

Algorithmes numériques

Marc DANIEL

Ecole Supérieure d'Ingénieurs de Luminy,
Campus de Luminy, case 925, 13288 Marseille cedex 9
Marc.Daniel@esil.univmed.fr

Février 2012

ESIL, Algorithmes numériques 1

MCours.com

Plan

Première partie : Les bases de l'algorithmique numérique

- Généralités
- Les nombres sur l'ordinateur
- Les calculs sur ordinateur
- Les erreurs, les chiffres significatifs, les tests
- les arithmétiques alternatives
- Considérations algorithmiques

Deuxième partie : Equations non linéaires

- Racine d'une équation
- Evaluation d'un polynôme
- Dichotomie
- Méthode de Newton
- Localisation des racines
- Méthode de Bairstow

ESIL, Algorithmes numériques 2

Plan (suite)

Troisième partie : Systèmes d'équations linéaires

- **Généralités et définitions**
- **Méthodes directes**
 - Méthode de Gauss
 - Méthode du LU
 - Méthode de Cholesky
- **Méthodes itératives**
- **Problèmes aux moindres carrés**
- **Résolution aux moindres carrés**

Quatrième partie : Différentiation et Intégration numériques

- **Approximation par un polynôme**
- **Différentiation numérique**
- **Intégration numérique**
- **Intégrales multiples**

ESIL, Algorithmes numériques 3

Plan (suite)

Cinquième partie : Interpolation - Approximation - Lissage : notions

- **Problème**
- **Interpolation**
- **Meilleure approximation**
- **Lissage**

Conclusion

Bibliographie succincte

ESIL, Algorithmes numériques 4

Objectifs

- **Faire découvrir**
 - la réalité des calculs sur ordinateur
 - le monde des méthodes numériques
 - les méthodes de base
- **Poser la problématique en tant que numéricien**
- **Etre en mesure de faire face à un problème**
 - sans a priori
 - dans le cadre d'une problématique d'ingénieur
- **Permettre d'accéder aux méthodes non vues en cours**

ESIL, Algorithmes numériques 5

Organisation du cours

- **2x6 h de cours**
 - TRES court
- **20h de TD sur machine (en 4 groupes) : travail en binôme**
 - Illustration du cours
 - Programmer les méthodes de base
 - Essayer de faire sentir les principales difficultés
 - Les TD sont organisés en feuilles
 - Un compte rendu sera à rendre pour chaque feuille au plus tard une semaine après la fin du temps imparti
 - Les programmes
 - Les résultats
 - Une analyse
 - Les TD seront notés
- **Un examen général sur le cours, voire sur les TD**

ESIL, Algorithmes numériques 6

Première partie : Les bases de l'algorithmique numérique

ESIL, Algorithmes numériques 7

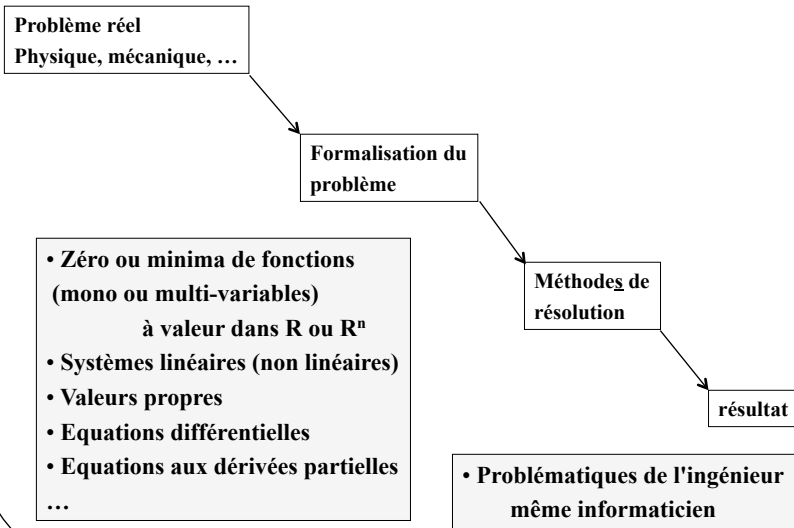
Généralités

- **Algorithmique numérique ou Analyse Numérique ?**
- **2 approches complémentaires possibles (nécessaires)**
 - Unicité de la solution, convergence, vitesse de convergence, ...
 - En lien direct avec les cours d'algèbre et d'analyse
 - Développement d'algorithmes pour résoudre les problèmes
 - Dans les meilleures conditions
 - Avec le problème tel qu'il se pose
 - Et pas tel que l'analyse numérique « aimerait » qu'il se pose
- **L'approche algorithmique n'exclut pas**
 - De savoir calculer avec un ordinateur
 - De savoir analyser et maîtriser les calculs
 - De connaître les principaux résultats théoriques

ESIL, Algorithmes numériques 8

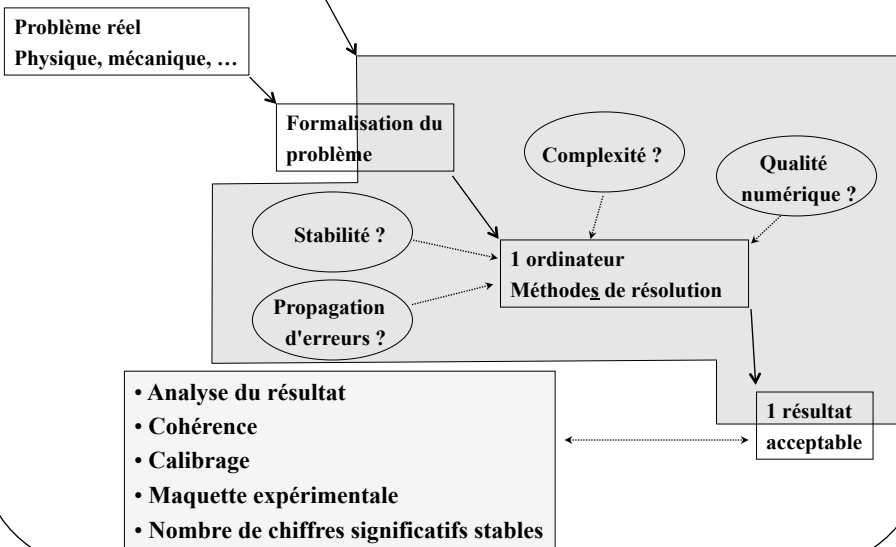
Généralités

Schéma général

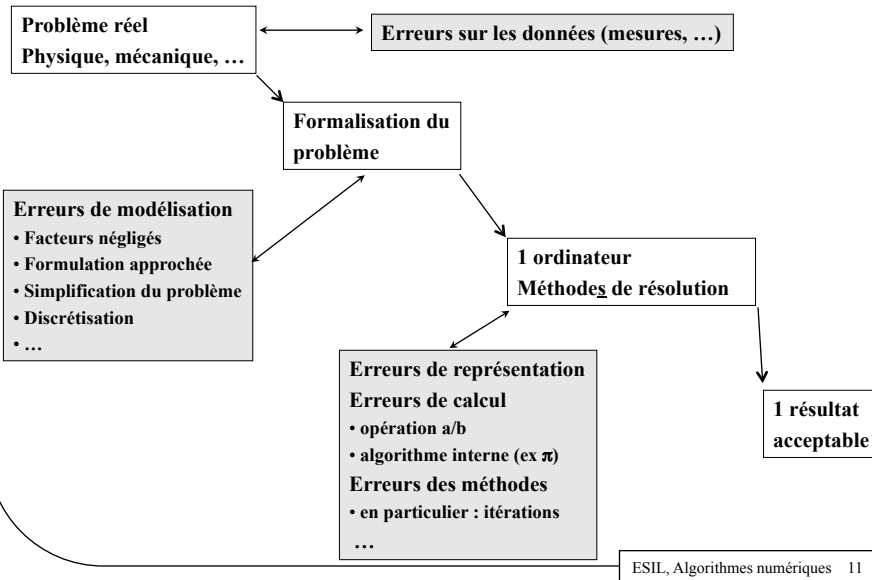


Généralités

Algorithmique numérique



Algorithmique numérique : les erreurs



Algorithmique numérique : les erreurs

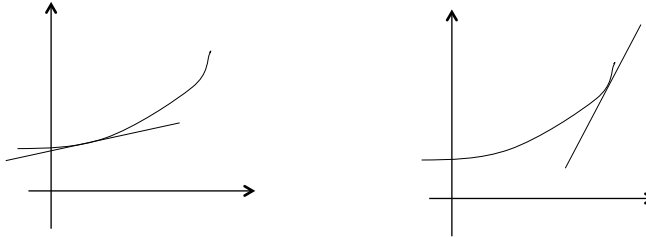
- **Les erreurs sont très nombreuses**
- **Elles peuvent se compenser**
- **Elles ont plutôt tendance à se cumuler**
- **Les erreurs introduites par l'ordinateur doivent être**
 - Maîtrisées,
 - Analysées,
 - Compatibles avec les erreurs sur les données et de modélisation,
 - Compatibles entre elles,
 - Acceptables au niveau du résultat.

Généralités

- **Le conditionnement**

Un problème est dit bien (mal) conditionné si une petite variation sur les données entraîne une petite (grande) variation sur les résultats

- **Peut se rapprocher d'une idée de pente**



- **Problème théorique mais nombreuses conséquences pratiques**

- Si le problème est mal conditionné, il faut écrire le problème autrement ou être en mesure de le gérer

ESIL, Algorithmes numériques 13

Généralités

- **On peut aussi avoir un algorithme mal conditionné (instable)**

- **Cas particulier des matrices mal conditionnées**

- **Données arrondies, coefficients arrondis, calculs approximatifs**

- L'erreur d'arrondi sur les données peut entraîner à elle seule d'énormes erreurs sur le résultat
- Plus tout le reste
- On peut avoir 0 chiffres corrects avec un problème très mal conditionné
 - Voir TP

- **En prime : propagation des erreurs, méthodes de résolution instables, ...**

ESIL, Algorithmes numériques 14

Les nombres

Les entiers

- **Représentés de façon exacte sur n bits (n = 16, 32, 64)**
 - Chaque bit représente une puissance positive de 2
- **Sous forme « complément à 2 »** (utile pour les négatifs)

$$\text{Nombre} = -a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + a_{n-3}2^{n-3} + \dots + a_12^1 + a_{n-1}2^0 \quad a_i \in \{0,1\}, i = 0, \dots, n-1$$

- *Nombre* > 0 : $a_{n-1} = 0$ ($2^{n-1} > 1+2+2^2+\dots+2^{n-2}$)
- Plus grand entier positif : $2^{n-1}-1$ ($2^{15}-1=32767, 2^{31}-1, \dots$)
- Plus petit entier négatif : -2^{n-1} ($-2^{15}=-32768, -2^{31}, \dots$)
- *Nombre* = neg < 0 : on code le nombre sous la forme $-2^{n-1} + \underbrace{(2^{n-1} + \text{neg})}_{= \text{pos} > 0}$

ESIL, Algorithmes numériques 15

Les nombres

Les entiers

- **La forme complément à 2**
 - $x + (-x) = 0$ + 1 bit de retenue perdu
- **Toutes les opérations entières sont exactes**
 - sous réserve de représentation possible du résultat
 - Pas de contrôle
- **La manipulation d'entiers est exacte, mais non contrôlée**
 - Attention aux débordements
 - Source réelle d'erreurs dans les programmes
 - short int : plus compact, mais plus risqué

ESIL, Algorithmes numériques 16

Les « réels »

- **Utilisation de la virgule flottante**

- $x = \pm m \cdot b^l$
- m = mantisse
- b^l = exposant, b est la base
- La base usuelle : 10
- Les bases des ordinateurs : 8, 16, et surtout 2

- $m = 0, d_1 d_2 d_3 \dots d_n \Leftrightarrow m = d_1 \cdot b^{-1} + d_2 \cdot b^{-2} + d_3 \cdot b^{-3} + \dots + d_n \cdot b^{-n}$

- n limité \Rightarrow erreurs de représentation
- $0 \leq d_i < b$ (en base 2, $d_i = 0$ ou 1)

- **Représentation normalisée**

- $d_1 \neq 0$
- Unicité de la représentation

Les « réels » et leurs limites

- **Quels sont les nombres représentés exactement ?**

- **Les flottants dépendent du mode de représentation**

- Plus grand nombre
- Plus petit nombre
- Plus petit nombre en $||$
- Erreur de représentation
- Choix de la base
- Nombre de bits de l'exposant
- Nombre de bits de la mantisse
- Nombre de mots pour représenter le flottant
- **L'erreur de représentation des réels est une erreur relative**

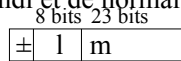
Les « réels » et leurs limites

- **La plupart des réels sont approchés dès leur saisie**

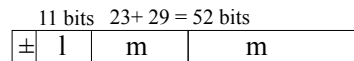
- Influe sur les calculs (erreur de représentation)
- **La norme IEEE-754** : portabilité des programmes
 - Les mêmes résultats !
 - Existe une version 2008

- **La norme IEEE-754**

- Représentation normalisée en base 2 ($d_1=1$, donc non conservé)
- Des règles d'arrondi et de normalisation



- Flottant SP



- Flottant DP
- 1 en décalage de $2^{8-1}-1=127$ ($2^{11-1}-1=1023$)
 - Que des positifs à représenter

Les « réels » et leurs limites

- **La plupart des réels sont approchés dès leur saisie**

- erreur au niveau du dernier bit
- $err = 2^{-24}$ en SP soit $10^{-7,22}$
- $err = 2^{-53}$ en DP soit $10^{-15,95}$

- **err est une quantité quasi nulle par rapport à 1**

- **On peut avoir $x+\delta = x$ (voir TP) ($\delta < err$)**

- **Un flottant résultat d'un calcul ne vaut jamais 0**

- **Un flottant qui doit être nul doit être testé voisin de 0. Deux nombres à comparer ne sont jamais égaux mais voisins**

- $|x| \leq \epsilon$
- $|x-x'| \leq \epsilon$
- ϵ est au mieux de l'ordre de err (très optimiste) ($10^{-7}, 10^{-15}$)

Les nombres

Les « réels » et leurs limites

- **Les exposants diffèrent entre la SP et la DP**
- **Les précisions sur le nombre diffèrent**
 - Le dernier bit de la représentation est faux
 - Affichages réalistes SVP
 - L'ordre de grandeur de l'erreur de représentation est donc de l'ordre de la puissance associée au dernier bit
 - 2^{-24} en SP soit $10^{-7,22}$
 - 2^{-53} en DP soit $10^{-15,95}$
- **Ne pas utiliser la DP par principe**
 - Plus de mémoire
 - Plus de temps de calcul
- **Tout va être à reconsidérer avec les processeurs 64 bits !!!**
 - Sous réserve d'une prise en compte dans la normalisation

ESIL, Algorithmes numériques 21

Les nombres

Les « réels » et leurs limites

- **Un SP ne se convertit pas en DP**

	8 bits	23 bits	
±	1	m	xxxx
	11 bits	23+29 = 52 bits	
±	1	m	0 ou xxxx ?

- Sauf si le SP est exact ET si la conversion complète avec des 0 !

```
int main(void)
{
float x= 0.1 ; /* 0.1 est double en C, mais est convertie dans x */
double xx ;
xx = x ;      /* xx n'est pas la valeur DP correcte de 0.1 */
xx = 0.1e0 ; /* xx est une valeur DP de 0.1e0 (double en C) */
return 1 ;
}
```

- Importante source d'erreur dans les programmes
 - Vérifier les types des variables, des constantes
 - Faire des tests adaptés aux types des variables
- Un seul élément en SP (ou adapté à la SP) \Rightarrow problème garanti
 - Sauf si la DP ne servait à rien !

ESIL, Algorithmes numériques 22

Les nombres

- **Cas des valeurs petites**

- $a \cdot 10^{-p}$ avec p grand : pas de problème
- obtenues par différences : problème important

- **On ne divise jamais par un nombre petit**

- Peut correspondre à un 0
- Est a priori mal défini et le résultat est grand !

ESIL, Algorithmes numériques 23

Les opérations

En entier

- **Tous les calculs sont exacts tant que que le résultat peut s'exprimer en entier**

- `float x ; int a,b ; x = a*b ; /* ne change rien */`

- **En règle général, la division s'effectue dans le type « le plus général », conversion automatique (cast)**

- `float x ; int a ; a = ... ; x = 1/a ; /* */`
- **Une cause d'erreur « on ne peut plus classique »**
- `float x ; int a ; x = 1.0/a ; /* */`
 - En C : 1.0 est double,
 - 1.0/a est double,
 - le résultat est converti en float !

ESIL, Algorithmes numériques 24

Les opérations

En virgule flottante

- **Les opérations sont réalisées dans des registres plus longs**
 - N'augmente pas la précision des opérandes
- **Additions**
 - Mise en correspondance des exposants
 - Addition des mantisses
 - Normalisation du résultat
 - Pas de problème sur des variables du même ordre
 - $a+b=a$ même si $b \neq 0$
 - Résultat arrondi
- **Multiplication**
 - Le résultat dans le registre est bon : comme à la main !
 - Bonne taille des registres
 - Résultat est arrondi

ESIL, Algorithmes numériques 25

Les opérations

En virgule flottante

- **Les divisions ne sont pas exactes dans le registre**
 - Il faut mieux décaler les divisions quand c'est possible
 - Résultat arrondi
- **Les autres opérations**
 - Algorithmes approchés câblés :
 - Développements limités, méthodes de Newton, ...
 - Algorithmes différents suivant les processeurs
 - Résultat arrondi
- **« Tous » les calculs en virgule flottante sont approchés**
 - Sur des nombres approchés
- **L'ordre des opérations n'est pas anodin**
 - Quand il est gérable
- **Arithmétique commutative, mais ni associative ni distributive**

ESIL, Algorithmes numériques 26

Les opérations

La norme définit 4 arrondis

- Vers $-\infty$: $\nabla(x)$ plus petit nombre machine $\leq x$
- Vers $+\infty$: $\Delta(x)$ plus petit nombre machine $\geq x$
- Vers 0 : $:= \nabla(x)$ si $x \geq 0 = \Delta(x)$ si $x < 0$
- Au plus près : on prend le nombre machine le plus proche de x
 - Si x au milieu de deux flottants successifs, on prend la mantisse paire
 - **C'est l'arrondi par défaut**
- **Le choix de l'arrondi n'est pas sans conséquence dans les cas critiques**

ESIL, Algorithmes numériques 27

Les opérations

Gestion des arrondis de l'arithmétique virgule flottante en « C »

```
#include <stdio.h>
#include <fenv.h> // entetes de la virgule flottante
// pour l'édition de liens :action sur la figure flottante
#pragma STDC FENV_ACCESS ON

int main(void){
  /* changement d'arrondi
  to nearest FE_TONEAREST
  toward zero FE_TOWARDZERO
  to +infinity FE_UPWARD
  to -infinity FE_DOWNWARD
  sauvegarde de l'arrondi par défaut */
  const int originalRounding = fegetround(); // FE_TONEAREST
  fesetround(FE_TOWARDZERO); ...
  printf("avec l'arrondi vers 0\n") ; ...
  fesetround(FE_UPWARD); ...
  printf("avec l'arrondi vers + infini\n") ; ...
  fesetround(FE_DOWNWARD); ...
  printf("avec l'arrondi vers - infini\n") ; ...
  fesetround(originalRounding); // on remet l'arrondi d'origine
  return 0; }
```

ESIL, Algorithmes numériques 28

Les opérations

- Soit $f(a,b)=333,75b^5+a^2(11a^2b^2-b^6-121b^4-2)+5,5b^8+a/2b$
 $a=77617$ $b=33096$ extrait de [7]
- **en format IBM (en base 16)**
 - $f(a,b) = 1,172603$ en SP
 - $f(a,b) = 1,1726039400531$ en DP
- **Sur une SUN Sparc**
 - $f(a,b) = -9,875012 \cdot 10^{29}$ en SP
 - $f(a,b) = -1.180592... \cdot 10^{21}$ en DP
- **Sur la machine de votre serveur (IEEE 754)**
 - $-7.498167 \cdot 10^{29}$ en SP
 - $-1.180592... \cdot 10^{21}$ en DP
- **Résultat théorique : -0,827396.....**

Il faut toujours avoir un regard critique

ESIL, Algorithmes numériques 29

Les erreurs

- **Tous les nombres sont approchés - Tous les calculs sont approchés**
 - 2 variables ne peuvent être comparées qu'à une erreur près
- **Comparer x et x^***
 - « $x = x^*$ » ?
 - Erreurs de représentation + erreurs de calcul
 - $x \neq x^*$ **n'a pas de sens**
- **Choix d'une précision ε**
 - $|x-x^*| < \varepsilon$: erreur absolue
 - $\frac{|x-x^*|}{|x|} < \varepsilon$: erreur relative

ESIL, Algorithmes numériques 30

Les erreurs

- $|x - x^*| = |\Delta x| < \varepsilon$: **erreur absolue**
 - L'erreur ne dépend pas de l'ordre de grandeur de x
 - Naturel a priori sur les nombres petits $|x| < 1$
 - Si $|x|$ est grand, peut conduire à aller chercher énormément de chiffres significatifs
 - Ces chiffres peuvent ne pas être accessibles
- $\frac{|x - x^*|}{|x|} = \frac{|\Delta x|}{|x|} < \varepsilon$: **erreur relative**
 - L'erreur dépend de l'ordre de grandeur de x
 - Naturel a priori pour les nombres grands $|x| \gg 1$
- **Une première règle :**
 - Erreur absolue sur les nombres $|x| < 1$, on parle de précision absolue
 - Erreur relative pour les nombres $|x| > 1$, on parle de précision relative
 - Équivalent si $|x| \approx 1$
 - Il existe des applications nécessitant une précision absolue

ESIL, Algorithmes numériques 31

Les erreurs

- **L'important : le nombre de chiffres significatifs (stables)**
 - Quel que soit l'ordre de grandeur de la variable
- **Comment relier la précision au nombre de chiffres significatifs ?**
- **Un retour arrière : précision absolue de l'ordre de 10^{-n}**
 - $\Delta x \approx 10^{-n}$
 - $x = 0.\text{ddddddddd} 10^\alpha$
 - 3 cas

– $\alpha = 0$	$x = 0.\text{ddddddddd}$	nbchif =
– $\alpha < 0$	$x = 0.\text{0000ddd}$	nbchif =
– $\alpha > 0$	$x = \text{ddd}.\text{ddddddd}$	nbchif =
- **Le nombre de chiffres dépend de α ...**

ESIL, Algorithmes numériques 32

Les erreurs

- **Un retour arrière : précision relative de l'ordre de 10^{-n}**

- $|\Delta x|/|x| \approx 10^{-n}$
- $x = 0.\text{ddddddddd} 10^\alpha$, l'ordre de grandeur de x est $10^{\alpha-1}$

Cela revient à étudier avec la précision absolue $10^{-n} \cdot 10^{\alpha-1} = 10^{-(n-\alpha+1)}$

3 cas

- $\alpha=0$ nbchif =
- $\alpha<0$ nbchif =
- $\alpha>0$ nbchif =

ESIL, Algorithmes numériques 33

Les erreurs

- **En conclusion**

- $|x| > 1$ précision relative de 10^{-n}
- $|x| < 1$ précision absolue de 10^{-m}
- La cohérence impose $m=n+1$

- **Test d'arrêt très général**

- $|\Delta x| < \epsilon_{\text{relative}} |x| + \epsilon_{\text{absolue}}$
- Valable si x grand ou non

- **Et ramené en nombre de chiffres significatifs stables**

- n chiffres significatifs stables

- $|\Delta x| < 10^{-n} (|x| + 0.1)$

- Un vrai bon test (largement utilisé)

ESIL, Algorithmes numériques 34

Arithmétique d'intervalle

- **Norme IEEE1788 en 2012 ?**
- **Chaque variable appartient à un intervalle et est notée [v]**
 - $[v]=[v_*,v^*] = \{x/ v_* \leq x \leq v^*\}$
- **Les opérations sont définies sur des intervalles**
 - $[a]+[b] = [a_*+b_*,a^*+b^*]$
 - $[a]-[b] = [a_*-b^*,a^*-b_*]$
 - $[a]*[b] = [\min(a_*b_*, a_*b^*, a^*b_*, a^*b^*), \max(a_*b_*, a_*b^*, a^*b_*, a^*b^*)]$
 - $[a]/[b] = [a_*,a^*] * [1/b^*,1/b_*]$ si $0 \notin [b]$
- **Généralisation aux fonctions continues : $\exp([v])=[\exp(v_*),\exp(v^*)]$**
 - Dépend de la monotonie de la fonction
- **La vraie valeur toujours à l'intérieur d'un intervalle : fiable**
 - Utilisation des arrondis vers $-\infty$ (v_*) et $+\infty$ (v^*)

Arithmétique d'intervalle

- **Les limites**
 - Le temps de calcul
 - La taille des intervalles des variables qui croît très vite
 - Problème quand 0 est dans l'intervalle
 - Surestimation des intervalles
 - Ex : $P(x) = x-x^2$
 - $P([0,1]) = [0,1] - [0,1]^2 = [0,1] - [0,1] = [-1,1]$
 - Ou avec $P(x) = x(1-x)$
 - $P([0,1]) = [0,1] * [1-[0,1]] = [0,1] * [0,1] = [0,1]$
 - Or le bon intervalle est $[0,1/4]$
 - $[a]^2 = [\min(a_*^2, a^{*2}), \max(a_*^2, a^{*2})]$ si $0 \notin [a]$
 $= [0, \max(a_*^2, a^{*2})]$ autrement
 plus précis que $[a]*[a]$ car les bornes sont corrélées
 - Il faut éventuellement envisager de réduire les intervalles

Arithmétique stochastique

- **Basée sur les travaux de Laporte et Vigne (années 70)**
- **On perturbe aléatoirement les données et les calculs**
 - En particulier en jouant sur les règles d'arrondi
- **On obtient le nombre de chiffres significatifs stables obtenus en comparant les résultats**
 - Nombre de chiffres obtenu avec très peu de tirages en pratique (≈ 3)
 - Permet de statuer sur :
 - une variable sans chiffres significatifs
 - 2 variables stochastiquement équivalentes (tous les chiffres stables sont identiques)
- **Nombre de chiffres significatifs du résultat du code donné**
 - CADNA : <http://www.lip6.fr/cadna>
 - Existe pour MPI et GPU
 - Beaucoup plus long, plus de mémoire, mais fiable

Arithmétique rationnelle

- **La valeur d'une variable est de la forme a/b**
 - a et b entiers
 - Premiers entre eux (forme irréductible)
- **Les calculs sont effectués sur des fractions rationnelles**

$$\frac{a}{b} + \frac{c}{d} = \frac{ad + bc}{bd} = \frac{num}{den}$$
, num et den premiers entre eux
- **Les fonctions mathématiques sont codées en rationnel**
- **Les limites**
 - Ne représente pas tous les rationnels
 - L'ensemble Q n'est pas l'ensemble R
 - Calculs très lents

Précision infinie

- **Les nombres sont décomposés en blocs de chiffres successifs.**

– X1 =

xxxx	xxxx	yyyyyyyyyy	zzzzzzzz
------	------	------------	----------

– X2 =

uuuu	uuuu	vvvvvvvv	ssssssss
------	------	----------	----------

- **Les opérations sont faites sur les blocs « comme à la main »**
- **X1 opé X2 à la précision désirée**
- **Les limites**
 - La mémoire disponible
 - Le temps de calcul
 - L'intérêt réel sur des données imprécises
- **Voir par exemple :** <http://www.swox.com/gmp/>
 - The GNU Multiple Precision Arithmetic Library

Le calcul symbolique

- **Transformation des expressions mathématiques**
 - Evaluation finale
- **Des outils intéressants de calcul symbolique**
 - Mathematica, Maple, ...
- **Les limites**
 - Le temps de calcul
 - Les capacités de résolution
 - L'intérêt réel sur des données imprécises

Tendances

- **Utilisation systématique de la double précision**
 - Masque ou repousse de nombreuses difficultés
- **Les processeurs 64 bits masquent ou repoussent de nombreuses difficultés**
 - Peut convenir
- **Un fort intérêt pour les arithmétiques alternatives**
 - Développement de bibliothèques
- **Freins importants liés à la lourdeur des approches**
- **L'arithmétique flottante a encore de beaux jours devant elle**
 - Toujours être conscient des risques et contrôler les résultats

Tendances : Abstract Program Equivalent Graphs

Précompilation

- **Grappe des différentes combinaisons en calculant les expressions dans tous les sens possibles (exhaustif)**
- **Calcul de l'erreur faite dans chaque chemin (via des calculs d'intervalles)**
- **Choix de la solution qui minimise les erreurs : meilleur programme associé à un ensemble de calculs**

Considérations algorithmiques

• Les problèmes itératifs

- De nombreux processus sont itératifs
- D'un point de vue théorique : $\lim_{n \rightarrow \infty} x_n = l$
 - l est inconnue (on ne peut pas comparer x_n à l !)
 - La démonstration de la convergence passe par des critères mathématiques puissants : invérifiable sur un ordinateur
 - Il peut exister des conditions de convergence : invérifiable sur un ordinateur
- Les critères utilisés sont plus faibles que les critères théoriques
- Les processus itératifs peuvent être divergents
 - Pour des raisons théoriques
 - Pour des raisons liées à l'ordinateur
 - Peuvent sembler converger sur l'ordinateur (ex : $u_n = n$)

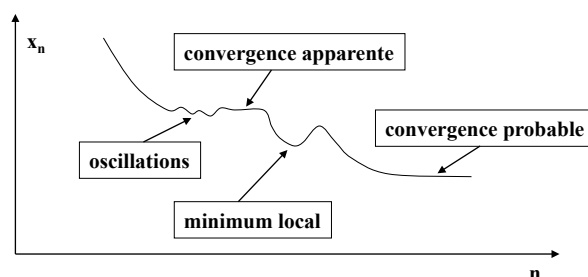
ESIL, Algorithmes numériques 43

Considérations algorithmiques

• L'idéal serait de comparer x_n avec la limite l , mais celle-ci est inconnue !

- On ne peut comparer que des itérés successifs $|x_{n+1} - x_n|$
- On peut s'assurer que plusieurs itérés successifs semblent converger

• La réalité de la convergence d'un processus itératif



ESIL, Algorithmes numériques 44

Considérations algorithmiques

- **Il faut des tests robustes**

- Comparaison des itérés successifs $|x_{n+1}-x_n|$ (au minimum)
- Limiter le nombre d'itérations (indispensable)
- Tester tous les problèmes intermédiaires
 - Ex : dénominateur petit

- **En sortie de la structure itérative**

- Il faut savoir sur quelle condition la structure itérative a été quittée

- **Rapidement : de nombreux tests associés à de nombreux ϵ**

- Programmation par les ϵ
- Souvent incompréhensibles pour l'utilisateur
- Un bon programme doit pouvoir laisser l'utilisateur choisir les ϵ
- Si possible, les faire dépendre d'un minimum de précisions maîtrisables
- Problème complexe !

ESIL, Algorithmes numériques 45

Considérations algorithmiques

Bon algorithme ? (1/3)

- **Des critères que doivent respecter un bon algorithme**

- fiable (gère tous les cas)
- robuste
- stable
- fournit des résultats de qualité
- ...
- performant

- **mais aussi**

- facile à implémenter
- facile à maintenir
- taille mémoire nécessaire ...

ESIL, Algorithmes numériques 46

Bon algorithme ? (2/3)

- **Algorithme adapté aux données à traiter en général ?**
 - même si non optimal
 - ex : algorithme de tri (bulle) pour des données presque triées
 - Pas de l'algorithmique numérique !
- **Cadre dans lequel il faut s'intéresser à la complexité :**
 - plus rapide, à **qualité et adéquation au problème égales**
 - pas toujours si simple à faire
 - cas des hypothèses particulières (sur un exemple, et sont-elles vérifiées ?)

Bon algorithme ? (3/3)

- **Ne pas oublier une présentation soignée**
 - Indentation rigoureuse
 - Beaucoup de commentaires (utiles)
 - Entêtes
 - Numéro de version,
 - Date de création, date de modification
 - Auteur, entreprise
 - Descriptif
 - Dictionnaire des variables en entrée et sortie
- **Tout ce qui est vrai pour l'algorithmique est vrai pour l'algorithmique numérique**
 - Les problèmes numériques en plus !

La complexité

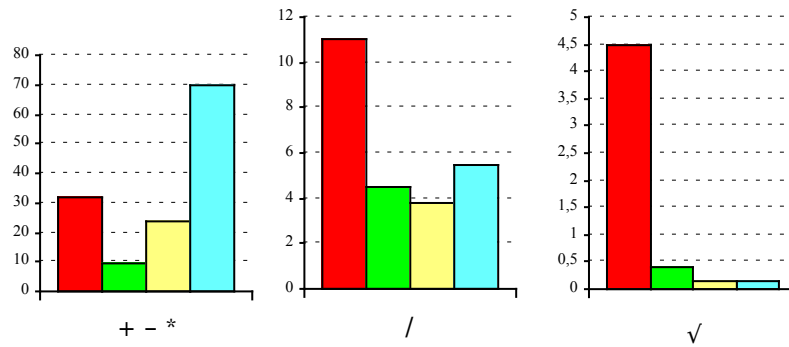
- **La complexité mémoire**
 - Structure mémoire adaptable aux problèmes à traiter
 - Pas de float T[10] en « dur »; !
 - Attention à la taille des structures de données
 - La taille augmente très vite
 - Attention à l'allocation dynamique de mémoire
 - Surtout si peu de désallocation ...,
 - Ou mal gérée !
 - Cela a un coût si dans une structure itérative très interne

- **La complexité temporelle**
 - Il faut considérer le Mflops et non le Mips
- **On n'additionne pas n'importe quoi ensemble**
 - pour passer de l'évaluation en nombre d'opérations à l'évaluation en temps (nombre de cycles) :
 - contexte logiciel
 - contexte matériel
 - obtenir les ratios de temps entre les opérations
 - appliquer ces ratios
 - obtenir un temps global

Considérations algorithmiques

Nombre de Megaflops pour effectuer des opérations élémentaires

■ Silicon Graphics R10000 ■ Sun Sparc 20 □ Pentium 166 ■ Pentium Pro 200



Attention aux techniques de mesure

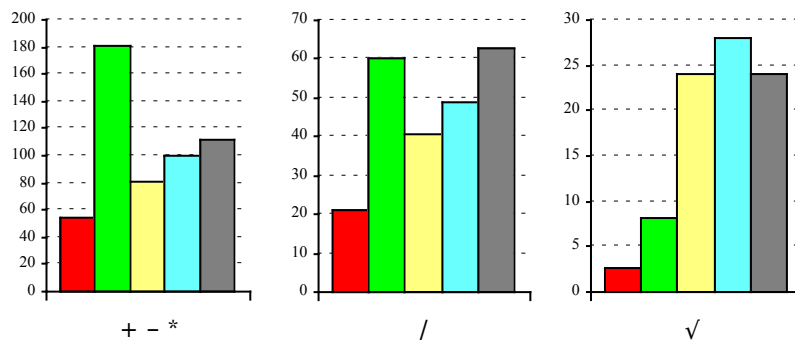
* pas d'optimisation possible

* mesurer et déduire les temps des opérations « accessoires »

Considérations algorithmiques

Nombre de Megaflops pour effectuer des opérations élémentaires (tests 2005)

■ G4 550 MHz ■ G4 1,5 GHz □ P4 2 GHz ■ P4 2,4 GHz ■ AMD Athlon 1 GHz



Données fiables en faisant les tests sur 10⁹ opérations

Pas de résultats stables sur le G5

Considérations algorithmiques

- **On ne peut pas aborder l'algorithmique numérique sans avoir une idée des temps de calcul**
 - Il y a 30 ans, une école d'ingénieurs avec une machine d'1Mflops était une école bien dotée
 - On peut considérer un PC actuel à 200 Mflops
 - Cray X1E : 147 Tflops (2005)(1 Tflops = 10^6 Mflops)
 - Blue Gene : 280 Tflops (fin 2005) (131072 processeurs!) (478 Tflops en 2007)
 - Jaguar (Cray) 1759 Tflops (en 2009) (224162 processeurs !)
 - Fujitsu annonce 10 pétaflops fin 2012 ($10 \cdot 10^{15}$)
- **Mais**
 - problèmes mécaniques exceptionnels très non linéaires, 2 000 000 ddl, soit 1 ou 2 semaines de calcul sur quelques dizaines de processeurs
 - problèmes de mécanique linéaire (éléments finis) on peut atteindre 10^9 ddl, soit des systèmes linéaires ($10^9, 10^9$), soit de l'ordre de 10^{27} opérations soit 10^{13} secondes de Blue Gene
 - Pour tempérer : matrices creuses
 - **A méditer ...**

ESIL, Algorithmes numériques 53

Deuxième partie : Equations non linéaires

ESIL, Algorithmes numériques 54

Racine d'une équation

- **Soit l'équation non linéaire $f(x)=0$**
 - r est une racine de l'équation ou par abus de f si $f(r)=0$
- **f à valeur dans \mathbf{R}^q , $\mathbf{x} \in \mathbf{R}^p$**
 - On considérera surtout $p=q=1$
- **Liens avec l'optimisation non linéaire : $\min f(x)$, $\mathbf{x} \in \mathbf{D}$**
 - Un extremum de f est caractérisé par :
 - $f(x)=0$ ($p=1$)
 - Minimum : $f'(x) > 0$
 - Maximum : $f'(x) < 0$
 - $\text{grad } f = 0$ ($p > 1$), si f est à valeur dans \mathbf{R} ($q=1$)
 - Matrice Hessienne H (symétrique) définie positive : minimum
 - Matrice Hessienne H (symétrique) définie négative : maximum
- **La recherche des racines d'une équation est un problème de base**

ESIL, Algorithmes numériques 55

Racine d'une équation

- **Quelles situations ? (1/2)**
 - La fonction est connue discrètement
 - A la précision de discrétisation près
 - Aux conditions de régularité supposées près
 - Rem : outil imparfait de localisation des racines
 - La dérivée première est accessible ou non
 - Elle coûte à calculer ou non
 - La fonction est connue sous forme d'une fonction d'évaluation
 - La fonction est un polynôme, et on a accès aux coefficients du polynôme

ESIL, Algorithmes numériques 56

Racine d'une équation

- **Quelles situations ? (2/2)**

- Une racine cherchée
 - La plus petite, la plus grande, la plus proche d'une valeur donnée
- Toutes les racines
 - Combien faut-il en chercher ?
 - Problème difficile même pour un polynôme (R n'est pas algébriquement clos !)
- Le cas des racines multiples ($f(r) = f'(r) = \dots = f^{(p)}(r) = 0$)
- Le cas insoluble de l'infinité de racines
 - Cas très important en CAO
 - Cas limite : racines multiples

ESIL, Algorithmes numériques 57

Evaluation d'un polynôme

- **Dans quelle base est exprimée le polynôme**

- Base canonique (power basis)
- Autre base, Bernstein en particulier $B_{i,n}(t) = C_i^n t^i (1-t)^{n-i} \quad i = 0, \dots, n$
- Attention, changement de base mal conditionné

- **$P(x) = a_0 x^n + a_1 x^{n-1} + \dots + a_{n-1} x + a_n$**

- Evaluation « directe » à proscrire
 - Stabilité
 - Nombre d'opérations

- **Il faut utiliser le schéma de Horner dans la base canonique**

- **D'autres solutions dans d'autres bases**

- Ex l'algorithme de De Casteljau dans la base de Bernstein

ESIL, Algorithmes numériques 58

Evaluation d'un polynôme

- **Le Schéma de Horner** $P(x) = a_0 x^n + a_1 x^{n-1} + \dots + a_{n-1} x + a_n$
 - Ou $P_n(x) = a_0 x^n + a_1 x^{n-1} + \dots + a_{n-1} x + a_n$
- **Formellement :**
 - $P_n(x) = a_0 x^n + a_1 x^{n-1} + \dots + a_{n-1} x + a_n$
 - $P_n(x) = (a_0 x^{n-1} + a_1 x^{n-2} + \dots + a_{n-1}) x + a_n = a_n + x P_1(x)$
 - $P_n(x) = ((a_0 x^{n-1} + a_1 x^{n-2} + \dots + a_{n-2}) x + a_{n-1}) x + a_n$
 - $\quad = a_n + x (a_{n-1} + x P_2(x))$
 - ...
- **Présentation pratique : on cherche à calculer $P_n(\alpha)$**
 - $P_n(x) = (x - \alpha) Q_{n-1}(x) + A_n$

Evaluation d'un polynôme

- $b_0 = a_0$
 - ...
 - $b_i = a_i + \alpha b_{i-1}$
 - ...
 - $A_n = a_n + \alpha b_{n-1}$
- Nombre d'opérations ?
 - On garde Q_{n-1} ou pas
- $$b = a_i + \alpha b$$

- **Dérivées successives :**
 - $P_n(x) = (x - \alpha) Q_{n-1}(x) + A_n$
 - $Q_{n-1}(x) = (x - \alpha) Q_{n-2}(x) + A_{n-1}$
 - ...
 - $Q_1(x) = (x - \alpha) Q_0(x) + A_1$
 - $Q_0(x) = A_0$

$$P_n(x) = A_n + (x - \alpha)A_{n-1} + (x - \alpha)^2 A_{n-2} + \dots + (x - \alpha)^n A_0$$

A comparer avec le développement de Taylor au voisinage de α :

$$P_n(x) = P_n(\alpha) + (x - \alpha)P_n'(\alpha) + (x - \alpha)^2 \frac{P_n''(\alpha)}{2!} + \dots + (x - \alpha)^n \frac{P_n^{(n)}(\alpha)}{n!} + 0$$

Dichotomie

- **(bisection en anglais)**
- **Soit f continue sur $[a,b]$ $f(a).f(b)<0$**
 - Théorème des valeurs intermédiaires : $\exists \alpha \in]a,b[f(\alpha)=0$
 - Hypothèse sup. : f est monotone sur $[a,b]$ $\exists! \alpha \in]a,b[f(\alpha)=0$
 - Il faut commencer par trouver 2 telles valeurs a et b !

Tantque (précision non atteinte) faire

$$c=(a+b)/2$$

Si $(f(a).f(c))<0$ alors

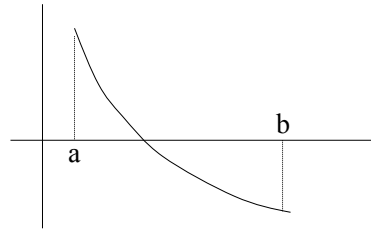
$$b=c$$

sinon

$$a=c$$

Finsi

Fintantque /* la racine $\in [a,b]$ */



- Et si la fonction n'est pas monotone ?

ESIL, Algorithmes numériques 61

Dichotomie

- **Et si $|f(c)|<\epsilon$?**
- **Et si $f(c) = 0$?**
 - Cas peu probable qui statistiquement se rencontre toujours sur les tests
 - Avec l'algorithme proposé, $[a,b] \rightarrow [c^{(1)},b] \rightarrow [c^{(2)},b] \rightarrow [c^{(3)},b]$

Perdu !

~~Tantque (précision non atteinte) faire~~

~~$$c=(a+b)/2$$~~

~~Si $(f(a).f(c))<0$ alors~~

~~$$b=c$$~~

~~sinon~~

~~$$a=c$$~~

~~Finsi~~~~Fintantque /* la racine $\in [a,b]$ */~~

Tantque (précision non atteinte) faire

$$c=(a+b)/2$$

Si $(f(a).f(c))\leq 0$ alors

$$b=c$$

sinon

$$a=c$$

Finsi

Fintantque /* la racine $\in [a,b]$ */

- Convergence normale : trouvera c comme borne sup de l'intervalle résultat

ESIL, Algorithmes numériques 62

Dichotomie

- **$l=b-a$**
 - Itération 1 : $b-a=$
 - Itération n : $b-a=$
- **On a un encadrement de la racine**
 - si α^* est la vraie valeur
 - $\forall \alpha \in]a, b[\quad |\alpha - \alpha^*| \leq$
Pour avoir une erreur de l'ordre de ϵ , il suffit de prendre $n \geq$
- **Attention aux précisions non atteignables et aux problèmes numériques**
 - Toujours limiter le nombre d'itérations
- **Ce n'est pas la méthode la plus rapide**
- **C'est la méthode LA PLUS ROBUSTE**
 - C'est une bonne méthode générique

Tantque (précision non atteinte) faire

$c=(a+b)/2$

Si $(f(a).f(c)) \leq 0$ alors

$b=c$

sinon

$a=c$

Finsi

Fintantque /* la racine $\in [a, b]$ */

ESIL, Algorithmes numériques 63

La méthode de Newton

- **Une des méthodes les plus célèbres**
 - Utilisable pour les fonctions de plusieurs variables
 - Convergence rapide (quand elle converge)
- **Soit un point de départ donné x_0**
 - On cherche Δx t.q. $f(x_0 + \Delta x) = 0$
 - Soit le développement de Taylor de f au voisinage de x :

– En négligeant les termes à partir du 2e ordre :

– D'où $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$

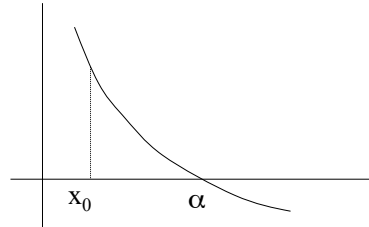
– La méthode est approchée, il faut itérer : $k \geq 1 \quad x_k = x_{k-1} - \frac{f(x_{k-1})}{f'(x_{k-1})}$

ESIL, Algorithmes numériques 64

La méthode de Newton

- **Interprétation géométrique**

- Équation de la tangente en x_0
- La tangente coupe l'axe des x en
- Convergence rapide (quand elle converge)



- **La méthode de Newton impose d'avoir accès à la dérivée**
- **Une dérivée approchée conduira tout de même à un résultat**
 - À l'extrême, n'importe quelle droite va fonctionner
 - Attention à l'«annulation» de la dérivée
 - Que se passe-t-il ?
- **Mieux vaut « a priori » partir près de la solution**

ESIL, Algorithmes numériques 65

La méthode de Newton

- **Analyse de l'erreur**

- Développement de Taylor au voisinage de x_0
- Et en α (1) :
- L'algorithme revient à écrire : $0=f(x_0)+(x_1-x_0)f'(x_0)$ (2)
- Donc (1)+(2) (si f' ne s'annule pas) :

$$\varepsilon_{i+1} \leq K\varepsilon_i^2$$

- **La convergence de la méthode de Newton est quadratique**
 - Beaucoup plus rapide que la dichotomie
 - N'offre pas un encadrement de la racine
 - Imparfaitement, double le nombre de chiffres corrects à chaque itération

ESIL, Algorithmes numériques 66

La méthode de Newton

- **L'application la plus connue : racine pième d'un nombre a**

- $f(x) = x^p - a = 0$

- $f'(x) = p \cdot x^{p-1}$

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

- $x_{k+1} = \frac{1}{p} \left((p-1)x_k + \frac{a}{x_k^{p-1}} \right)$

- **Si p=2**

- $x_{k+1} = \frac{1}{2} \left(x_k + \frac{a}{x_k} \right)$

- 1 des algorithmes cablés sur les processeurs

- + un algorithme pour trouver un point de départ convenable

ESIL, Algorithmes numériques 67

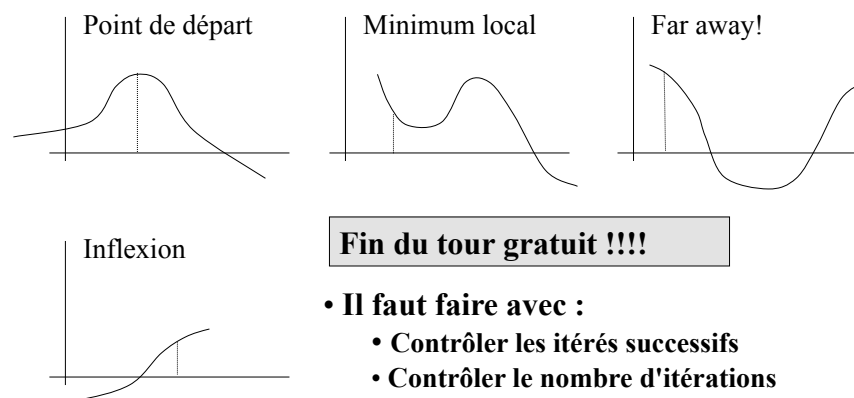
La méthode de Newton

- **Tout n'est pas si merveilleux**

- Cela ne fonctionne pas toujours ...

- Il y a des conditions théoriques pour que Newton converge

- Elles sont invérifiables en pratique



Fin du tour gratuit !!!!

- **Il faut faire avec :**

- Contrôler les itérés successifs
 - Contrôler le nombre d'itérations
 - Contrôler les dérivées
 - ...

ESIL, Algorithmes numériques 68

La méthode de Newton

- **Cas des racines multiples d'ordre p**

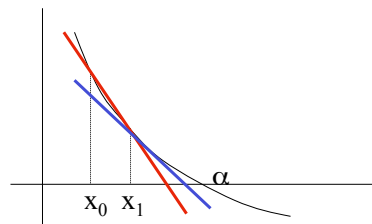
- $f(\alpha) = f'(\alpha) = \dots = f^{(p-1)}(\alpha), f^{(p)}(\alpha) \neq 0$
- La convergence se ralentit fortement et attention à la dérivée ...
- α est racine simple de $u(x)=f(x)/f'(x)$
 - On peut appliquer Newton à u
 - Fait apparaître f''
 - Convergence quadratique

- **Méthode de la sécante**

- Méthode à 2 pas
- Pas d'évaluation de la dérivée
- Pas de convergence quadratique

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

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



ESIL, Algorithmes numériques 69

Localisation des racines

- **Cas général**

- L'idéal : avoir l'information provenant du problème lui-même
- Discrétiser l'intervalle de définition et encadrer des racines
 - Problème pour un encadrement de $2m$ racines
 - Une seule racine si encadrement de $2m+1$ racines

- **Attention avec Newton, l'encadrement de la racine ne conduit pas forcément à l'obtention de la racine**

- **Cas du polynôme : diviser à chaque fois $P(x)$ par $(x-\alpha)$**

- Il faut programmer la division euclidienne de 2 polynômes
 - Comme « à la main », ou Horner (!)
- Attention à l'accumulation des erreurs
- Vérifier que α est toujours racine du polynôme initial ...

ESIL, Algorithmes numériques 70

Localisation des racines

Cas des polynômes

- **Règles de Descartes**

- Localisation
- Nombre de racines positives

- **Suite de Sturm**

- $P(x)$ polynôme (à coefficients réels) n'ayant que des racines simples
- $P_0(x)=P(x)$ $P_1(x)=P'(x)$
- $j \geq 2$ $P_j : P_{j-2}(x)=P_{j-1}(x) \cdot Q_{j-1}(x)-P_j(x)$ ($P_0=P_1 \cdot Q_1-P_2$)
- On arrête la suite quand P_m est constant (au pire, $m=n$)

- $S(x) = (P_0, P_1, \dots, P_m)$ et $V(u)$ le nombre de changements de signe dans $S(u)$
- Le nombre de racines de P sur $[u, v] = V(u) - V(v)$

ESIL, Algorithmes numériques 71

Méthode de Bairstow

- **Polynômes à coefficients réels: la méthode de Bairstow**

- Recherche des racines sur \mathbb{C} (corps algébriquement clos)

- $P_n(x)=(x^2+p \cdot x+q)Q_{n-2}(x)+R(p,q) \cdot x+S(p,q)$
 - On cherche p et q de sorte que $S(p,q)$ et $R(p,q)$ soient nuls
 - Application de Newton multivariables (p et q)

 - L'algorithme s'arrête lorsque le polynôme Q est de degré 1 ou 2

 - On cherche les racines de $x^2+p \cdot x+q$
 - $\Delta > 0$: deux racines réelles
 - $\Delta = 0$: une racine double
 - $\Delta < 0$: deux racines complexes conjuguées

 - Reste les dernières racines du dernier polynôme Q

ESIL, Algorithmes numériques 72

Méthode de Bairstow

$$\begin{aligned}
 & - a_0 x^n + a_1 x^{n-1} + \dots + a_{n-1} x + a_n \\
 & = (x^2 + p x + q) (b_0 x^{n-2} + b_1 x^{n-3} + \dots + b_{n-3} x + b_{n-2}) + R x + S
 \end{aligned}$$

$$\begin{aligned}
 a_0 &= b_0 \\
 a_1 &= b_1 + p b_0 \\
 a_2 &= b_2 + p b_1 + q b_0 \\
 &\dots \\
 a_k &= b_k + p b_{k-1} + q b_{k-2} \quad 2 \leq k \leq n-2 \\
 &\dots \\
 a_{n-1} &= R + p b_{n-2} + q b_{n-3} \\
 a_n &= S + q b_{n-2}
 \end{aligned}$$

En posant, $b_{n-1} = R$ et $b_n = S - p R$

$$\begin{aligned}
 b_0 &= a_0 \\
 b_1 &= a_1 - p b_0 \\
 b_2 &= a_2 - p b_1 - q b_0 \\
 &\dots \\
 b_k &= a_k - p b_{k-1} - q b_{k-2} \\
 &\dots \\
 b_{n-1} &= a_{n-1} - p b_{n-2} - q b_{n-3} \\
 b_n &= a_n - p b_{n-1} - q b_{n-2}
 \end{aligned}$$

- Le calcul des b_i est donc effectué jusqu'à n à partir de $i=2$

Méthode de Bairstow

• Partant de p_0 et q_0 quelconques, on cherche Δp et Δq

$$- R(p + \Delta p, q + \Delta q) = 0 \text{ et } S(p + \Delta p, q + \Delta q) = 0$$

$$\text{- Au premier ordre : } \begin{cases} R(p, q) + \frac{\partial R}{\partial p} \Delta p + \frac{\partial R}{\partial q} \Delta q = 0 \\ S(p, q) + \frac{\partial S}{\partial p} \Delta p + \frac{\partial S}{\partial q} \Delta q = 0 \end{cases}$$

$$\text{- Système de Cramer } 2 \times 2 \quad \begin{cases} \Delta p = \frac{N_p}{Det} = \frac{S \frac{\partial R}{\partial q} - R \frac{\partial S}{\partial q}}{\frac{\partial R}{\partial p} \frac{\partial S}{\partial q} - \frac{\partial R}{\partial q} \frac{\partial S}{\partial p}} \\ \Delta q = \frac{N_q}{Det} = \frac{R \frac{\partial S}{\partial p} - S \frac{\partial R}{\partial p}}{\frac{\partial R}{\partial p} \frac{\partial S}{\partial q} - \frac{\partial R}{\partial q} \frac{\partial S}{\partial p}} \end{cases}$$

Méthode de Bairstow

• **Reste à estimer les dérivées partielles**

$$\begin{aligned} \partial b_0 / \partial p &= 0 \\ \partial b_1 / \partial p &= -b_0 - p \partial b_0 / \partial p \\ \partial b_2 / \partial p &= -b_1 - p \partial b_1 / \partial p - q \partial b_0 / \partial p \\ &\dots \\ \partial b_k / \partial p &= -b_{k-1} - p \partial b_{k-1} / \partial p - q \partial b_{k-2} / \partial p \\ &\dots \\ \partial b_{n-1} / \partial p &= -b_{n-2} - p \partial b_{n-2} / \partial p - q \partial b_{n-3} / \partial p \\ \partial b_n / \partial p &= -b_{n-1} - p \partial b_{n-1} / \partial p - q \partial b_{n-2} / \partial p \end{aligned}$$

• Notations : $\partial b_{k+1} / \partial p = -c_k, 0 \leq k \leq n-2$

• Et pour alléger les notations finales :

$$\partial b_n / \partial p = -c_{n-1} - b_{n-1}$$

$$\begin{aligned} b_0 &= a_0 \\ b_1 &= a_1 - p b_0 \\ b_2 &= a_2 - p b_1 - q b_0 \\ &\dots \\ b_k &= a_k - p b_{k-1} - q b_{k-2} \\ &\dots \\ b_{n-1} &= a_{n-1} - p b_{n-2} - q b_{n-3} \\ b_n &= a_n - p b_{n-1} - q b_{n-2} \end{aligned}$$

$$\begin{aligned} c_0 &= b_0 \\ c_1 &= b_1 - p c_0 \\ c_2 &= b_2 - p c_1 - q c_0 \\ &\dots \\ c_k &= b_k - p c_{k-1} - q c_{k-2} \\ &\dots \\ c_{n-2} &= b_{n-2} - p c_{n-3} - q c_{n-4} \\ c_{n-1} &= -p c_{n-2} - q c_{n-3} \end{aligned}$$

Méthode de Bairstow

• **Reste à estimer les dérivées partielles**

$$\begin{aligned} \partial b_0 / \partial q &= 0 \\ \partial b_1 / \partial q &= 0 \\ \partial b_2 / \partial q &= -b_0 - p \partial b_1 / \partial q - q \partial b_0 / \partial q \\ &\dots \\ \partial b_k / \partial q &= -b_{k-2} - p \partial b_{k-1} / \partial q - q \partial b_{k-2} / \partial q \\ &\dots \\ \partial b_{n-1} / \partial q &= -b_{n-3} - p \partial b_{n-2} / \partial q - q \partial b_{n-3} / \partial q \\ \partial b_n / \partial q &= -b_{n-2} - p \partial b_{n-1} / \partial q - q \partial b_{n-2} / \partial q \end{aligned}$$

• Notations : $\partial b_{k+2} / \partial q = -c'_k, 0 \leq k \leq n-2$

$$\begin{aligned} c_0 &= b_0 \\ c_1 &= b_1 - p c_0 \\ c_2 &= b_2 - p c_1 - q c_0 \\ &\dots \\ c_k &= b_k - p c_{k-1} - q c_{k-2} \\ &\dots \\ c_{n-2} &= b_{n-2} - p c_{n-3} - q c_{n-4} \\ c_{n-1} &= -p c_{n-2} - q c_{n-3} \end{aligned}$$

$$c_k = c'_k \quad k \leq n-2$$

$$\begin{aligned} b_0 &= a_0 \\ b_1 &= a_1 - p b_0 \\ b_2 &= a_2 - p b_1 - q b_0 \\ &\dots \\ b_k &= a_k - p b_{k-1} - q b_{k-2} \\ &\dots \\ b_{n-1} &= a_{n-1} - p b_{n-2} - q b_{n-3} \\ b_n &= a_n - p b_{n-1} - q b_{n-2} \end{aligned}$$

$$\begin{aligned} c'_0 &= b_0 \\ c'_1 &= b_1 - p c'_0 \\ c'_2 &= b_2 - p c'_1 - q c'_0 \\ &\dots \\ c'_k &= b_k - p c'_{k-1} - q c'_{k-2} \\ &\dots \\ c'_{n-2} &= b_{n-2} - p c'_{n-3} - q c'_{n-4} \end{aligned}$$

Méthode de Bairstow

• Il reste à tout regrouper

- $b_{n-1} = R$ et $b_n = S - pR$ (1)
- $\partial b_{k+1}/\partial p = -c_k$ et $\partial b_n/\partial p = -c_{n-1} - b_{n-1}$
- $\partial b_{k+2}/\partial q = -c'_k$
- (1) \Rightarrow

$$\left\{ \begin{array}{l} \Delta p = \frac{N_p}{Det} = \frac{S \frac{\partial R}{\partial q} - R \frac{\partial S}{\partial q}}{\frac{\partial R}{\partial p} \frac{\partial q}{\partial q} - \frac{\partial R}{\partial q} \frac{\partial S}{\partial p}} \\ \Delta q = \frac{N_q}{Det} = \frac{R \frac{\partial S}{\partial p} - S \frac{\partial R}{\partial p}}{\frac{\partial R}{\partial p} \frac{\partial q}{\partial q} - \frac{\partial R}{\partial q} \frac{\partial S}{\partial p}} \end{array} \right.$$

$$\begin{aligned} \frac{\partial R}{\partial p} &= \frac{\partial b_{n-1}}{\partial p} = -c_{n-2} \\ \frac{\partial R}{\partial q} &= \frac{\partial b_{n-1}}{\partial q} = -c'_{n-3} = -c_{n-3} \\ \frac{\partial S}{\partial p} &= \frac{\partial b_n}{\partial p} + R + p \frac{\partial R}{\partial p} = -c_{n-1} - b_{n-1} + b_{n-1} - pc_{n-2} \\ \frac{\partial S}{\partial q} &= \frac{\partial b_n}{\partial q} + p \frac{\partial R}{\partial q} = -c_{n-2} - pc_{n-3} \end{aligned}$$

$$\left\{ \begin{array}{l} Det = c_{n-2}^2 - c_{n-1}c_{n-3} \\ N_p = b_{n-1}c_{n-2} - b_n c_{n-3} \\ N_q = b_n c_{n-2} - b_{n-1}c_{n-1} \end{array} \right.$$

ESIL, Algorithmes numériques 77

Méthode de Bairstow

Algorithme

- On part d'un polynôme courant $P_{cour} = \text{Polynôme initial}$
- A chaque fois que l'on trouve 2 racines, le degré de P_{cour} diminue de 2 ($P_{cour} = Q_{n-2}$)
- **Tests d'arrêt**
 - Degré de $P_{cour} \leq 2$
 - On sait directement calculer ses racines
 - Sur p et q ou sur R et S ?
 - Précision sur x ou sur f(x) ?
 - La convergence de Newton implique : $\Delta p = 0$ et $\Delta q = 0$
 - A considérer évidemment proche de 0 avec test relatif ou absolu par rapport à p et q (suivant les valeurs de p et q)

ESIL, Algorithmes numériques 78

Méthode de Bairstow

Algorithme SIMPLIFIÉ

P_{cour} = Polynôme initial ; $\text{deg}_P_{\text{cour}} = n$; processus convergent est vrai

Tantque ($(\text{deg}_P_{\text{cour}} > 2)$ et (processus convergent)) faire

$p = p_0$; $q = q_0$

Δp et Δq initialisés pour passer le test ci-dessous

Tantque (convergence sur p et q non atteinte et nombre d'itérations OK) faire

 calcul des b_i et des c_i

 calcul de Δp et Δq avec décalage si problème de dénominateur

$p = p + \Delta p$ $q = q + \Delta q$

Fintantque

Si (processus convergent) alors (soit nombre d'itérations OK)

 calculer les 2 racines de $x^2 + px + q$

 mettre à jour les b_i (avec les nouveaux p et q) ($i=0, \dots, \text{deg}_P_{\text{cour}}-2$)

$P_{\text{cour}} = Q_{n-2}$ (les b_i) ; $\text{deg}_P_{\text{cour}} = \text{deg}_P_{\text{cour}} - 2$

Finsi

Fintantque

Si (processus convergent) alors

 calculer la (ou les 2) racine(s) de P_{cour} (polynôme de degré 1 ou 2)

Finsi

ESIL, Algorithmes numériques 79

Troisième partie : Systèmes d'équations linéaires

ESIL, Algorithmes numériques 80

Généralités et définitions

- **Il existe des systèmes algébriques et semi-algébriques**
- **Il existe des systèmes linéaires et non-linéaires**
- **Système linéaire : $A.X = B$**
- **Il existe des systèmes**
 - Carrés :
 - Sur-déterminés :
 - Sous-déterminés :

ESIL, Algorithmes numériques 81

Généralités et définitions

- **Formellement : $A.X = B \Rightarrow X = A^{-1}.B$ (A carrée)**
 - Ne jamais calculer l'inverse
 - Long : revient à résoudre n systèmes
 - Numériquement par formidable
 - Inutile : on peut faire autrement
- **Système de Cramer :**
 - Complexité d'un calcul de déterminant $n!$
 - Numériquement instable !
- **Méthodes**
 - Directes : les principales
 - Gauss, LU, Cholesky, QU (Householder,...), ...
 - Itératives
 - Construire une suite X_k qui doit converger vers X (Jacobi, Gauss-Seidel, ...)
- **Penser à la possibilité d'avoir plusieurs seconds membres**



ESIL, Algorithmes numériques 82

Généralités et définitions

• **Norme matricielle : Anneau des matrices d'ordre n**

- Application vérifiant les propriétés des normes ET $\|A.B\| \leq \|A\| \|B\|$
- Ex : norme de Frobenius : $\|A\|^2 = \sum_{i=1}^n \sum_{j=1}^n |a_{ij}|^2$
- Norme subordonnée à une norme vectorielle : $\|A\| = \sup_{X \neq 0} \frac{\|A.X\|}{\|X\|}$
 - La définition implique $\|A.X\| \leq \|A\| \|X\|$

• **Théorème 1**

$$\begin{aligned} \|X\|_1 &= \sum_i |x_i| & \|A\|_1 &= \max_j \sum_i |a_{ij}| \\ \|X\|_2 &= \sqrt{\sum_i x_i^2} & \|A\|_2 &= \sqrt{\rho(A^T A)} \\ \|X\|_\infty &= \max_i x_i & \|A\|_\infty &= \max_i \sum_j |a_{ij}| \end{aligned}$$

- $\rho(A^T A)$ = rayon spectral de $A^T A$
- Les racines carrées positives des valeurs propres de $A^T A$: valeurs singulières de A (définition)
 - $\|A\|_2$: plus grande valeur singulière de A = σ_{\max}

Généralités et définitions

• **Théorème 2 : Si A est normale : $\|A\|_2 = \rho(A)$**

(toute matrice symétrique ou hermitienne est normale)

• **Théorème 3 : Si Q est orthogonale (unitaire) $\|Q.A\|_2 = \|A\|_2$**

• **Définition : Conditionnement d'une matrice (condition number)**

$$\text{Cond}(A) = \|A\| \cdot \|A^{-1}\|$$

• **Théorème 4 :**

- $\text{Cond}(A) \geq 1$
- $\text{Cond}(A) = \text{Cond}(A^{-1})$
- $\text{Cond}_2(A) = \frac{\sigma_{\max}}{\sigma_{\min}}$ $\text{Cond}_2(A) = \frac{|\lambda_{\max}|}{|\lambda_{\min}|}$ si A est normale
- Si Q est unitaire $\text{Cond}_2(Q) = 1$, $\text{Cond}_2(AQ) = \text{Cond}_2(A) = \text{Cond}_2(QA)$
- $\text{Cond}(\alpha A) = \text{Cond}(A)$ ($\alpha \neq 0$)

Généralités et définitions

- **Le rang d'une matrice $A(n,m)$ (ou rang du système) = rang du système de ses vecteurs colonnes $\text{rg } A \leq \inf(n,m)$**
- **Une matrice est de rang maximum si : $\text{rg } A = \inf(n,m)$**
- **Une matrice carrée $A(n,n)$ est inversible si $\text{rg } A = n$**
- **On ne résout que les systèmes de rang maximum**
 - Possible mais « sans intérêt » sur les systèmes rectangulaires
- **Si la matrice n'est pas de rang maximum $\sigma_{\min} = 0$**
 - $\text{Cond}_2(A) = \infty$
 - $\text{Cond}_2(A) \approx 10^7 \Leftrightarrow \infty$ en float, $\text{Cond}_2(A) \approx 10^{16} \Leftrightarrow \infty$ en double
 - On parle de rang numérique d'un système
- **Le rang d'une matrice = nombre de valeurs singulières $\neq 0$**

ESIL, Algorithmes numériques 85

Généralités et définitions

- **Influence du conditionnement : $A X = B$**
- **Soit une perturbation des données δB**
- **Soit une perturbation de A : δA**
- **L'erreur relative sur X est amplifiée par le conditionnement**
 - Majoration, mais atteinte en pratique

ESIL, Algorithmes numériques 86

Généralités et définitions

- **Exemple d'un système mal conditionné**

$$\begin{pmatrix} 6 & 13 & -17 \\ 13 & 29 & -38 \\ -17 & -38 & 50 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ -5 \end{pmatrix}$$

- **+/- 0.1 sur le second membre**

$$\begin{pmatrix} 6 & 13 & -17 \\ 13 & 29 & -38 \\ -17 & -38 & 50 \end{pmatrix} \begin{pmatrix} 2.1 \\ -1.2 \\ -0.3 \end{pmatrix} = \begin{pmatrix} 2.1 \\ 3.9 \\ -5.1 \end{pmatrix}$$

- **+/- 0.1 aléatoirement sur les coefficients de la matrice**

$$\begin{pmatrix} 6.1 & 13.1 & -17 \\ -12.9 & 29 & -38.1 \\ -16.9 & -37.9 & 50.1 \end{pmatrix} \begin{pmatrix} -0.08 \\ 2.83 \\ -2.08 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ -5 \end{pmatrix}$$

ESIL, Algorithmes numériques 87

Généralités et définitions

Une formule approchée bien utile

- **Influence des erreurs de représentation (sur B et sur A)**

- Soit c le nombre de chiffres significatifs sur la solution : $\frac{\|\delta X\|}{\|X\|} \approx 10^{-c}$
- C_{\max} associé à la plus petite perturbation (2^{-24} en float ou 2^{-53} en double) : $2^{-n} = 10^{-c_{\max}}$

$$c \approx c_{\max} - \log_{10}(\text{cond}(A))$$

- Formule approchée à 1 chiffre près environ due à J.C. Daubisse

- Ex : en float $\text{cond}(A) = 10^5$, alors $c = 7 - 5 = 2$
en float $\text{cond}(A) = 10^7$, alors $c = 7 - 7 = 0$

- **Si les erreurs sont plus grandes : remplacer c_{\max} par la valeur correspondante**

- Il y aura moins de chiffres significatifs

ESIL, Algorithmes numériques 88

2 approches

- **Remplacer le système par un système ayant les mêmes solutions**

- $A X = B \Rightarrow M.A X = M B = M' X = B'$
 - Gauss, Jordan, QU (Householder, Givens, ...)
- Attention au conditionnement de M'
- L'idéal : M orthogonale $\Rightarrow \text{Cond}_2(M') = \text{Cond}_2(A)$
 - Cas du QU, (SVD)
- Il faut évidemment que M soit inversible :
 - $M' X = B' \Rightarrow M^{-1} M' X = M^{-1} B' \Rightarrow A X = B$

- **Décomposer A en un produit de matrices remarquables**

- La résolution devient triviale
 - LU, Cholesky, (SVD)

Résolution d'un système triangulaire $A X = B$

- **Triangulaire supérieure : $a_{ij} = 0$ si**
- **Triangulaire inférieure : $a_{ij} = 0$ si**
- **Le rang du système doit être maximal :**

- **Résolution :**
$$x_k = \frac{1}{a_{kk}} \left(b_k - \sum_{j=k+1}^n a_{kj} \cdot x_j \right)$$

- **Evaluation :**
 - Algorithme « évidemment » en n^2

3 opérations de modification d'un système $A X = B$ • **Inversion de 2 lignes i et j $M(l_i \leftrightarrow l_j)$**

- Il suffit d'inverser les lignes i et j de la matrice identité
- $M.M=I \Rightarrow M$ inversible
- $\det M.A = -\det A$

• **Multiplier une ligne i par λ**

- Remplacer le terme a_{ii} de la matrice I par λ $M(l_i \leftrightarrow \lambda l_i)$
- $M(l_i \leftrightarrow \lambda l_i) M(l_i \leftrightarrow 1/\lambda l_i) = M(l_i \leftrightarrow 1/\lambda l_i) M(l_i \leftrightarrow \lambda l_i) = I \Rightarrow M$ inversible
- $\det M.A = \lambda \det A$

• **Remplacer une ligne i par ligne $i + \lambda$ ligne j**

- $M(l_i \leftrightarrow l_i + \lambda l_j) = I + M^*$ avec $M^*_{im} = \lambda \delta_{ij} \delta_{mj}$
- $M(l_i \leftrightarrow l_i + \lambda l_j)^{-1} = M(l_i \leftrightarrow l_i - \lambda l_j)$
- $\det M.A = \det A$

Cas très important de modification d'un système $A X = B$ • **Inversion de 2 colonnes**

- Possible d'après la commutativité de l'addition
- La forme matricielle, revient à changer le nom des inconnues
- Il faut
 - Garder trace des permutations
 - Résoudre
 - Remettre les inconnues dans le bon ordre
- Prendre un vecteur de n éléments initialisé avec $\text{perm}(i) = i$
- Echange des colonnes i et j :
 - échanger les contenus de $\text{perm}(i)$ et $\text{perm}(j)$
- perm mémorise toutes les inversions de colonnes
- Chaque inversion change le signe du déterminant

• **Utile pour Gauss avec pivot total, QR avec classement des colonnes par norme décroissante, ...**

Méthode de Gauss

• **Sans doute la méthode la plus connue (résultats honnêtes)**

- Considérer la première colonne
- Faire apparaître des 0 sous la diagonale
- Considérer la sous-matrice (n-1,n-1) à partir de i=2, j=2
- Recommencer le procédé

$$\begin{pmatrix} x & x & x & x \\ x & x & x & x \\ x & x & x & x \\ x & x & x & x \end{pmatrix} \Rightarrow \begin{pmatrix} x & x & x & x \\ 0 & x' & x' & x' \\ 0 & x' & x' & x' \\ 0 & x' & x' & x' \end{pmatrix} \Rightarrow \begin{pmatrix} x & x & x & x \\ 0 & x' & x' & x' \\ 0 & x' & x' & x' \\ 0 & x' & x' & x' \end{pmatrix} \Rightarrow \begin{pmatrix} x & x & x & x \\ 0 & x' & x' & x' \\ 0 & 0 & x'' & x'' \\ 0 & 0 & x'' & x'' \end{pmatrix}$$

- NE PAS OUBLIER DE TRAITER LE SECOND MEMBRE ...
- La matrice finale est triangulaire supérieure
 - Résolution directe

Méthode de Gauss

• **Description de l'étape k (k=1 à n (ou n-1))**

- $a_{ij}=0 \quad j < k, j < i$

$$\begin{pmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & 0 & a & b & c \\ 0 & 0 & d & e & f \\ 0 & 0 & g & h & l \end{pmatrix}$$

- Faire apparaître des 0 sous a : remplacer d par d - d/a * a
- Remplacer la ligne(k+1) par la ligne(k+1) - d/a ligne(k) :
 - $M(l_{k+1} \leftrightarrow l_{k+1} - (d/a) l_k)$ soit : $M(l_{k+1} \leftrightarrow l_{k+1} - (a_{k+1,k}/a_{k,k}) l_k)$
- On ne fait évidemment pas le produit M.A, on calcule directement les termes utiles. Formule générale :
- On ne calcule pas les 0 : $a_{ik}, i=k+1,n$
- Possibilité de précalculer : $a_{k+1,k}/a_{k,k}$
- Le second membre B est souvent mis en tant que (n+1) colonne de A

Méthode de Gauss

- Le terme a_{kk} est le pivot

$$\begin{pmatrix} x & x & x & x & \dots & x \\ 0 & x & x & x & \dots & x \\ 0 & 0 & a_{kk} & a_{k,k+1} & \dots & a_{kn} \\ 0 & 0 & a_{k+1,k} & & & a_{k+1,n} \\ \dots & \dots & \dots & & & \\ 0 & 0 & a_{nk} & & & a_{nn} \end{pmatrix}$$

- Et si le pivot s'« annule » ?

- Il faut changer de pivot
- Solution minimale : $M(l_k \leftrightarrow l_p)$ $p > k$
- La division par un nombre petit pose problème : mieux vaut un grand pivot

- Stratégie : pivot partiel : pivot t.q.

$$|pivot| = \sup_{p=k \dots n} |a_{pk}|$$

- Si $|pivot| < \epsilon$?

- Stratégie : pivot total : pivot t.q.

$$|pivot| = \sup_{\substack{p=k \dots n \\ l=k \dots n}} |a_{pl}|$$

- Si $|pivot| < \epsilon$?
- Modifie l'ordre des inconnues
- Optimal, mais inversion de lignes et colonnes

Méthode de Gauss

- Evaluation (1/2)

$$\begin{cases} a_{ij} = a_{ij} - \frac{a_{ik}}{a_{kk}} a_{kj} & j = k+1, n \\ b_i = b_i - \frac{a_{ik}}{a_{kk}} b_k & i = k+1, n \end{cases}$$

- Division pré-calculée
- Nouvelle ligne i :
- Avec le second membre :
- Calcul effectué pour les toutes les lignes soit :
- Et calcul effectué pour tous les k :

Méthode de Gauss

• **Evaluation (2/2)**

$$\begin{cases} a_{ij} = a_{ij} - \frac{a_{ik}}{a_{kk}} a_{kj} & j = k+1, n \\ b_i = b_i - \frac{a_{ik}}{a_{kk}} b_k & i = k+1, n \end{cases}$$

- Pour la division :
- En comptant la résolution du système triangulaire sup :

$\frac{n^3 - n}{3} + \frac{n(n-1)}{2}$	add
$\frac{n^3 - n}{3} + \frac{n(n-1)}{2}$	mul
$\frac{n(n-1)}{2} + n$	div

Algorithme en :
 $n^3/3$ add et mul et $n^2/2$ div

heuristique : $\max(|a_{ii}|)/\min(|a_{ii}|)$
une idée de $\text{cond}(A)$

ESIL, Algorithmes numériques 97

Méthode du LU

- **Méthode classique**
- **A est décomposée en L.U**
 - L triangulaire inférieure
 - U triangulaire supérieure
 - Décomposition non unique
 - Méthode Crout $u_{ii}=1$

- **L.U X=B**
 - LY=B puis UX=Y
 - La décomposition ne modifie pas B

$$\begin{pmatrix} a_{11} & a_{12} & & a_{1n} \\ a_{21} & a_{22} & & \\ & & a_{jj} & \\ a_{n1} & & & a_{nn} \end{pmatrix} = \begin{pmatrix} l_{11} & 0 & 0 & 0 \\ & l_{22} & 0 & 0 \\ & & l_{jj} & 0 \\ l_{n1} & & & l_{nn} \end{pmatrix} \begin{pmatrix} 1 & u_{12} & & u_{1n} \\ 0 & 1 & u_{22} & \\ 0 & 0 & 1 & u_{jn} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

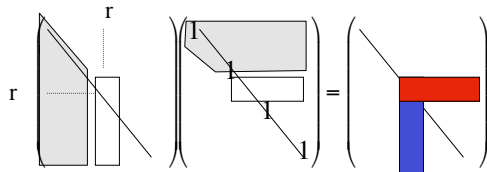
ESIL, Algorithmes numériques 98

Méthode du LU

Construction de :
L par colonnes
U par lignes

$$l_{ir} = a_{ir} - \sum_{j=1}^{r-1} l_{ij} u_{jr} \quad i = r, \dots, n$$

$$u_{ri} = \frac{1}{l_{rr}} \left(a_{ri} - \sum_{j=1}^{r-1} l_{rj} u_{ji} \right) \quad i = r+1, \dots, n$$



- Il faut $l_{rr} \neq 0$
 - Même principe que Gauss
 - Pivot sur la colonne $l_{jr}, j > r$ (le plus grand). Pivot « nul » : non inversible !
 - On inverse les lignes
 - Vecteur perm pour le second membre !!
- On peut stocker U et L sur A
 - $A = L + U - I$

Méthode du LU

Evaluation de la décomposition

$$l_{ir} = a_{ir} - \sum_{j=1}^{r-1} l_{ij} u_{jr} \quad i = r, \dots, n$$

$$u_{ri} = \frac{1}{l_{rr}} \left(a_{ri} - \sum_{j=1}^{r-1} l_{rj} u_{ji} \right) \quad i = r+1, \dots, n$$

- $r=1, \dots, n$
 - Pour les l_{ij} :
 - $(n-r+1)$ termes et $(r-1)$ add et mul
 - Pour les u_{ij} :
 - 1 division
 - $(n-r)$ termes et $(r-1)$ add et mul 1 mul sup $(1/l_{rr} * (...))$
 - Il faut ajouter deux résolutions de systèmes triangulaires
 - Pas plus intéressant que Gauss
 - Pas de traitement du second membre
 - Peut être intéressant

$\frac{2n^3 - 3n^2 + n}{6}$	add
$\frac{n^3 - n}{3}$	mul
$\frac{n(n-1)}{2}$	div

Méthode de Cholesky

- **S'applique sur les matrices symétriques définies positives**
 - Matrices pour les méthodes des éléments finis
 - Equations normales des problèmes aux moindres carrés
- **Ne peut s'appliquer que sur ces matrices**
 - 1 racine carrée et 1 division
 - Permet de contrôler que les matrices ont la bonne forme
- **A est décomposée en L.L^T (L triangulaire inférieure)**
 - L.L^TX=B
 - LY=B puis L^TX=Y

$$\begin{pmatrix} a_{11} & a_{12} & a_{1n} \\ a_{21} & a_{22} & \\ & a_{jj} & \\ a_{n1} & & a_{nn} \end{pmatrix} = \begin{pmatrix} l_{11} & 0 & 0 & 0 \\ & l_{22} & 0 & 0 \\ & & l_{jj} & 0 \\ & & & l_{nn} \end{pmatrix} \begin{pmatrix} l_{11} & & & l_{1n} \\ 0 & l_{22} & & \\ 0 & 0 & l_{jj} & \\ 0 & 0 & 0 & l_{nn} \end{pmatrix}$$

Méthode de Cholesky

Construction par colonnes

$$\begin{pmatrix} a_{11} & a_{12} & a_{1n} \\ a_{21} & a_{22} & \\ & a_{jj} & \\ a_{n1} & & a_{nn} \end{pmatrix} = \begin{pmatrix} l_{11} & 0 & 0 & 0 \\ & l_{22} & 0 & 0 \\ & & l_{jj} & 0 \\ & & & l_{nn} \end{pmatrix} \begin{pmatrix} l_{11} & & & l_{1n} \\ 0 & l_{22} & & \\ 0 & 0 & l_{jj} & \\ 0 & 0 & 0 & l_{nn} \end{pmatrix}$$

– $l_{jj} = \sqrt{a_{jj} - \sum_{l=1}^{j-1} l_{jl}^2}$ (choix de la racine positive, la racine carrée existe si A correcte)

$l_{kj} = \frac{1}{l_{jj}} \left(a_{kj} - \sum_{l=1}^{j-1} l_{kl} l_{jl} \right) \quad k = j + 1, \dots, n \quad (l_{jj} \neq 0 \text{ si A correcte})$

expression directe de A=L.L^T, L stockée sur A

– Evaluation :

$\frac{n^3 - n}{6} + n(n - 1)$	add
$\frac{n^3 - n}{6} + n(n - 1)$	mul
$\frac{n(n - 1)}{2} + 2n$	div
n	rac

Algorithme en :
n³/6 add et mul et n²/2 div

Méthodes itératives

- **Résoudre $A.X=B$**
 - Prendre X_0 quelconque
 - Construire la suite : $X^{k+1}=M.X^k+C \quad k \geq 0$
 - Telle que : $\lim_{k \rightarrow \infty} X^k = X$
 - Il faut savoir gérer un bon test d'arrêt sur un vecteur !
- **Théorème : Les méthodes itératives convergent si $\rho(M)<1$**
- **Il faut que A soit « particulière » pour assurer la convergence**
 - Si A est à diagonale dominante, $\rho(M_{\text{Jacobi}})<1$ et $\rho(M_{\text{gauss-Seidel}})<1$
 - La convergence
 - Peut ne pas exister, être très lente (accélération possible)
 - Elle ne dépend pas de X_0 (mais la vitesse de convergence en dépend)
- **2 intérêts**
 - Matrices creuses
 - Solutions approximatives

ESIL, Algorithmes numériques 103

Méthodes itératives

- **Jacobi**

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \dots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = b_n \end{cases}$$

– soit $X_0 = (x_0, x_1, \dots, x_n)^T$

• Par exemple
$$X_0 = \begin{pmatrix} \frac{b_1}{a_{11}} & \frac{b_2}{a_{22}} & \dots & \frac{b_n}{a_{nn}} \end{pmatrix}^T$$

– On peut calculer X^{k+1} :
$$x_i^{k+1} = \frac{1}{a_{ii}} (b_i - \sum_{j=1, j \neq i}^n a_{ij}x_j^k)$$

– Calcul très simple et rapide des itérés :

- On injecte dans le système les valeurs de l'étape d'avant

- **Coefficient diagonal nul**

– Inversion de lignes

ESIL, Algorithmes numériques 104

Méthodes itératives

• Gauss-Siedel

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \dots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = b_n \end{cases}$$

– soit $X_0 = (x_0, x_1, \dots, x_n)^T$

– Simple amélioration de Jacobi :

- Dès que x_j^{k+1} est calculé, pourquoi utiliser x_j^k ?

$$x_i^{k+1} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{k+1} - \sum_{j=i+1}^n a_{ij}x_j^k \right)$$

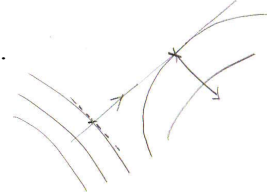
– Calcul très simple et rapide des itérés

ESIL, Algorithmes numériques 105

Méthodes itératives

• Méthodes de descente :

- « a priori » sur des matrices symétriques définies positives
- Font partie des méthodes d'optimisation vues en 2 année (MIRA)
- On minimise la forme quadratique : $f(x) = x^T A x - B^T x$ (f de \mathbb{R}^n dans \mathbb{R})
 - On doit avoir $\text{grad}(f) = 0$
 - $\text{grad}(f) = A x - B$
- Un point de départ, une direction de descente : on minimise dans cette direction
- De ce nouveau point, nouvelle direction de descente et nouvelle minimisation, ...
- Les méthodes diffèrent dans la méthode de minimisation directionnelle et dans le choix des directions de descente
- Plus grande pente, gradient conjugué, bi-conjugué, ...

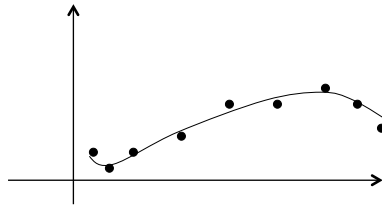


ESIL, Algorithmes numériques 106

Problèmes de moindres carrés

• Problèmes excessivement classiques et TRES importants

– Un exemple



- n points du plan (x_i, y_i)
- On cherche à faire passer une courbe « au mieux » par ces points (éventuellement erronés)
- On choisit (par exemple) de prendre un polynôme P
- On choisit son degré (m) (avec $m+1 < n$)

- Cela revient à écrire $P(x_i) = y_i \quad i=1, \dots, n$
- $(m+1)$ coefficients inconnus de P : $P(x) = a_m x^m + a_{m-1} x^{m-1} + \dots + a_0$
- Soit : $M \cdot X_a = Y$, système linéaire à n lignes et $m+1$ colonnes
- Le système est surdéterminé (trop contraint) ($m+1 = n$: système carré)
- Nombreux autres exemples (droite de régression linéaire, plan moyen, ..., les courbes de TP de physique !)

ESIL, Algorithmes numériques 107

Problèmes de moindres carrés

• Problème surdéterminé : recherche d'une solution avec une erreur

• L'approche la plus classique : approche aux moindres carrés (least square approach)

– Recherche des $(m+1)$ coefficients de P permettant d'obtenir :

$$e = \min F(a_m, \dots, a_0) = \min \sum_{i=1}^n \|P(x_i) - y_i\|_2^2$$

- Il s'agit de la minimisation d'une fonction F à $(m+1)$ inconnues
- e est atteint si : $\text{Grad } F = 0$

– Si $\frac{\partial F}{\partial a_j} = 0 \quad j = 0, \dots, m$

ESIL, Algorithmes numériques 108

Problèmes de moindres carrés

$$\min F(a_m, \dots, a_0) = \min \sum_{i=1}^n \|P(x_i) - y_i\|_2^2$$

$$j = 0, \dots, m \quad \frac{\partial F}{\partial a_j} = 0 = 2 \left(\sum_{i=1}^n (P(x_i) - y_i) x_i^j \right)$$

$$j = 0, \dots, m \quad \sum_{i=1}^n P(x_i) x_i^j = \sum_{i=1}^n y_i x_i^j$$

$$\begin{pmatrix} x_1^m & \dots & x_n^m \\ \dots & \dots & \dots \\ 1 & \dots & 1 \end{pmatrix} \begin{pmatrix} x_1^m & \dots & 1 \\ \dots & \dots & \dots \\ x_n^m & \dots & 1 \end{pmatrix} \begin{pmatrix} a_m \\ \dots \\ a_0 \end{pmatrix} =$$

$$\begin{pmatrix} x_1^m & \dots & x_n^m \\ \dots & \dots & \dots \\ 1 & \dots & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ \dots \\ y_n \end{pmatrix} =$$

Problèmes de moindres carrés

- **Approche aux moindres carrés**
- **Revient à résoudre :** $M^T M \cdot X_a = M^T \cdot Y$
- **Problème linéaire (m+1, m+1)**
- **On montre facilement qu'il s'agit bien d'un minimum**
- **Un problème aux moindres carrés est « simple » à résoudre**
- **Développé sur un exemple**
 - mais se généralise facilement (voir [5])

Résolution aux moindres carrés

• **Résoudre aux moindres carrés $A.X=B$**

- A nl lignes et nc colonnes
- Rechercher X qui minimise la fonction f : $f(X) = \|A.X - B\|_2^2$
 - Si $nl=nc$
 - Si $nl>nc$

• **Rappel (théorème 3)**

- Si Q est orthogonale : $\|A.X - B\|_2 = \|Q.A.X - Q.B\|_2$
- Il existe des transformations orthogonales qui conduisent à

$$\begin{array}{c}
 \uparrow \\
 \text{nl} \\
 \left(\begin{array}{cccc}
 \cdot & \cdot & \cdot & \cdot \\
 0 & \cdot & \cdot & \cdot \\
 0 & 0 & \cdot & \cdot \\
 0 & 0 & 0 & \cdot \\
 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0
 \end{array} \right) \\
 \downarrow \\
 \text{nc}
 \end{array}$$

Q.A composée de :

- $A_1(nc,nc)$
- $A_2(nl-nc,nc)$

Transformation de Householder

Transformation de Givens

Résolution aux moindres carrés

• **Il faut résoudre**

$$\begin{array}{c}
 \mathbf{A}_1 \\
 \mathbf{A}_2
 \end{array}
 \begin{array}{c}
 \left(\begin{array}{cccc}
 \cdot & \cdot & \cdot & \cdot \\
 0 & \cdot & \cdot & \cdot \\
 0 & 0 & \cdot & \cdot \\
 0 & 0 & 0 & \cdot \\
 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0
 \end{array} \right)
 \begin{array}{c}
 \left(\begin{array}{c} \\ \\ \\ \\ \\ \end{array} \right)
 -
 \begin{array}{c}
 \left(\begin{array}{c} \\ \\ \\ \\ \\ \end{array} \right)
 =
 \begin{array}{c}
 \left(\begin{array}{c} \\ \\ \\ \\ \\ \end{array} \right)
 \end{array}
 \end{array}$$

- Le vecteur R est appelé résidu
- Il faut trouver le minimum de $\|R\|_2$

- On a $R_1 = A_1.X - B_1$
 $R_2 = B_2$

Résolution aux moindres carrés

• Théorème de « Pythagore »

$$\|A.X - B\|_2^2 = \|R_1\|_2^2 + \|R_2\|_2^2$$

$$\begin{matrix} A_1 & \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & \cdot & \cdot \\ 0 & 0 & \cdot & \cdot \\ 0 & 0 & 0 & \cdot \end{pmatrix} & \cdot & \begin{pmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \end{pmatrix} & - & \begin{pmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \end{pmatrix} & = & \begin{pmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \end{pmatrix} \\ A_2 & \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} & & & & \begin{pmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \end{pmatrix} & = & \begin{pmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \end{pmatrix} \end{matrix}$$

– Donc $\min\|A.X - B\|_2^2 = \min\|R_1\|_2^2 + \min\|R_2\|_2^2$

– $\min\|R_1\|_2^2$?

– $\min\|R_2\|_2^2$?

• Résoudre $A.X=B$ aux moindres carrés $\Leftrightarrow A^T A.X=A^T B$

– L'erreur commise est $\|R_2\|_2^2$

– **La différence** : $\text{cond}_2(A^T A)=\text{cond}_2(A)^2$!!!!!

– ~~$A^T A.X=A^T B$~~

Il faut résoudre $A.X=B$ aux moindres carrés

Résolution aux moindres carrés

• Transformation QU ou transformation de Householder

• **Lemme** : u un vecteur de R^n normé ($u^T u=1$)
la matrice $H = I - 2 u.u^T$ est symétrique et orthogonale

– $u^T.u$ et $u.u^T$

– démonstration :

• **Théorème** : soit a de R^n $a^T=(a_1, a_2, \dots, a_n)$, alors il existe une matrice H , t.q. $H.a = \alpha e_1$ ($e_1^T=(1, 0, \dots, 0)$),

– En d'autres termes, si a est une colonne de matrice, H fait apparaître un coefficient en première position et des 0 en dessous

– $H = I - \frac{1}{\eta} \gamma.\gamma^T$ avec $\begin{cases} \alpha = -(\text{signe}(a_1))\|a\| \\ \eta = \alpha(\alpha - a_1) \\ \gamma = a - \alpha.e_1 \end{cases}$

Résolution aux moindres carrés

- En pratique, à l'étape r : $H = I - \frac{1}{\eta} \gamma \gamma^T$ avec $\begin{cases} \alpha = -(\text{signe}(a_1)) \|a\| \\ \eta = \alpha(\alpha - a_1) \\ \gamma = a - \alpha e_1 \end{cases}$

$$A^{(r)} = \begin{pmatrix} \Gamma & & \\ \mathbf{0} & & \\ & & a \end{pmatrix} \begin{matrix} \updownarrow \\ n-r+1 \end{matrix}$$

On définit H_r ($n-r+1, n-r+1$) avec la colonne a

Et $H^{(r)} : \begin{pmatrix} I & 0 \\ 0 & H_r \end{pmatrix}$ ($H^{(r)}$ est orthogonale)

$$A^{(r+1)} = H^{(r)} A^{(r)} = \begin{pmatrix} I & 0 \\ 0 & H_r \end{pmatrix} \begin{pmatrix} A_{11}^{(r)} & A_{21}^{(r)} \\ 0 & A_{22}^{(r)} \end{pmatrix} = \begin{pmatrix} A_{11}^{(r)} & A_{21}^{(r)} \\ 0 & H_r A_{22}^{(r)} \end{pmatrix}$$

$$B^{(r+1)} = H^{(r)} B^{(r)} = \begin{pmatrix} I & 0 \\ 0 & H_r \end{pmatrix} \begin{pmatrix} B_1^{(r)} \\ B_2^{(r)} \end{pmatrix} = \begin{pmatrix} B_1^{(r)} \\ H_r B_2^{(r)} \end{pmatrix}$$

- On travaille donc sur des sous-matrices $r=1, \dots, n_c$
 - on peut diviser par η le plus grand possible, soit avec la norme de colonne la plus grande \Rightarrow inversion de colonne (vecteur perm !)
 - plus grande norme $\ll = \gg 0 \Rightarrow$ rang non maximum

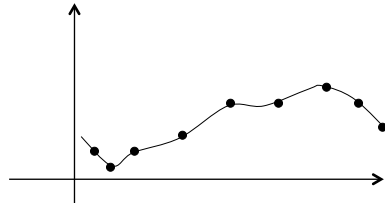
Quatrième partie : Différentiation et Intégration numériques

Approximation par un polynôme

- Soit une fonction f connue en $(n+1)$ points x_0, \dots, x_n

- f supposée suffisamment dérivable
- Par $(n+1)$ points, il passe un polynôme de degré n P_n

- $f(x) = P_n(x) + E_n(x)$
 - $E_n(x) = 0$ pour $x = x_i, i = 0, \dots, n$
 - $E_n(x)$ fait intervenir $f^{(n+1)}(x)$



- Si on maîtrise E_n , on peut remplacer f par P_n et intégrer ou dériver P_n
- On peut aussi considérer les développements de Taylor

ESIL, Algorithmes numériques 117

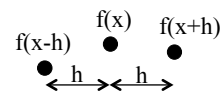
Différentiation numérique

- Soit une fonction f connue en $(n+1)$ points x_0, \dots, x_n

- Le problème est d'estimer un certain nombre de dérivées en $x_0, \dots,$

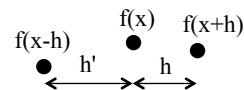
- Les formules les plus connues

- $f'(x) = [f(x+h) - f(x)]/h$
 - Différence avant d'ordre 1 (D.L en $x+h$)



- $f'(x) = [f(x) - f(x-h)]/h$
 - Différence arrière d'ordre 1 (D.L en $x-h$)

- $f'(x) = [f(x+h) - f(x-h)]/2h$
 - Différence centrée d'ordre 1 (2D.L en $x+h$ et $x-h$)
 - Plus générale : $f'(x) = [f(x+h) - f(x-h')]/(h+h')$



- On peut faire des formules d'ordre 2 en combinant 2 DL d'ordre 2
 - En $x+h$ et $x+2h$, ou $x-h$ et $x-2h$, ou $x-h$ et $x+h$

- Problème très instable

- grosses approximations et différences de quantités voisines
- Les données sont bruitées \Rightarrow grosses instabilités

ESIL, Algorithmes numériques 118

Différentiation numérique

- **Estimation de f' : Appliquer les mêmes formules aux valeurs $f'(x_i)$ calculées auparavant**
 - Instabilités garanties !
- **Estimation de f' : Lisser les résultats**
 - $f(x)=P_n(x)+E_n(x)$
 - $f'=P_n'$ et $f''=P_n''$
 - Calculer $f'(x_i)$
 - Approcher f' par un polynôme $f'(x)=Q_m(x)+E_m(x)$
 - $f''(x)=Q_m'(x)$
- **Estimer des dérivées est très utile en modélisation géométrique et en imagerie**
 - C'est toujours un problème difficile
 - On estime aussi des dérivées partielles

ESIL, Algorithmes numériques 119

Intégration numérique

- **[a,b] un intervalle et f une fonction intégrable**

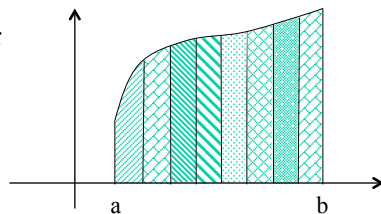
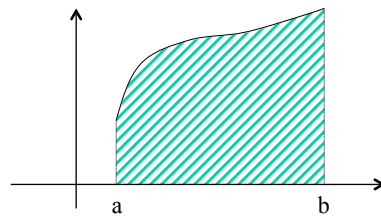
– On cherche $I = \int_a^b f(x)dx$

- En pratique on découpe [a,b] en
- n intervalles de longueur h
- $x_0=a, x_1=a+h, \dots, x_n=a+n.h=b$

– On calcule : $I = \sum_{i=1}^n \int_{x_{i-1}}^{x_i} f(x)dx$

- **Plus le pas h est petit**

- Plus la méthode est précise
- Plus les calculs sont longs



ESIL, Algorithmes numériques 120

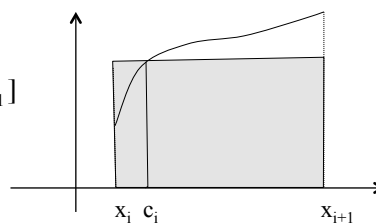
Intégration numérique

- **Sur chaque intervalle $[x_i, x_{i+1}]$:**
 - $f(x) = P_n(x) + E_n(x) \Rightarrow \int_{x_i}^{x_{i+1}} f(x) dx = \int_{x_i}^{x_{i+1}} P_n dx + \int_{x_i}^{x_{i+1}} E_n dx$
- **Ordre d'une méthode d'intégration**
 - Ou degré de précision
 - Le nombre n maximum pour lequel la méthode permet d'intégrer exactement un polynôme de degré inférieur ou égal à n
- **2 possibilités d'action :**
 - Le pas h d'intégration (nombre d'intervalles)
 - Le choix de la formule donc l'ordre de la méthode
- **L'intégration revient à faire des moyennes, de globaliser**
 - Procédé numériquement stable
 - Contrairement à la dérivation

ESIL, Algorithmes numériques 121

Intégration numérique

- **La méthode des rectangles :**
 - $\int_{x_i}^{x_{i+1}} f(x) dx = (x_{i+1} - x_i) f(c_i) \quad c_i \in [x_i, x_{i+1}]$
- $$\int_a^b f(x) dx = \sum_{i=0}^{n-1} h f(c_i) \quad c_i \in [x_i, x_{i+1}]$$
- Si $c_i = x_i$ ou $c_i = x_{i+1}$
 - La méthode est d'ordre 0
 - Si $c_i = (a+b)/2$
 - La méthode est d'ordre 1
- **Ce n'est pas une méthode performante !**



ESIL, Algorithmes numériques 122

Intégration numérique

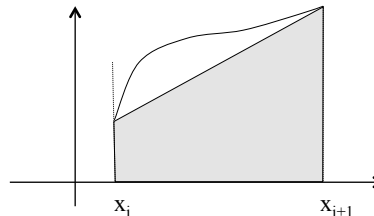
• **La méthode des trapèzes :**

$$- \int_{x_i}^{x_{i+1}} f(x)dx = \left(\frac{x_{i+1} - x_i}{2}\right)(f(x_i) + f(x_{i+1}))$$

$$\int_a^b f(x)dx = \sum_{i=0}^{n-1} \frac{h}{2}(f(x_i) + f(x_{i+1}))$$

- Que l'on écrit :

$$\int_a^b f(x)dx = h \left[\frac{f(x_0) + f(x_n)}{2} + \sum_{i=1}^{n-1} f(x_i) \right]$$



• **C'est la méthode de base minimale**

- Méthode d'ordre 1

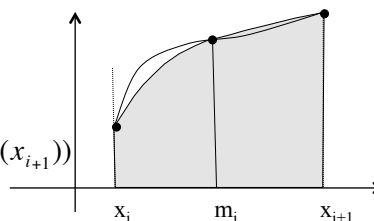
• **On peut montrer que l'erreur est en h³**

Intégration numérique

• **La méthode de Simpson**

- Prendre 3 points et de faire passer une parabole par ces 3 points (on prend en général le point milieu)
- Intégrer cette parabole
- Méthode évidemment d'ordre 2

$$\int_{x_i}^{x_{i+1}} f(x)dx = \left(\frac{x_{i+1} - x_i}{6}\right)(f(x_i) + 4f(m_i) + f(x_{i+1}))$$



$$\int_a^b f(x)dx = \sum_{i=0}^{n-1} \frac{h}{6}(f(x_i) + 4f(m_i) + f(x_{i+1}))$$

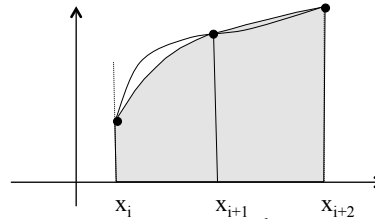
- Que l'on écrit :

$$\int_a^b f(x)dx = [f(x_0) + f(x_n) + 2 \sum_{i=1}^{n-1} f(x_i) + 4 \sum_{i=0}^{n-1} f(m_i)] \cdot \frac{h}{6}$$

Intégration numérique

• La méthode de Simpson (suite)

- Deux notations possibles
- $x_{i+1} - x_i = h, m_i \in [x_i, x_{i+1}]$
 - formules précédentes
- $x_{i+1} - m_i = h, m_i = x_{i+1}$
 - Intégration sur $[x_i, x_{i+2}]$
 - n intervalles, mais $2n+1$ points



$$\int_{x_i}^{x_{i+2}} f(x) dx = \left(\frac{x_{i+2} - x_i}{6}\right)(f(x_i) + 4f(x_{i+1}) + f(x_{i+2}))$$

$$\int_a^b f(x) dx = \sum_{i=1}^n \frac{h}{3} (f(x_{2(i-1)}) + 4f(x_{2i-1}) + f(x_{2i}))$$

- Que l'on écrit :

$$\int_a^b f(x) dx = [f(x_0) + f(x_{2n}) + 2 \sum_{i=1}^{n-1} f(x_{2i}) + 4 \sum_{i=1}^n f(x_{2i-1})] \cdot \frac{h}{3}$$

ESIL, Algorithmes numériques 125

Intégration numérique

• La méthode de Simpson une bonne méthode

- Sous réserve d'un pas adapté aux éventuelles oscillations de f

• On peut montrer que l'erreur est en h^5

- Calcul un peu délicat
- Plus précise que ce qui aurait pu être attendu

• En partant de ces méthodes

- En subdivisant le pas par 2
- En combinant les résultats entre les subdivisions successives
- On obtient des résultats très précis
 - C'est la méthode Romberg

ESIL, Algorithmes numériques 126

Intégration numérique

• **Quadrature de Gauss**

- Chercher des points astucieux pour évaluer la fonction à intégrer pour diminuer le nombre de calculs

$$\int_{-1}^1 f(x) dx = \sum_{i=1}^n \omega_i f(x_i)$$

- n points d'intégration : la quadrature de Gauss est d'ordre 2n-1
 - Approximativement : 2n coefficients (ω_i et x_i). On peut gérer 2n contraintes pour les monômes t^i , $i=0, \dots, 2n-1$ (base des polynômes de degré $\leq 2n-1$)
 - 2 exact pour les polynômes de degré 3
 - 3 exacte pour les polynômes de degré 5
 -
 - Voir par exemple [8] ou wikipédia pour les démonstrations
- Fournit de très bons résultats et très peu de calcul
- Très utile en mécanique où les évaluations de f sont très lourdes
- Pas de contrôle de la précision en jouant sur h

Intégration numérique

• **Quadrature de Gauss**

- Formule à 1 point ($\omega_1=2, x_1=0$) $\int_{-1}^1 1 dx = 2 = \omega_1 \cdot 1$ $\int_{-1}^1 x dx = \left[\frac{x^2}{2} \right]_{-1}^1 = 0 = \omega_1 \cdot x_1$
- Formule à 2 points

$$\int_{-1}^1 1 dx = 2 = \omega_1 \cdot 1 + \omega_2 \cdot 1$$

$$\int_{-1}^1 x dx = 0 = \omega_1 \cdot x_1 + \omega_2 \cdot x_2 \longrightarrow *x_1^2$$

$$\int_{-1}^1 x^2 dx = \frac{2}{3} = \omega_1 \cdot x_1^2 + \omega_2 \cdot x_2^2$$

$$\int_{-1}^1 x^3 dx = 0 = \omega_1 \cdot x_1^3 + \omega_2 \cdot x_2^3 \quad \downarrow \longrightarrow \omega_2 \cdot x_2 (x_2^2 - x_1^2) = 0$$

$$\int_{-1}^1 f(x) dx = f\left(-\sqrt{\frac{1}{3}}\right) + f\left(\sqrt{\frac{1}{3}}\right)$$

Intégration numérique

- **Quadrature de Gauss** (formules approchées)

- $\int_{-1}^1 f(x)dx = f(-\sqrt{\frac{1}{3}}) + f(\sqrt{\frac{1}{3}})$

- $\int_{-1}^1 f(x)dx = \frac{5}{9}f(-\sqrt{\frac{3}{5}}) + \frac{8}{9}f(0) + \frac{5}{9}f(\sqrt{\frac{3}{5}})$

-

- **Intervalle [a,b] quelconque**

- $I = \int_a^b f(x)dx = \frac{b-a}{2} \int_{-1}^1 f(\frac{b-a}{2}t + \frac{a+b}{2})dt$

- $I = \int_a^b f(x)dx = \frac{b-a}{2} \sum_{i=1}^n \omega_i f(\frac{b-a}{2}x_i + \frac{a+b}{2})$

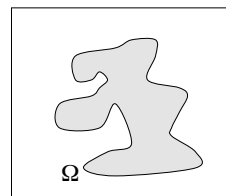
ESIL, Algorithmes numériques 129

Intégrales multiples

- **Mêmes idées mais**

- Beaucoup plus de calculs
- Problème du domaine d'intégration

$$\iint_{\Omega} f(x,y)dx dy$$

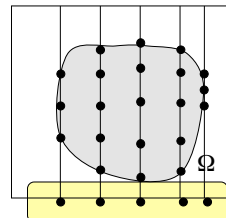


- **On essaie de se ramener à :**

$$\int_{x_0}^{x_1} \int_{y_0(x)}^{y_1(x)} f(x,y)dx dy$$

- Le cas idéal fréquent :

$$\int_a^b \int_c^d f(x,y)dx dy$$



- **Méthodes classiques**

- Quadrature de Gauss multivariables
- Généralisation des formules précédentes

ESIL, Algorithmes numériques 130

Intégrales multiples

- **Méthode de Monte-Carlo**

- Approche très générale et très utilisée de résolution de problèmes
- Approche probabiliste : Tirage aléatoire d'un ensemble de n valeurs

$$\int_a^b f(x)dx = (b-a)E(f(U))$$

- $E(f(U))$: espérance de la variable aléatoire uniforme $f(U)$

$$E(f(U)) = \lim_{n \rightarrow \infty} \frac{1}{n} \sum_{i=1}^n f(x_i)$$

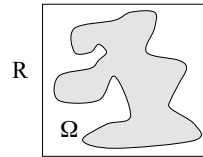
- **En dimension supérieure (ici 2):**

- On place Ω dans un rectangle R
- On remplace f par f^*

$$f^*(x,y) = \begin{cases} f(x,y) & \text{si } (x,y) \in \Omega \\ 0 & \text{autrement} \end{cases}$$

- On tire aléatoirement n valeurs sur R

$$\iint_{\Omega} f(x,y)dxdy = \text{aire}(R) \cdot \left[\frac{1}{n} \sum_{i=1}^n f^*(x_i, y_i) \right]$$



ESIL, Algorithmes numériques 131

Cinquième partie : Interpolation - Approximation - Lissage : notions

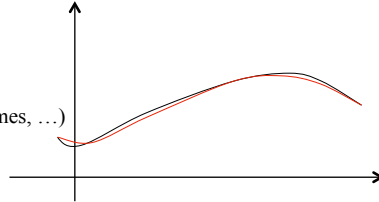
ESIL, Algorithmes numériques 132

Problème

- **Soit une fonction f connue analytiquement ou en tout point**

- Remplacer f par Φ^* t.q. $\|f - \Phi^*\|$

- Norme fonctionnelle
- Φ^* exprimée dans une bonne base
 - (fonctions exponentielles, polynômes, ...)



- On remplace f par Φ^* :

- Φ^* moins lourde à calculer
- Φ^* a des propriétés connues (ex: primitives)
- On supprime les irrégularités non désirées de f

- **On parle d'approximation**

- Domaine de la théorie de l'approximation

- Bon espace fonctionnel
- Bonne base de fonctions d'approximation
- Bonne norme

ESIL, Algorithmes numériques 133

Problème

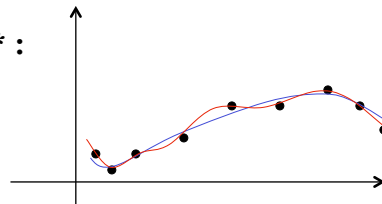
- **On se ramène toujours à un problème discret**

n points de l'espace considéré

On cherche à faire passer une courbe Φ^* :

- par ces points (**interpolation**)

$$\Phi^*(x_i) = f(x_i), \quad i = 1, \dots, n$$



- au mieux au sens d'une norme parmi ces points (**approximation ou lissage**)

$$\min \sum_{i=1}^n \|\Phi(x_i) - f(x_i)\|^2 \quad \text{ou} \quad \min(\max \|\Phi(x_i) - f(x_i)\|) \quad \text{ou} \quad \dots$$

- **L'interpolation est un problème classique**

- Plus contraint
- Plus oscillant (données erronées)

ESIL, Algorithmes numériques 134

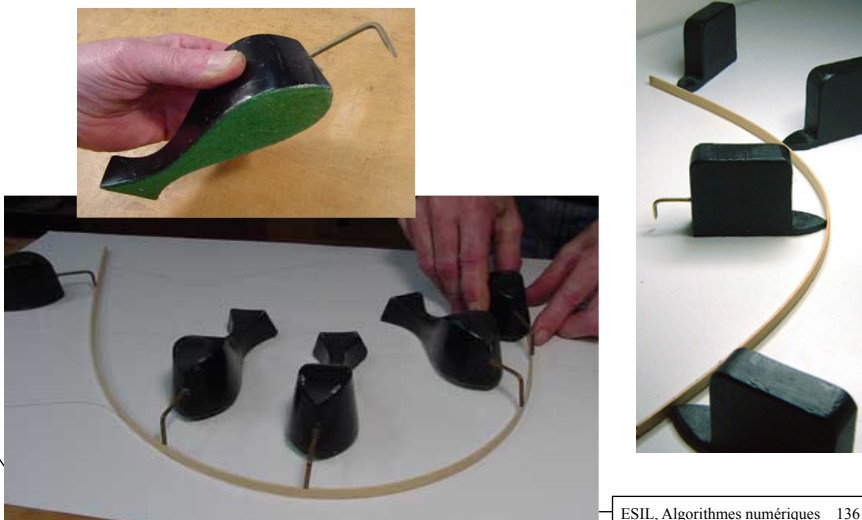
Interpolation

- **Exemple 1 : polynôme de Lagrange** $P(x) = \sum_{i=1}^n y_i \prod_{j=1, j \neq i}^n \frac{(x - x_j)}{(x_i - x_j)}$
 - Connu pour ses fortes oscillations
entre les points d'appui
- **Exemple 2 : polynôme dans la base canonique**
 - $P(x_i) = y_i \quad i=1, \dots, n$
 - n coefficients inconnus de $P : P(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \dots + a_0$
 - Système linéaire (n, n) : connu pour être instable
 - conditionnement de la matrice élevé (type Van der Monde)

ESIL, Algorithmes numériques 135

Interpolation

- **Exemple 3 : Spline d'interpolation (voir cours 3e année)**
 - Reproduction du comportement du latte (spline) de Plexiglas
 - Beaucoup mieux que les exemples 1 et 2



ESIL, Algorithmes numériques 136

Meilleure approximation

- **Soit E un espace vectoriel fonctionnel préhilbertien**
 - (E.V. muni d'un produit scalaire, E complet \Rightarrow E espace de Hilbert)
 - F un sous-espace vectoriel de dimension finie

- **Théorème de la projection orthogonale**

- f une fonction de E
- Il existe Φ^* unique de F qui réalise la meilleure approximation de f par des éléments de F

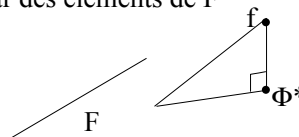
$$\|f - \Phi^*\| = \min_{\Phi \in F} \|f - \Phi\|$$

- **Φ^* est la projection orthogonale de f sur F**

- C'est la meilleure approximation de f par des éléments de F

- **CNS : Φ^* vérifie**

$$\forall \Phi \in F, \quad \langle f - \Phi^*, \Phi \rangle = 0$$



ESIL, Algorithmes numériques 137

Lissage

- **Exemple 1 : polynôme dans la base canonique**

- $P(x_i) = y_i \quad i=1, \dots, n$
- Approximation par un polynôme de degré m ($m < n-1$)
- m+1 coefficients inconnus de P : $P(x) = a_m x^m + a_{m-1} x^{m-1} + \dots + a_0$
- On considère le produit scalaire classique
 - problème aux moindres carrés
- Système linéaire (n, m+1) : connu pour être instable
 - Surtout pour m grand

- **Pour éviter l'instabilité numérique**

- Les Bézier, splines, B-splines, ... (cours de 3e année)
- On considère des bases de polynômes orthogonaux

ESIL, Algorithmes numériques 138

Lissage

• Exemple 2 : base de polynômes orthogonaux

- Produit scalaire continu $\langle P_l, P_k \rangle = \int_l P_l(x) P_k(x) \omega(x) dx$
- Produit scalaire discret $\langle P_l, P_k \rangle = \sum_{i=1}^n P_l(x_i) P_k(x_i) \omega(x_i)$
- On fixe le degré : $\Phi^*(x) = \sum_{i=0}^{\text{deg}} a_i^* P_i(x)$
- En appliquant la CNS : $\langle f - \Phi^*, \Phi \rangle = 0$ appliquée pour tous les P_k
$$\langle f, P_k \rangle = \sum_{i=1}^n a_i^* \langle P_i, P_k \rangle \quad k = 0, \dots, \text{deg}$$
- Système linéaire (diagonal dans les meilleures situations) beaucoup plus stable

ESIL, Algorithmes numériques 139

Lissage

• Exemple 2 : base de polynômes orthogonaux (suite)

- La plus connue : base de Thebychev, base orthogonale avec
- Produit scalaire continu $\langle P_l, P_k \rangle = \int_{[-1,1]} \frac{P_l(x) P_k(x)}{\sqrt{1-x^2}} dx$
- Polynômes définis par : $P_n(\cos(x)) = \cos(nx)$
- $P_0(x) = 1, P_1(x) = x, P_{n+1}(x) = 2xP_n(x) - P_{n-1}(x) \quad n \geq 1$
- Sur $[a, b]$, changement de variable pour se ramener à $[-1, 1]$
 - $u = 1/2[(b-a)x + (b+a)]$
- Il existe des points de Gauss adaptés à l'intégration rapide
- Produit scalaire discret, les polynômes sont orthogonaux si les points d'appui sont :
$$\cos\left(\frac{2i+1}{2(n-1)+2} \pi\right), \quad i = 0, \dots, n-1$$
 - Le système est diagonal
 - Sinon, la matrice est pleine

ESIL, Algorithmes numériques 140

Dernière partie : Conclusion

ESIL, Algorithmes numériques 141

Conclusion

- **Le domaine est très vaste**
 - Vision partielle
 - Qui doit vous permettre d'aborder les autres problèmes
- **Pour chaque problème : plusieurs méthodes**
 - Qui ne sont pas équivalentes
 - Qui peuvent dépendre du contexte
 - A choisir avec attention
- **Un ordinateur**
 - Extraordinaire outil
 - Un outil limité
 - Contrôler les précisions
 - Maîtriser les itérations
 - Contrôler la complexité
 - Contrôler a posteriori les résultats

Il est possible d'obtenir de très bons résultats avec un peu de soin

ESIL, Algorithmes numériques 142

Bibliographie succincte

1. **Numerical Recipes in C++: The Art of Scientific Computing**, W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, Cambridge University Press, 2002
2. **Numerical Recipes in C: The Art of Scientific Computing**, W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, Cambridge University Press
3. **Numerical Recipes in Fortran: The Art of Scientific Computing**, W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, Cambridge University Press
4. **Méthodes de Calcul Numérique**, J.P. Nougier, Masson, 1983 (1e édition) (épuisé ?)
5. **Introduction à l'analyse numérique matricielle et à l'optimisation**, P. G. Ciarlet, Dunod, 1990 (1e édition)
6. **Algorithmique Numérique**, C. Brezinski, Ellipses, 1988
7. **Qualité des calculs sur ordinateur, vers des arithmétiques plus fiables**, M. Daumas et J.-M. Muller éditeurs, Masson, 1997
8. **Analyse Numérique pour Ingénieurs**, A. Fortin, Presses Internationales Polytechnique, 2001
9. **Introduction aux méthodes numériques**, F. Jędrzejewski, Springer, 2 édition, 2006