||.||x||$$

$$\frac{||ABx||}{||x||} \leq |||A|||.|||B|||$$

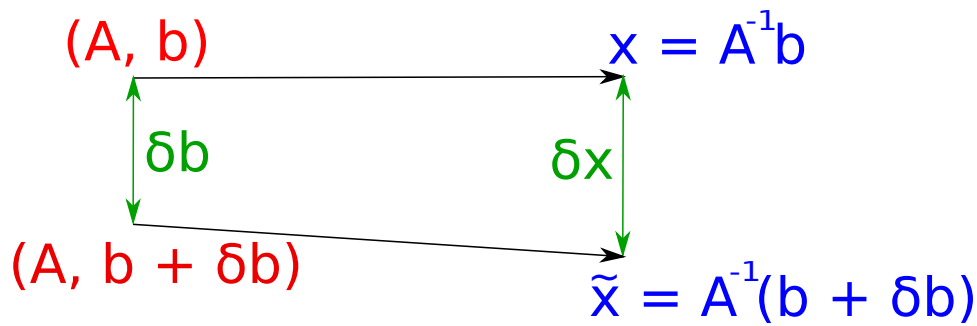
En prenant le *sup* pour x de cette expression, on a bien la propriété à démontrer.

Conditionnement d'une matrice

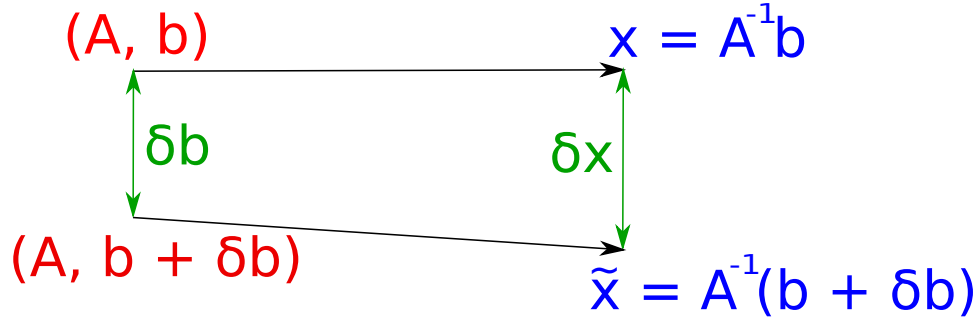
On peut maintenant définir le conditionnement d'une matrice.

Prenons comme problème le cas de la résolution d'un système linéaire $Ax = b$, avec A une matrice carrée, b un second membre donné et x le vecteur solution à trouver. (Ce qui est d'ailleurs le sujet du chapitre 2 !)

On a donc en entrée A et b, en sortie la solution x. Pour simplifier, imaginons avoir simplement une erreur sur δb sur b.



Conditionnement d'une matrice



$$\delta x = x - \tilde{x} = A^{-1}b - A^{-1}(b+\delta b) = -A^{-1}\delta b$$

$$\kappa = \frac{||\delta x||/||x||}{||\delta b||/||b||} = \frac{||A^{-1}\delta b||/||A^{-1}b||}{||\delta b||/||b||}$$

$$= \frac{||A^{-1}\delta b||}{||A^{-1}b||} \times \frac{||b||}{||\delta b||} = \frac{||A^{-1}\delta b||}{||\delta b||} \times \frac{||A(A^{-1}b)||}{||A^{-1}b||}$$

$$\leq |||A^{-1}|||.|||A|||$$

Conditionnement d'une matrice

On définit donc le conditionnement d'une matrice A :

$$\kappa(A) = |||A^{-1}|||.|||A|||$$

Remarque

Pour le système de Wilson vu précédemment, on avait $\kappa(A) \approx 3000$.

Remarque

Plus généralement, le conditionnement de A mesure le conditionnement d'autres problèmes :

- la résolution de $Ax = b$, en fixant b et perturbant A
- le calcul du produit matrix vecteur Ax, en perturbant A, x ou les deux
- ...

Conditionnement d'une matrice réelle inversible

Si A est une matrice réelle inversible, alors ${}^tAA$ est symétrique, donc diagonalisable dans une base orthonormée.
Si on note ses valeurs propres dans l'ordre croissant $0 \leq |\lambda_1| \leq |\lambda_2| \leq ... \leq |\lambda_n|$, alors le conditionnement en $||.||_2$:

$$\boxed{\kappa_2 = \frac{|\lambda_n|}{|\lambda_1|}}$$

Remarque

Les matrices orthogonales ont un conditionnement de 1 (car ${}^tOO = I$), elles sont donc particuièrement stables numériquement.

Plan

Principes généraux

Intéressons-nous aux principes généraux pour éviter tant que possible la propagation des erreurs dans nos calculs machine.

Principes généraux - règle 1

Règle générale : éviter d'augmenter les erreurs relatives sur les variables.

Règle 1

Éviter de soustraire deux quantités équivalentes.

Example

Racines d'un polynôme : $ax^2 + bx + c$, avec $a \neq 0$.

$$\Delta = b^2 - 4ac \text{ , et } x = \frac{-b \pm \sqrt{\Delta}}{2a} \text{ si } \Delta > 0$$

Principes généraux - règle 1

Example

Racines d'un polynôme : $ax^2 + bx + c$, avec $a \neq 0$.

$$\Delta = b^2 - 4ac \text{ , et } x = \frac{-b \pm \sqrt{\Delta}}{2a} \text{ si } \Delta > 0$$

Supposons $|ac| << b^2$, donc $\sqrt{\Delta} \approx b$.
Par exemple $a = c = 1$ et $b = -1634$, avec 8 décimales.

$$\sqrt{\Delta} = 1633.9988$$
$$x_1 = 1633.9944$$
$$x_2 = 6 \times 10^{-4}$$

1 seul chiffre significatif !

On peut faire mieux : en utilisant la formule $x_1 x_2 = \frac{c}{a} = 1$, on trouve $x_2 = \frac{1}{x_1} = 6.1199533 \times 10^{-4}$ (8 chiffres significatifs).

Principes généraux - règle 1

Exemple 2 : calcul d'une fonction dérivée

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

- Problème 1 : f(x + h) et f(x) sont des quantités équivalentes
- Problème 2 : division de ce nombre (imprécis) par h, donc deux nombres proches de 0

Astuce : utiliser un h ayant une représentation exacte en machine (une puissance de 2 par exemple)

Principes généraux - règle 2

Règle 2

Éviter de sommer des termes d'ordre de grandeur trop différents.
Par exemple : sommer les termes de valeur absolue les plus petits en premier.

Exemple : ordre des addition

On veut calculer la somme des $\frac{1}{k}$ pour k allant de 1 à n. On considère l'ordre naturel N et l'ordre opposé O.

n	N (4 déc.)	N (6 déc.)	O (4 déc.)	O (6 déc.)	Exact
10	2.929	2.92897	2.929	2.92897	2.92896825
50	4.497	4.49921	4.500	4.49921	4.49920534
100	5.186	5.18739	5.187	5.18737	5.18737752
1000	7.448	7.48545	7.485	7.48546	7.48547086
5000	8.448	9.09437	9.274	9.09448	9.09450885
10000	8.448	9.78716	9.449	9.78753	9.78760604
100000	8.448	10.7625	9.449	12.0902	12.09014613

Bordeaux INP AQUITAINE

Principes généraux - règle 3

Règle 3

Éviter d'évaluer des fonctions en des valeurs où elles ont un conditionnement élevé.

Exemple

On considère la suite (U_n) définie par $U_0 = 2$ et $U_{n+1} = log|U_n|$.

n	Calcul de U_n			
	4 déc.	7 déc.	10 déc.	100 déc.
1	0.6931	0.6931472	0.6931471806	0.6931471806
5	5.810	5.595352	5.595484996	5.5954851833
10	0.5773	0.7040199	0.7039347036	0.7039345855
15	0.4033	1.124368	1.126695313	1.1266985498
20	0.1629	1.305218	1.266159318	1.2661060564
24	0.6591	1.273670	1.001307399	1.0009714391
25	0.4169	0.2419025	0.001306545098	0.0009709675
26	0.8749	1.419221	6.640368955	6.9372175459
30	0.3572	3.029780	0.8006809852	0.8822116404

Principes généraux - règle 3

Exemple

n	Calcul de U_n			
	4 déc.	7 déc.	10 déc.	100 déc.
1	0.6931	0.6931472	0.6931471806	0.6931471806
5	5.810	5.595352	5.595484996	5.5954851833
10	0.5773	0.7040199	0.7039347036	0.7039345855
15	0.4033	1.124368	1.126695313	1.1266985498
20	0.1629	1.305218	1.266159318	1.2661060564
24	0.6591	1.273670	1.001307399	1.0009714391
25	0.4169	0.2419025	0.001306545098	0.0009709675
26	0.8749	1.419221	6.640368955	6.9372175459
30	0.3572	3.029780	0.8006809852	0.8822116404

On voit que le calcul de U_{24} est crucial, car très proche de 1.
Regardons le conditionnement de $f : x \to log|x|$.

$$\kappa_f(x) = \frac{xf'(x)}{f(x)} = \frac{1}{|log(x)|}$$

L'erreur relative est donc multipliée par un facteur $\frac{1}{|log(x)|}$, qui explose quand x est proche de 1.

Bordeaux INP AQUITAINE

Principes généraux - règle 4

Règle 4

Lors d'un calcul itératif, éviter d'itérer sur des opérations connues pour augmenter l'erreur relative sur les variables.

Exemple : propagation des erreurs dans le pivot de Gauss

Regardons l'algorithme classique de résolution de système linéaire sur le système suivant, d'inconnue $X = {}^t(x_1 x_2 x_3)$.

$$\begin{pmatrix} 20 & 15 & 10 \\ -3 & -2.249 & 7 \\ 5 & 1 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 45 \\ 1.751 \\ 9 \end{pmatrix} \text{ de solution } \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

Principes généraux - règle 4

Exemple : propagation des erreurs dans le pivot de Gauss

$$\begin{pmatrix} 20 & 15 & 10 \\ -3 & -2.249 & 7 \\ 5 & 1 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 45 \\ 1.751 \\ 9 \end{pmatrix} \text{ de solution } \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

En éliminant dans l'ordre la première colonne, puis la deuxième :

$$\begin{pmatrix} 20 & 15 & 10 \\ 0 & 0.001 & 8.5 \\ 0 & -2.75 & 0.5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 45 \\ 1.751 \\ 9 \end{pmatrix}$$

$$\begin{pmatrix} 20 & 15 & 10 \\ 0 & 0.001 & 8.5 \\ 0 & 0 & 23375.5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 45 \\ 8.501 \\ 23375.5 \end{pmatrix}$$

Supposons qu'on a une erreur Δ sur x_3, donc $x_3 = 1 + \Delta$.

$$x_2 = \frac{8.501 - 8.5x_3}{0.001} = 1 - 8500\Delta$$

$$x_1 = \frac{45 - 10x_3 - 15x_2}{20} = 1 - 6367.5\Delta$$

Bordeaux INP AQUITAINE

Principes généraux - règle 4

Exemple : propagation des erreurs dans le pivot de Gauss

$$\begin{pmatrix} 20 & 15 & 10 \\ 0 & 0.001 & 8.5 \\ 0 & 0 & 23375.5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 45 \\ 8.501 \\ 23375.5 \end{pmatrix}$$

Supposons qu'on a une erreur Δ sur x_3, donc $x_3 = 1 + \Delta$.

$$x_2 = \frac{8.501 - 8.5x_3}{0.001} = 1 - 8500\Delta$$

$$x_1 = \frac{45 - 10x_3 - 15x_2}{20} = 1 - 6367.5\Delta$$

Il y a deux facteurs de propagation de Δ :

- la multiplication par les termes extra-diagonaux
- la division par les termes diagonaux (ou *pivots*)

En pratique, le seul choix de l'algorithme est le choix du pivot.

Plan

Calcul de la complexité

La difficulté est de prendre en compte un ensemble d'opération de coûts différents :

- opérations de branchement (conditions, boucles . . .)
- opérations de chargement et écriture en mémoire (load / store)
- calculs entiers (ALU : *Arithmetic–Logic Unit*) et flottans (FPU : *Floating-Point Unit*) : additions, multiplication, dépendant des nombres utilisés et de la précision des calculs attendue.

Le coût en temps de ces opérations dépend grandement du matériel. Par exemple :

- certains *FPU* peuvent avoir des instructions spéciales pour calculer des racines carrées, opération assez coûteuse
- il peut aussi exister des instructions spécialisées pour les calculs matriciels : des *fused add-multiply* afin d'ajouter et multiplier des vecteurs en un nombre de cycles réduit.
- la gesion de la mémoire dépend fortement de l'architecture (taille des caches, latence mémoire, largeur de bus pour le calcul GPU. . .)

Calcul de la complexité - exemples

Modestement, on va se contenter de compter les opérations qu'on considérera comme élémentaires. En plus de la complexité algorithmique (souvent polynômiale dans les cas que nous étudieront dans les chapitres suivants), il peut être intéressant d'évaluer la constante, afin de comparer 2 algorithmes de même complexité.

Example (Produit matrice-vecteur)

On multiplie une matrice A de dimensions $m \times n$ avec un vecteur de taille n.

$$\text{Calcul de la } i\text{-ème coordonnée} \begin{cases} n & \text{multiplications} \\ n-1 & \text{additions} \end{cases}$$

Avec m coordonnées à calculer, d'où une complexité totale en $\mathcal{O}(mn)$.

Example (Produit matrice-matrice (naïf))

Plus généralement, si on multiplie A par B de dimensions $n \times p$, on fait p produits matrice-vecteurs, soit une complexité en $\mathcal{O}(mnp)$.

Complexité du produit matrice-matrice

Considérons le produit de deux matrices $n \times n$. Il existe en réalité plusieurs algorithmes de complexités différentes.

Algorithme "naïf"

On n'a vu qu'on a une complexité en $\mathcal{O}(n^3)$.

Algorithme de Strassen

On peut effectuer le produit de matrices par blocs, en découpant la matrice en blocs 2×2. Il est alors possible de trouver une façon de calculer les produits par blocs de manière à économiser des produits sous-matriciels.
Complexité : $\mathcal{O}(n^{log_2(7)}) \approx \mathcal{O}(n^{2.807})$.
En pratique, il semble que cette amélioration devient intéressantes pour $n > 150000$, pour un facteur d'amélioration de l'ordre de 10, jouant sur un total de 3×10^{14} opérations.

Algorithme de Coppersmith-Winograd

La complexité la plus basse est obtenue avec l'algorithme de Coppersmith-Winograd (1987), en $\mathcal{O}(\gamma\, n^{2.3737})$, mais il n'est pas utilisé en pratique car il est difficile à implémenter et la constante multiplicative γ est particulièrement importante.