Algorithmes numériques

Marc DANIEL

Ecole Polytech Marseille,
Campus de Luminy, case 925, 13288 Marseille cedex 9
Marc.Daniel@univ-amu.fr

Février 2021

•1

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**

•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**
  - Méthode du QU

Quatrième partie : Différentiation et Intégration numériques

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

•3

Plan (suite)

Cinquième partie : Interpolation - Approximation - Lissage : notions

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

Conclusion

Bibliographie succincte

•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**

•5

Organisation du cours

- **2 x5 h de cours**
  - **TRES TRES court**
- **6h de TD en salle (en 2 groupes : Q1 avec Q3 et Q2 avec Q4)**
  - Préparation du travail sur ordinateur
  - 4 voire 5 feuilles de TD
  - Travail sur les algorithmes
  - 1 séance avant chacune des 3 premières feuilles
  - Illustration du cours
  - Programmer les méthodes de base

•6

Organisation du cours

- **14h de TD sur machine (en 4 groupes) : travail en binôme**
  - **Les TD sont organisés en feuilles**
  - **Les TD ne seront pas notés**
    - **Base du volontariat**
    - **Les enseignants corrigeront ce que les étudiants souhaite envoyer**
      - **Code, résultats, analyse**
  - **La présence (réelle) sera notée**
  - **Tout le matériel est sur ma page WEB**
- **Un examen général sur le cours avec des questions sur les TD**

•7

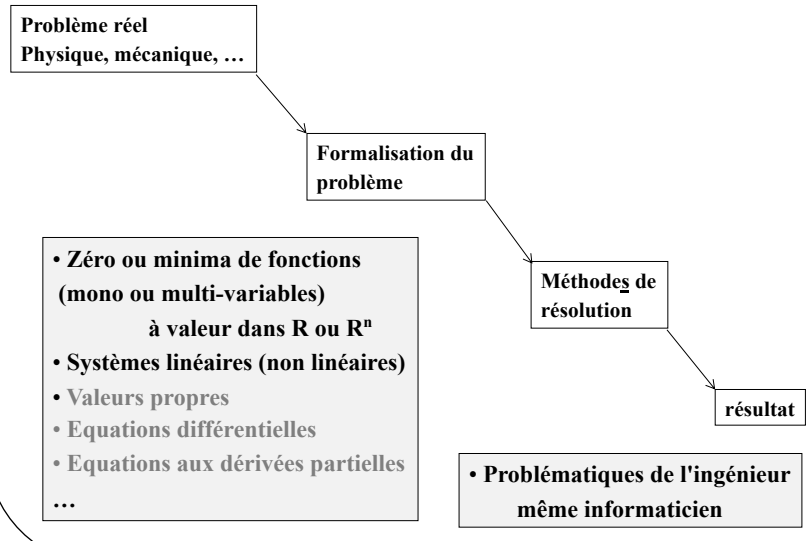
Première partie : Les bases de l'algorithmique numérique

•8

- **Algorithmique numérique ou Analyse Numérique ?**
- **2 approches <u>complémentaires</u> 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
- **<u>L'approche algorithmique n'exclut pas</u>**
  - De savoir calculer avec un ordinateur
  - De savoir analyser et maîtriser les calculs
  - De connaître les principaux résultats théoriques

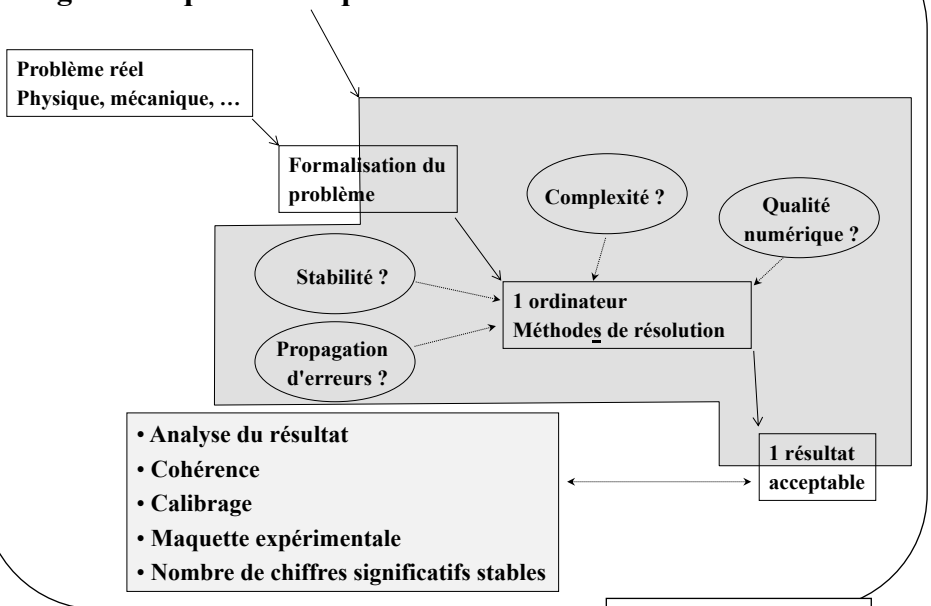
•9

Schéma général



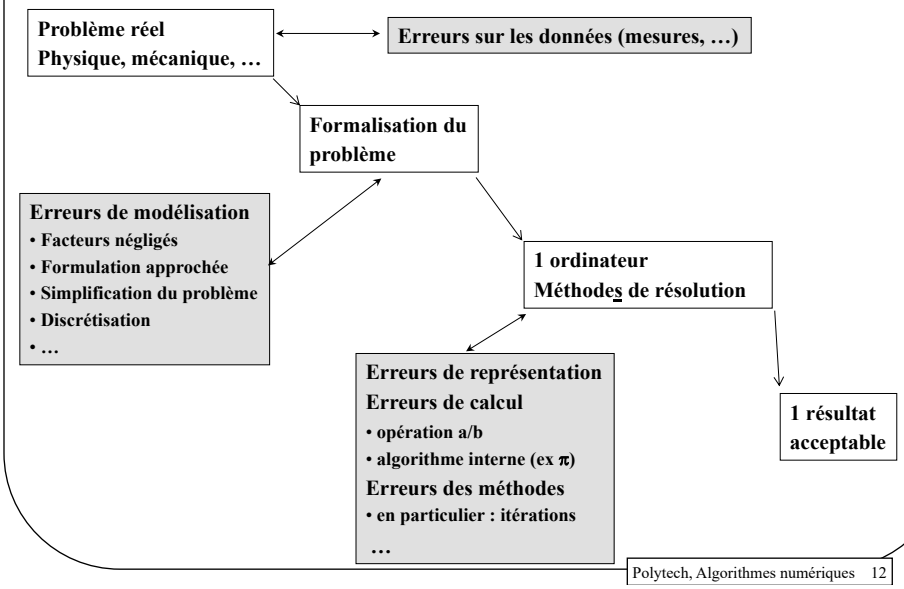
•10

Algorithmique numérique



•11

Algorithmique numérique : les erreurs



•12

Algorithmique numérique : les erreurs

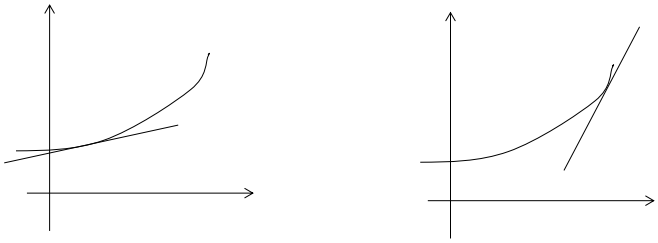
- **Les erreurs sont très nombreuses**
- **Elles peuvent se compenser**
- **Elles ont plutôt tendance à se cumuler**
- **Les erreurs introduites par l'ordinateur doivent être**
  - Maîtrisées,
  - Analysées,
  - Compatibles avec les erreurs sur les données et de modélisation,
  - Compatibles entre elles,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

•13

- **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

•14

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, …**

•15

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)

$$Nombre = -a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + a_{n-3}2^{n-3} + ... + a_1 2^1 + a_{n-1}2^0 \quad a_i \in \{0,1\}, i = 0,...,n-1$$

  - *Nombre* > 0 : $a_{n-1} = 0$ ($2^{n-1} > 1+2+2^2+\ldots+2^{n-2}$)
  - Plus grand entier positif : $2^{n-1}-1$ ($2^{15}-1=32767$, $2^{31}-1$,...)
  - Plus petit entier négatif : -2^{n-1} ($-2^{15}=-32768$, -2^{31},...)
  - *Nombre* = neg <0 : on code le nombre sous la forme $-2^{n-1}+\underbrace{(2^{n-1}+neg)}_{=pos>0}$
    - Ce qui revient à inverser tous les bits et ajouter 1

•16

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

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
- $m = 0,d_1d_2d_3\ldots d_n \Leftrightarrow m = d_1.b^{-1} + d_2.b^{-2} + d_3.b^{-3} + \ldots + d_n.b^{-n}$
  - 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 <u>erreur relative</u>**
    - $x = \pm\ 0,d_1d_2d_3\ldots d_n\ .\ b^l$

•19

Les « réels » et leurs limites

- **La plupart des réels sont approchés dès leur saisie**
  - Influe sur les calculs (erreur de représentation)
  - **<u>La norme IEEE-754-2008</u>** <u>(version actuelle)</u>
    - <u>Portabilité des programmes</u>
    - Les mêmes résultats !
    - Une mise à jour en juillet 2019
      - « plutôt confidentielle »
      - Va évidemment tenir compte des architectures 64 bits
      - Ne changera pas la difficulté inhérente à la représentation virgule flottante
      - Diminuera les problèmes numériques

•20

- **La norme IEEE-754**
  - Représentation normalisée en base 2 (d_1=1, donc non conservé)
  - Des règles d'arrondi et de normalisation
  - Flottant SP

	8 bits	23 bits
±	l	m

  - Flottant DP

	11 bits	23+ 29 = 52 bits	
±	l	m	m

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

•21

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+δ = x (voir TP) (δ < 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**
  - $|x|\leq\varepsilon$
  - $|x-x'|\leq\varepsilon$
  - ε est au mieux de l'ordre de err (très optimiste) $(10^{-7}, 10^{-15})$

•22

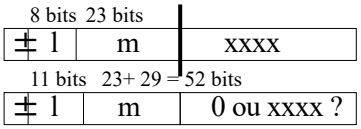
Les « réels » et leurs limites

- **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
- **Les processeurs 64 bits !!!**

•23

Les « réels » et leurs limites

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



  - 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 (double pour des doubles)
  - Un seul élément en SP (ou adapté à la SP) ⇒ problème garanti
    - Sauf si la DP ne servait à rien !

•24

Les nombres

- **Cas des valeurs petites**
  - a *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 !

•25

Les opérations

En entier

- **Tous les calculs sont exacts tant 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 !

•26

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≠0
  - Résultat arrondi
- **Multiplication**
  - Le résultat dans le registre est bon : comme à la main !
    - Bonne taille des registres
  - Résultat est arrondi

•27

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**

•28

La norme définit 4 arrondis

- Vers -∞ : $\nabla(x)$ plus petit nombre machine ≤ x
- Vers +∞ : $\Delta(x)$ plus petit nombre machine ≥ x
- Vers 0 : = $\nabla(x)$ si x ≥ 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**
- **Le choix de l'arrondi est fixé dans la norme pour les opérations usuelles**
  - Pas pour toutes les fonction mathématiques …

•29

- **Soit $f(a,b)=333{,}75b^6+a^2(11a^2b^2-b^6-121b^4-2)+5{,}5b^8+a/2b$**
  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 10^{29} en SP
  - f(a,b) = -1.180592… 10^{21} en DP
- **Sur la machine de votre serviteur (avec ou sans factorisations)**
  - -9,875012 10^{29} en SP ou -2.341194 10^{29} en SP
  - -1.180592… 10^{21} en DP ou 1.180592… 10^{21} en DP
  - 1.787028…10^{20} en long double ou 5.764608... 10^{17} en long double
    Avec la fonction pow qui est double !
  - ou -1.57142210^{29} en SP
  - ou 2.513544… 10^{21} en DP
  - ou 1.756112... 10^{17} en long double

Résultat théorique : -0,827396……

Il faut donc toujours avoir un regard critique

Avec Matlab = -1,1806 10^{21}

•30

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 = x*~~ **n'a pas de sens**
- **Choix d'une précision ε**
  - $|x-x^*| < \varepsilon$: erreur absolue
  - $\frac{|x-x^*|}{|x|} < \varepsilon$: erreur relative

•31

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|>>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|≈1
  - Il existe des applications nécessitant une précision absolue

•32

- **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}**
  - $\Delta x \approx 10^{-n}$
  - $x = 0.dddddddddd\ 10^{\alpha}$

  3 cas
  - $\alpha=0$ $x = 0.dddddddddd$ nbchif =
  - $\alpha<0$ $x = 0.0000dddddddd$ nbchif =
  - $\alpha>0$ $x = ddd.dddddd$ nbchif =
- **Le nombre de chiffres dépend de α …**

•33

- **Un retour arrière : précision relative de l'ordre de 10^{-n}**
  - $|\Delta x|/|x| \approx 10^{-n}$
  - $x = 0.dddddddddd\ 10^{\alpha}$, 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 =

•34

- **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**
  - $|\Delta x| < \varepsilon_{relative} |x| + \varepsilon_{absolue}$
  - 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)

•35

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^*)

•36

Arithmétique d'intervalle

- **Les limites**
  - Le temps de calcul
  - La taille des intervalles des variables qui croit très vite
  - Problème quand 0 est dans l'intervalle
  - Surestimation des intervalles
    - Ex : $P(x) = x\text{-}x^2$
    - $P([0,1])= [0,1]- [0,1]^2 = [0,1]-[0,1] = [-1,1]$
    - 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

•37

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

•38

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

•39

Précision infinie

- **Les nombres sont décomposés en blocs de chiffres successifs.**
  - X1 = xxxx.xxxx yyyyyyyyy zzzzzzz
  - X2 = uuuu.uuuu vvvvvvvvv 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 **G**NU **M**ultiple **P**recision Arithmetic Library

•40

Le calcul symbolique

- **Transformation des expressions mathématiques**
  - Evaluation finale
- **Des outils intéressants de calcul symbolique**
  - Mathematica, Mapple, …
- **<u>Les limites</u>**
  - Le temps de calcul
  - Les capacités de résolution
  - L'intérêt réel sur des données imprécises

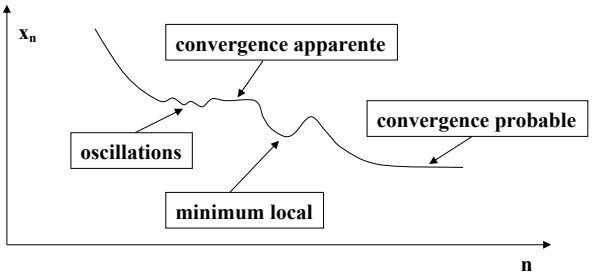
•41

- **Les problèmes itératifs**
  - De nombreux processus sont itératifs
  - D'un point de vue théorique : $\lim_{n\to\infty} x_n = l$
    - 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 : $u_n = n$)

•42

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**



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 !

Bon algorithme ? (1/2)

- **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 ...

•45

Bon algorithme ? (2/2)

- **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 ?)
- **Tout ce qui est vrai pour l'algorithmique est vrai pour l'algorithmique numérique**
  - Les problèmes numériques en plus !

•46

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

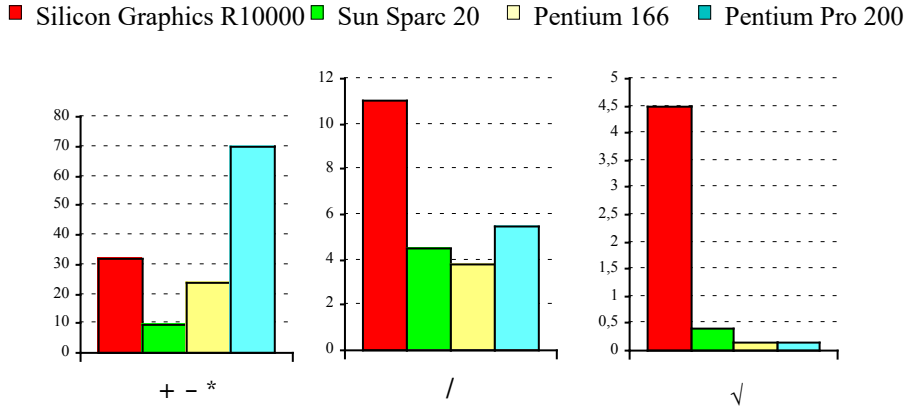
•47

- **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
- **Vrai apport du parallélisme, des GPU, ….**
  - Tout prendre en compte
  - Repenser les algorithmes

•48

Considérations algorithmiques

Nombre de Megaflops pour effectuer des opérations élémentaires

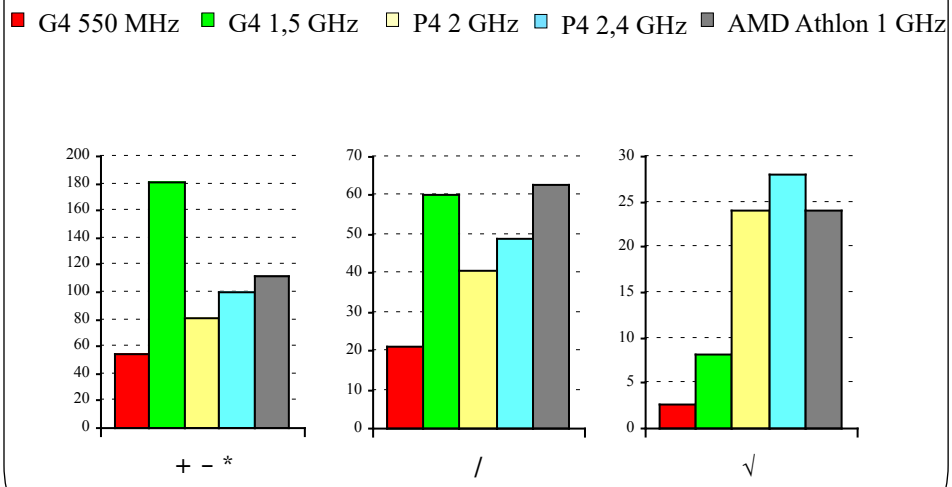


Attention aux techniques de mesure
* pas d'optimisation possible
* mesurer et déduire les temps des opérations « accessoires »

•49

Considérations algorithmiques

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



Données fiables en faisant les tests sur 10^9 opérations
Pas de résultats stables sur le G5

•50

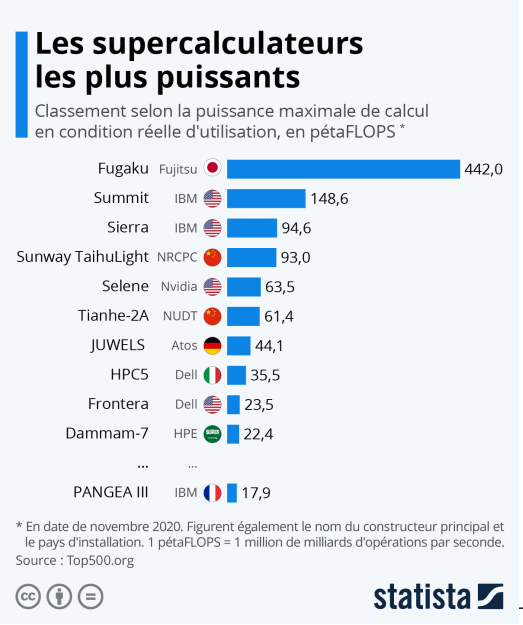
Considérations algorithmiques

- **Avoir une idée des temps de calcul**
  - En 1980, une école d'ingénieurs avec une machine d'1Mflops était une école bien dotée
  - On peut considérer un PC actuel de 1000 à 2000 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 !) (1,76 10^{15} flops)
  - Thianhe-2 (Chine) 33,86 Pétaflops fin 2013 (33,86 10^{15} flops)
  - Sunway TaihuLight (Chine) 100 Pétaflops (10^{17} flops)
  - 10 Pétaflops installés par le CNRS à Saclay en 2019
- **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

•51

Considérations algorithmiques

- **Avoir une idée des temps de calcul : données novembre 2020**



7, 3 millions de cœurs 64 bits …

•52

Ce qui est attendu dans votre travail (en C)

0) La présentation doit être soignée : indentation rigoureuse et constante, beaucoup de commentaires (utiles).

1) dès qu'un programme est un peu conséquent et intéressant, il y a un main et une fonction appelée. Le main est le programme de test.

2) cette fonction ne communique avec le main que par sa liste de paramètres et le code de retour.

3) aucune fonction inutile avec quelques lignes qui se lisent plus mal dans la fonction inutile que mises dans la fonction appelante.

4) toute fonction a un entête indiquant le rôle de cette fonction, le dictionnaire des variables, le nom des auteurs, la date de création et la date de dernière modification. En toute rigueur, il faut indiquer aussi l'entreprise.

•53

5) aucune variable sensible ne doit être fixée dans la fonction (en particulier la précision sous forme d'un nombre de chiffres attendu, et le nombre maximum d'itérations).

6) assez souvent le code de retour d'une fonction permet d'indiquer ce qui s'est passé dans la fonction. Dans les bibliothèques scientifiques, la règle est souvent une valeur positive : information de succès comme par exemple le nombre d'itérations effectuées, une valeur négative : indique un problème (1 valeur par problème).

7) Initialisation des variables là où cela est important, que cela a un sens et pas au niveau de la déclaration (en particulier juste devant les structures itératives).

8) 1 seul return par fonction, aucune sortie violente d'une structure itérative.

9) à la fin d'une structure itérative avec plusieurs conditions, il faut un test pour savoir sur quelle condition a eu lieu la sortie.

10) Un programme doit être testé. Il faut avoir analysé tous les cas possibles et avoir un jeu de test pour chaque cas possible, montrant ainsi qu'il est bien traité.

•54

Deuxième partie : Equations non linéaires

•55

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 R^q, $x \in R^p$**
  - On considérera surtout p=q=1
- **Liens avec l'optimisation non linéaire : min f(x), $x \in D$**
  - Un extremum de f est caractérisé par :
    - f'(x)=0 (p=1)
      - Minimum : f''(x) >0
      - Maximum : f ''(x)<0
    - grad f = 0 (p>1), si f est à valeur dans 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**

•56

- **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

•57

- **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 ($f(r) = f'(r) = \ldots. = f^{(p)}(r) = 0$
  - Le cas insoluble de l'infinité de racines
    - Cas très important en CAO
    - Cas limite : racines multiples

•58

Evaluation d'un polynôme

- **Dans quelle base est exprimée le polynôme**
  - Base canonique (power basis)
  - Autre base, Bernstein en particulier $B_{i\,n}(t) = C_i^n t^i (1-t)^{n-i} \quad i = 0,...,n$
  - Attention, changement de base mal conditionné
- **$P(x)=a_0\,x^n + a_1\,x^{n-1} + \ldots\, a_{n-1}\,x + a_n$**
  - Evaluation « directe » <u>à proscrire</u>
    - Stabilité
    - Nombre d'opérations
- **<u>Il faut utiliser</u> 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

•59

Evaluation d'un polynôme

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

•60

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

$b = a_i + \alpha\, b$

- **Exemple : $P(x)= 2x^3 + 3x^2 - x + 1$**

```
mdaniel$ ./horner
le polynome est a[0]=2.000000,a[1]=3.000000,a[2]=-
1.000000,a[3]=1.000000

alpha= 2.000000

b= 2.000000
b= 7.000000
b= 13.000000
b= 27.000000
```


•61

Evaluation d'un polynôme

- **Dérivées successives :**

$P_n(x) = (x-\alpha)\, Q_{n-1}(x) + A_n$

$Q_{n-1}(x) = (x-\alpha)\, Q_{n-2}(x) + A_{n-1}$

....

$Q_1(x) = (x-\alpha)\, Q_0(x) + A_1$

$Q_0(x) = A_0$

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

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

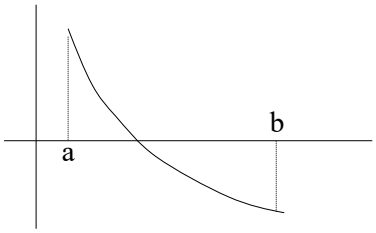
$P_n(x)=P_n(\alpha)+(x-\alpha)P_n'(\alpha)+(x-\alpha)^2 P_n''(\alpha)/2!+ \ldots +(x-\alpha)^n P_n^{(n)}(\alpha)/n! +0$

•62

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 ∈ [a,b]   */
```



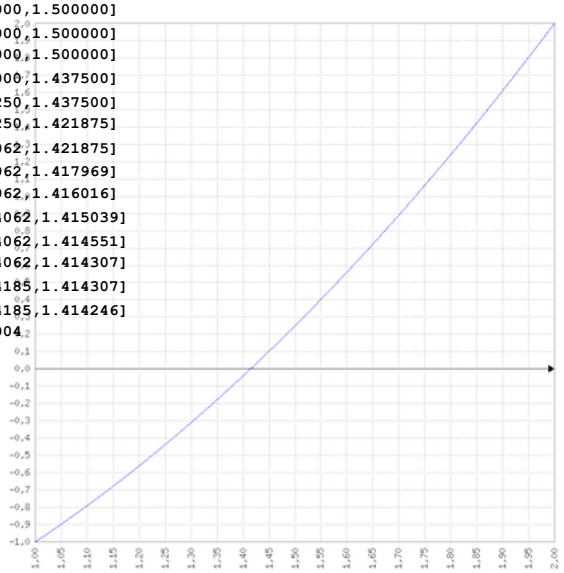
  - Et si la fonction n'est pas monotone ?

•63

Dichotomie

- **0 de x^2-2 sur [1,2]**

```
donnez a et b
1 2
intervalle initial = [1.000000,2.000000]
1, intervalle de recherche = [1.000000,1.500000]
2, intervalle de recherche = [1.250000,1.500000]
3, intervalle de recherche = [1.375000,1.500000]
4, intervalle de recherche = [1.375000,1.437500]
5, intervalle de recherche = [1.406250,1.437500]
6, intervalle de recherche = [1.406250,1.421875]
7, intervalle de recherche = [1.414062,1.421875]
8, intervalle de recherche = [1.414062,1.417969]
9, intervalle de recherche = [1.414062,1.416016]
10, intervalle de recherche = [1.414062,1.415039]
11, intervalle de recherche = [1.414062,1.414551]
12, intervalle de recherche = [1.414062,1.414307]
13, intervalle de recherche = [1.414185,1.414307]
14, intervalle de recherche = [1.414185,1.414246]
f au milieu de l'intervalle = 0.000004
```



•64

Dichotomie

- **Et si |f(c)|<eps ?**
- **Et si f(c) =0 ?**
  - Cas peu probable qui statistiquement se rencontre toujours sur les tests
  - Avec < ->[$c^{(1)}$,b]->[$c^{(2)}$,b]->[$c^{(3)}$,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 ∈ [a,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 ∈ [a,b] */

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

•65

Dichotomie

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 ∈ [a,b] */

- **l=b-a**
  - Itération 1 : b-a=
  - Itération n : b-a=
- **<u>On a un encadrement de la racine</u>**
  - si α^* est la vraie valeur
  - $\forall \alpha \in]a,b[\ | \alpha - \alpha^*| \leq$

  Pour avoir une erreur de l'ordre de ε, il suffit de prendre n ≥
- **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

•66

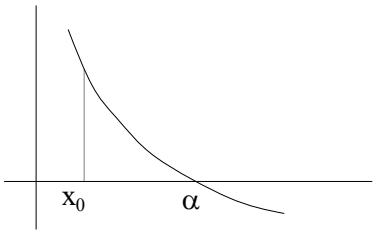
La méthode de Newton

- **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 : $k \geq 1 \quad x_k = x_{k-1} - \frac{f(x_{k-1})}{f'(x_{k-1})}$

•67

La méthode de Newton

- **Interprétation géométrique**
  - Équation de la tangente en x0
  - La tangente coupe l'axe des x en



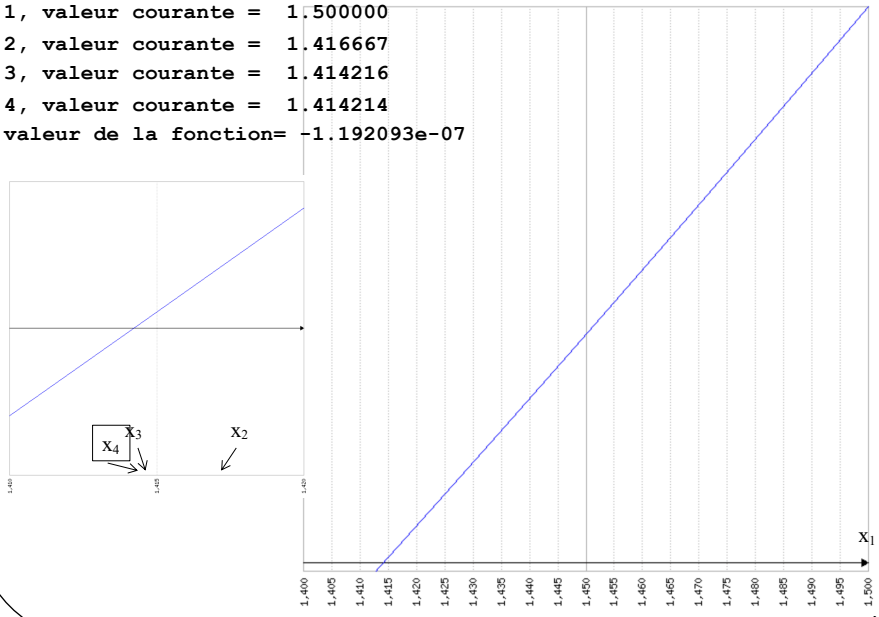
  - Convergence rapide (quand elle converge)
- **La méthode de Newton impose d'avoir accès à la dérivée**
- **Une dérivée approchée conduira tout de même à un résultat**
  - À l'extrême, n'importe quelle droite va fonctionner
  - Attention à l'«annulation » de la dérivée
    - Que se passe-t-il ?
- **Mieux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

•68

Dichotomie

- 0 de x^2-2 point de départ 2

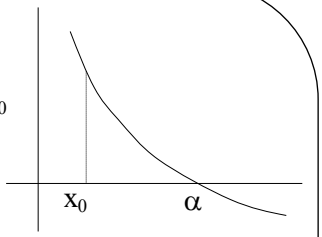
```
1, valeur courante =  1.500000
2, valeur courante =  1.416667
3, valeur courante =  1.414216
4, valeur courante =  1.414214
valeur de la fonction= -1.192093e-07
```



•69

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

•70

La méthode de Newton

- **L'application la plus connue : racine pième d'un nombre a**
  - $f(x)=x^p-a = 0$
  - $f'(x)=p.x^{p-1}$
  - $x_{k+1} = x_k - \frac{x_k^P - a}{p.x_k^{p-1}}$
  - $x_{k+1} = \frac{1}{p}((p-1)x_k + \frac{a}{x_k^{p-1}})$

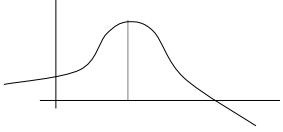
- **Si p=2**
  - $x_{k+1} = \frac{1}{2}(x_k + \frac{a}{x_k})$
  - 1 des algorithmes cablés sur les processeurs
    - + un algorithme pour trouver un point de départ convenable

•71

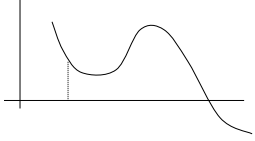
La méthode de Newton

- **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

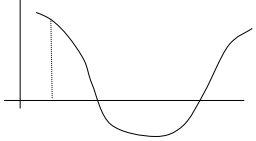
Point de départ



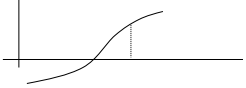
Minimum local



Far away!



Inflexion



Fin du tour gratuit !!!!

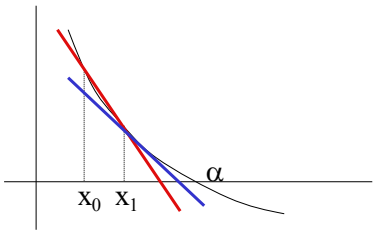
- **Il faut faire avec :**
  - **Contrôler les itérés successifs**
  - **Contrôler le nombre d'itérations**
  - **Contrôler les dérivées**
  - **…**

•72

La méthode de Newton

- **Cas des racines multiples d'ordre p**
  - $f(\alpha) = f'(\alpha) = \ldots = f^{(p-1)}(\alpha)$, $f^{(p)}(\alpha) \neq 0$
  - 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
  - $f'(x_k) \approx \dfrac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$

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



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
- **<u>Attention avec Newton, l'encadrement de la racine ne conduit pas forcément à l'obtention de la racine</u>**
- **Cas du polynôme : diviser à chaque fois P(x) par (x-α)**
  - 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 …

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
  - $P_0(x)=P(x)$ $P_1(x)=P'(x)$
  - $j \geq 2$ $P_j : P_{j-2}(x)=P_{j-1}(x).\ Q_{j-1}(x)-P_j(x)$ $(P_0=P_1.Q_1-P_2)$
  - On arrête la suite quand P_m est constant (au pire, m=n)
  - $S(x) = (P_0,P_1,\ldots,P_m)$ 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)

•75

- **Polynômes à coefficients réels: la méthode de Bairstow**
  - Recherche des racines sur C (corps algébriquement clos)
  - $P_n(x)=(x^2+p.x+q)Q_{n-2}(x)+R(p,q).x+S(p,q)$
    - 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 $x^2+p.x+q$
      - $\Delta > 0$: deux racines réelles
      - $\Delta = 0$: une racine double
      - $\Delta < 0$: deux racines complexes conjuguées
    - Reste les dernières racines du dernier polynôme Q

•76

Méthode de Bairstow

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

$a_0 = b_0$
$a_1 = b_1 + p\, b_0$
$a_2 = b_2 + p\, b_1 + q\, b_0$
…
$a_k = b_k + p\, b_{k-1} + q\, b_{k-2} \quad 2 \le k \le n-2$
…
$a_{n-1} = R + p\, b_{n-2} + q\, b_{n-3}$
$a_n = S + q\, b_{n-2}$

En posant, $b_{n-1} = R$ et $b_n = S - p\,R$
$b_0 = a_0$
$b_1 = a_1 - p\, b_0$
$b_2 = a_2 - p\, b_1 - q\, b_0$
…
$b_k = a_k - p\, b_{k-1} - q\, b_{k-2}$
…
$b_{n-1} = a_{n-1} - p\, b_{n-2} - q\, b_{n-3}$
$b_n = a_n - p\, b_{n-1} - q\, b_{n-2}$

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

•77

Méthode de Bairstow

- **Partant de p_0 et q_0 quelconques, on cherche Δp et Δq**
  - $R(p+\Delta p, q+\Delta q) = 0$ et $S(p+\Delta p, q+\Delta q) = 0$
  - Au premier ordre : $\begin{cases} R(p,q) + \dfrac{\partial R}{\partial p}\Delta p + \dfrac{\partial R}{\partial q}\Delta q = 0 \\ S(p,q) + \dfrac{\partial S}{\partial p}\Delta p + \dfrac{\partial S}{\partial q}\Delta q = 0 \end{cases}$
  - Système de Cramer 2X2 $\begin{cases} \Delta p = \dfrac{N_p}{Det} = \dfrac{S\dfrac{\partial R}{\partial q} - R\dfrac{\partial S}{\partial q}}{\dfrac{\partial R}{\partial p}\dfrac{\partial S}{\partial q} - \dfrac{\partial R}{\partial q}\dfrac{\partial S}{\partial p}} \\ \Delta q = \dfrac{N_q}{Det} = \dfrac{R\dfrac{\partial S}{\partial p} - S\dfrac{\partial R}{\partial p}}{\dfrac{\partial R}{\partial p}\dfrac{\partial S}{\partial q} - \dfrac{\partial R}{\partial q}\dfrac{\partial S}{\partial p}} \end{cases}$

•78

Méthode de Bairstow

- **Reste à estimer les dérivées partielles**

$\partial b_0/\partial p = 0$
$\partial b_1/\partial p = - b_0 - p\ \partial b_0/\partial p$
$\partial b_2/\partial p = - b_1 - p\ \partial b_1/\partial p - q\ \partial b_0/\partial p$
…
$\partial b_k/\partial p = - b_{k-1} - p\ \partial b_{k-1}/\partial p - q\ \partial b_{k-2}/\partial p$
…
$\partial b_{n-1}/\partial p = - b_{n-2} - p\ \partial b_{n-2}/\partial p - q\ \partial b_{n-3}/\partial p$
$\partial b_n/\partial p = - b_{n-1} - p\ \partial b_{n-1}/\partial p - q\ \partial b_{n-2}/\partial p$

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

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

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

•Et pour alléger les notations finales :

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

•79

Méthode de Bairstow

- **Reste à estimer les dérivées partielles**

$\partial b_0/\partial q = 0$
$\partial b_1/\partial q = 0$
$\partial b_2/\partial q = - b_0 - p\ \partial b_1/\partial q - q\ \partial b_0/\partial q$
…
$\partial b_k/\partial q = - b_{k-2} - p\ \partial b_{k-1}/\partial q - q\ \partial b_{k-2}/\partial q$
…
$\partial b_{n-1}/\partial q = - b_{n-3} - p\ \partial b_{n-2}/\partial q - q\ \partial b_{n-3}/\partial q$
$\partial b_n/\partial q = - b_{n-2} - p\ \partial b_{n-1}/\partial q - q\ \partial b_{n-2}/\partial q$

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

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

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

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

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

•80

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) $\Rightarrow$

$$\begin{cases} \Delta p = \dfrac{N_p}{Det} = \dfrac{S\dfrac{\partial R}{\partial q} - R\dfrac{\partial S}{\partial q}}{\dfrac{\partial R}{\partial p}\dfrac{\partial S}{\partial q} - \dfrac{\partial R}{\partial q}\dfrac{\partial S}{\partial p}} \\ \Delta q = \dfrac{N_q}{Det} = \dfrac{R\dfrac{\partial S}{\partial p} - S\dfrac{\partial R}{\partial p}}{\dfrac{\partial R}{\partial p}\dfrac{\partial S}{\partial q} - \dfrac{\partial R}{\partial q}\dfrac{\partial S}{\partial p}} \end{cases}$$

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

$$\begin{cases} Det = c_{n-2}^2 - c_{n-1}c_{n-3} \\ N_p = b_{n-1}c_{n-2} - b_n c_{n-3} \\ N_q = b_n c_{n-2} - b_{n-1}c_{n-1} \end{cases}$$

•81

Méthode de Bairstow

Algorithme

- On part d'un polynôme courant P_{cour} = Polynôme initial
- A chaque fois que l'on trouve 2 racines,
  le degré de P_{cour} diminue de 2 ($P_{cour}=Q_{n-2}$)

- **<u>Tests d'arrêt</u>**
  - 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)

•82

Méthode de Bairstow

Algorithme SIMPLIFIE

P_{cour} =Polynôme initial ; deg_P_{cour} = n ; processus convergent est vrai

Tantque ((deg_P_{cour}>2) et (processus convergent)) faire

  p=p0 ; q=q0

  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 (nombre max d'itérations atteint) alors

    processus convergent = faux

  Sinon

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

    mettre à jour les b_i (avec les nouveaux p et q) (i=0, …,deg_P_{cour}-2)

    $P_{cour}=Q_{n-2}$ (les b_i) ; deg_P_{cour}= deg_P_{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

•83

Méthode de Bairstow

Quelques éléments sur un exemple

- **$P(x)= (x-1)(x-1)((x-2)= x^3 - 4x^2 + 5x – 2$**

- **Pour p et q on doit trouver**
- **Soit**
  - p=-2, q=1 (x-1)(x-1) d 'où 1 racine double
  - Une seule étape du grand tant que, et le polynôme restant est x-2
- **Soit**
  - p=-3, q=2 (x-1)(x-2) d'où 1 et 2 racines
  - Une seule étape du grand tant que, et le polynôme restant est x-1

•84

Troisième partie : Systèmes d'équations linéaires

•85

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 :

•86

- **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**

•87

- **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**

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

•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 $\Rightarrow Cond_2(M') = Cond_2(A)$
    - 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 :** $$x_k = \frac{1}{a_{kk}}\left(b_k - \sum_{j=k+1}^{n} a_{kj}.x_j\right)$$
- **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 = λ det A
- **Remplacer une ligne i par ligne i + λ ligne j**
  - $M(l_i \leftrightarrow l_i+\lambda l_j)$=I+ M* avec $M^*_{lm}= \lambda \delta_{li}\delta_{mj}$
  - $M(l_i \leftrightarrow l_i+\lambda l_j)^{-1}= M(l_i \leftrightarrow l_i-\lambda l_j)$
  - 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 perm(i) =i
  - Echange des colonnes i et j :
    - échanger les contenus de perm(i) et 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

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

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

•93

Méthode de Gauss

- **Description de l'étape k (k=1 à n (ou n-1))**
  - $a_{ij}=0$ $j<k$, $j<i$

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

  - 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

•94

Méthode de Gauss

- **Le terme a_{kk} est le pivot**

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

- **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
- **Statégie : pivot partiel : pivot t.q.** $|pivot| = \sup_{p=k,..n} |a_{pk}|$
  - Si |pivot| <ε ?
- **Statégie : pivot total : pivot t.q.** $|pivot| = \sup_{\substack{p=k,..n \\ l=k,..n}} |a_{pl}|$
  - Si |pivot| <ε ?
  - Modifie l'ordre des inconnues
  - Optimal, mais inversion de lignes et colonnes

•95

Méthode de Gauss

- **Evaluation (1/2)**

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

  - 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 :

•96

Méthode de Gauss

- **Evaluation (2/2)**

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

  - 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 cond(A)

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

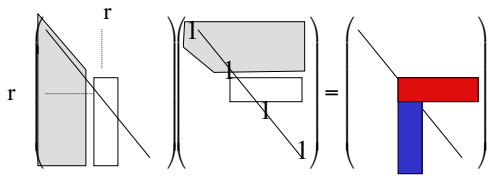
$$\begin{pmatrix} a_{11} & a_{12} & & a_{1n} \\ a_{21} & a_{22} & & \\ & & a_{jj} & \\ a_{n1} & & & a_{nn} \end{pmatrix} = \begin{pmatrix} l_{11} & 0 & 0 & 0 \\ & l_{22} & 0 & 0 \\ & & l_{jj} & 0 \\ l_{n1} & & & l_{nn} \end{pmatrix} \begin{pmatrix} 1 & u_{12} & & u_{1n} \\ 0 & 1 & u_{22} & \\ 0 & 0 &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

Méthode du LU

Construction de :
L par colonnes
U par lignes

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

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



- 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

•99

Méthode du LU

Evaluation de la décomposition

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

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

- **r=1,…, 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

•100

Méthode du LU-LR : remarque

<u>On suppose A inversible</u>

pas d'inversion de lignes ou colonnes

et les valeurs propres de A sont réelles et distinctes

- **$A = A_0 = L.U$ (ou $L_0.R_0$)**
  - → $R_0 = L_0^{-1}A_0$
- **$A_1 = R_0.L_0 = L_1.R_1$**
  - → $A_1 = L_0^{-1}A_0 L_0$
  - A_0 et A_1 sont semblables : elles ont les même valeurs propres
- **…**
- **$A_k = R_{k-1}.L_{k-1} = L_k.R_k$**
- $A_\infty = \lim_{k\to\infty} A_k$ **est une matrice triangulaire supérieure**
- **<u>Valeurs propres sur la diagonale</u>**

Méthode de Cholesky

- **S'applique sur les <u>matrices symétriques définies positives</u>**
  - 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 $L^TX=Y$

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

Méthode de Cholesky

Construction par colonnes

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

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

$$l_{kj} = \frac{1}{l_{jj}} \left(a_{kj} - \sum_{l=1}^{j-1} l_{kl} l_{jl} \right) \qquad k = j+1,\ldots,n$$ ($l_{jj} \neq 0$ 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 :
$n^3/6$ add et mul et $n^2/2$ div

•103

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 : $\lim_{k\to\infty} X^k = X$
  - 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, $\rho(M_{Jacobi})<1$ et $\rho(M_{gauss\text{-}Seidel})<1$
  - 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

•104

- **Jacobi**

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

  - soit $X_0 = (x_0, x_1, \ldots, x_n)^T$
    - Par exemple $X_0 = \left(\frac{b_1}{a_{11}} \quad \frac{b_2}{a_{22}} \quad ... \quad \frac{b_n}{a_{nn}} \right)^T$
  - 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

- **Gauss-Siedel**

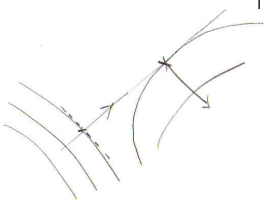
$$\begin{cases} a_{11}x_1 + a_{12}x_2 + ...+ a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + ...+ a_{2n}x_n = b_2 \\ ... \\ a_{n1}x_1 + a_{n2}x_2 + ...+ a_{nn}x_n = b_n \end{cases}$$

  - soit $X_0 = (x_0, x_1, \ldots, x_n)^T$
  - Simple amélioration de Jacobi :
    - Dès que x_j^{k+1} est calculé, pourquoi utiliser x_j^k ?

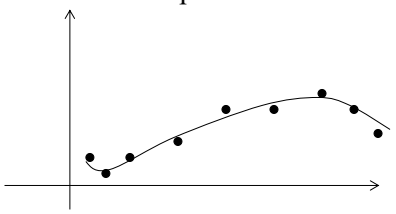
$$x_i^{k+1} = \frac{1}{a_{ii}}(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij}x_j^k)$$

  - Calcul très simple et rapide des itérés

- **Méthodes de descente :**
  - « a priori » sur des matrices symétriques définies positives
  - Font partie des méthodes d'optimisation vues en 2 année (REVA)
  - On minimise la forme quadratique : $f(X) = X^T A X - B^T X$ (f de R^n dans R)
    - On doit avoir grad(f)= 0
    - 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-conjugé, ...



- **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 $P(x_i)=y_i \quad i=1,\ldots,n$
  - (m+1) coefficients inconnus de P : $P(x)=a_m x^m+a_{m-1}x^{m-1}+\ldots. +a_0$
  - Soit : $M.X_a=Y$, 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 !)

- **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 :

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

  - Il s'agit de la minimisation d'un fonction F à (m+1) inconnues
  - e est atteint si : Grad F = 0
  - Si $\dfrac{\partial F}{\partial a_j} = 0 \quad j = 0,...,m$

•109

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

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

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

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

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

•110

Problèmes de moindres carrés

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

•111

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

nl

$$\begin{pmatrix} . & . & . & . \\ 0 & . & . & . \\ 0 & 0 & . & . \\ 0 & 0 & 0 & . \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

nc

nc

Q.A composée de :
- A_1(nc,nc)
- A_2(nl-nc,nc)

Transformation de Householder
Transformation de Givens

•112

- **Il faut résoudre**

$$\begin{matrix}\mathbf{A_1}\\ \\ \mathbf{A_2}\end{matrix}\begin{pmatrix}. & . & . & .\\ 0 & . & . & .\\ 0 & 0 & . & .\\ 0 & 0 & 0 & .\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\end{pmatrix}\begin{pmatrix}\\ \mathbf{X}\\ \\ \\ \\ \end{pmatrix}-\begin{pmatrix}\mathbf{B_1}\\ \\ \\ \\ \mathbf{B_2}\end{pmatrix}=\begin{pmatrix}\mathbf{R_1}\\ \\ \\ \\ \mathbf{R_2}\end{pmatrix}$$

  - Le vecteur R est appelé résidu
  - Il faut trouver le minimum de $||R||_2$
  - On a $R_1 = A_1.X - B_1$
    $R_2 = B_2$

- **Théorème de « Pythagore »**

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

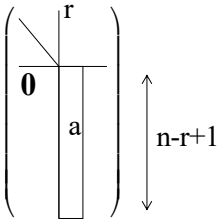
$$\begin{matrix}A_1\\ \\ A_2\end{matrix}\begin{pmatrix}. & . & . & .\\ 0 & . & . & .\\ 0 & 0 & . & .\\ 0 & 0 & 0 & .\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\end{pmatrix}\begin{pmatrix}\\ X\\ \\ \\ \\ \end{pmatrix}-\begin{pmatrix}B_1\\ \\ \\ \\ B_2\end{pmatrix}=\begin{pmatrix}R_1\\ \\ \\ \\ R_2\end{pmatrix}$$

  - 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 ⇔ $A^TA.X=A^TB$**
  - L'erreur commise est $\|R_2\|_2^2$
  - **La différence** : $cond_2(A^TA)=cond_2(A)^2$!!!!!!! (« rappel » ….)
  - ~~$A^TA.X=A^TB$~~ **Il faut résoudre A.X=B 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 $a^T=(a_1,a_2, \ldots, a_n)$, alors il existe une matrice H, t.q. $H.a = \alpha\, e1$ ($e_1^T=(1,0, \ldots, 0)$),**
  - 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 $\begin{cases} \alpha = -(signe(a_1))\|a\| \\ \eta = \alpha(\alpha - a_1) \\ \gamma = a - \alpha.e_1 \end{cases}$

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

$A^{(r)} =$ 

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, …, nc**
  - on peut diviser par η le plus grand possible, soit avec la norme de colonne la plus grande ⇒ inversion de colonne (vecteur perm !)
  - plus grande norme « = » 0 ⇒ rang non maximum

Méthode du QU - QR : remarque

On suppose A inversible

pas d'inversion de lignes ou colonnes

et les valeurs propres de A sont réelles et distinctes

- $\mathbf{A = A_0 = Q.R}$ **(ou** $\mathbf{Q_0.R_0}$**)**
  - → $R_0 = Q_0{}^tA_0$
- $\mathbf{A_1 = R_0.Q_0 = Q_1.R_1}$
  - → $A_1 = Q_0{}^tA_0\,Q_0$
  - A_0 et A_1 sont semblables : elles ont les même valeurs propres
- **…**
- $\mathbf{A_k = R_{k-1}.Q_{k-1} = Q_k.R_k}$
- $A_\infty = \lim_{k\to\infty} A_k$ **est une matrice triangulaire supérieure**
- **Valeurs propres sur la diagonale**

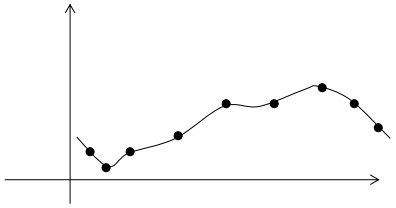
•117

Quatrième partie : Différentiation et Intégration numériques

•118

Approximation par un polynôme

- **Soit une fonction f connue en (n+1) points x_0, …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 $x=x_i$, $i=0,\ldots,n$
    - $E_n(x)$ fait intervenir $f^{(n+1)}(x)$

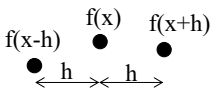


- **Si on maîtrise E_n, on peut remplacer f par P_n et intégrer ou dériver P_n**
- **On peut aussi considérer les développements de Taylor**

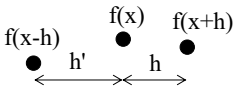
•119

Différentiation numérique

- **Soit une fonction f connue en (n+1) points x_0, …x_n**
  - Le problème est d'estimer un certain nombre de dérivées en x_0, …,
- **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
- **<u>Problème très instable</u>**
  - grosses approximations et différences de quantités voisines
  - Les données sont bruitées ⇒ grosses instabilités

•120

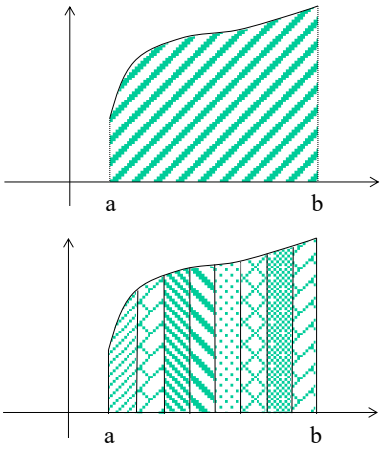
Différentiation numérique

- **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

•121

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
  - $x_0=a$, $x_1=a+h$, …, $x_n=a+n.h=b$
  - On calcule : $I = \sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} f(x)dx$



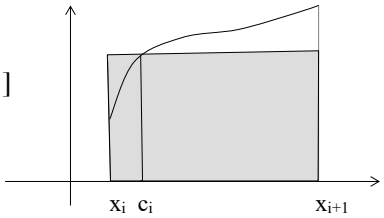
- **Plus le pas h est petit**
  - Plus la méthode est précise
  - Plus les calculs sont longs

•122

- **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

- **La méthode des rectangles :**
  - $\int_{x_i}^{x_{i+1}} f(x)dx = (x_{i+1} - x_i) f(c_i) \qquad c_i \in [x_i, x_{i+1}]$

$$\int_a^b f(x)dx = \sum_{i=0}^{n-1} h f(c_i) \qquad c_i \in [x_i, x_{i+1}]$$

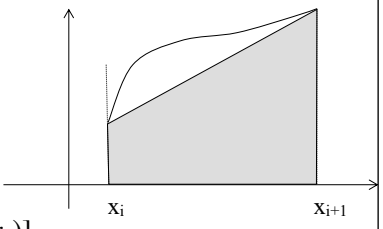


  - Si $c_i=x_i$ ou $c_i=x_{i+1}$
    - La méthode est d'ordre 0
  - Si c=(a+b)/2
    - La méthode est d'ordre 1
- **Ce n'est pas une méthode performante !**

- **La méthode des trapèzes :**
  - $\int_{x_i}^{x_{i+1}} f(x)dx = (\frac{x_{i+1}-x_i}{2})(f(x_i)+f(x_{i+1}))$

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

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

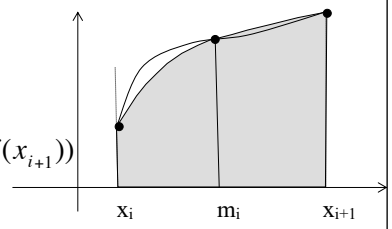


- **C'est la méthode de base minimale**
  - Méthode d'ordre 1
- **On peut montrer que l'erreur est en h^3**

•125

- **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

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



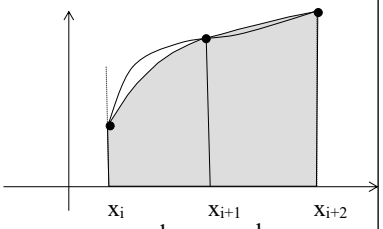
$\int_a^b f(x)dx = \sum_{i=0}^{n-1} \frac{h}{6}(f(x_i)+4f(m_i)+f(x_{i+1}))$

  - Que l'on écrit :

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

•126

- **La méthode de Simpson (suite)**
  - Deux notations possibles
  - $x_{i+1}-x_i=h$, $m_i \in [x_i, x_{i+1}]$
    - formules précédentes
  - $x_{i+2}-x_i=2h$, $m_i=x_{i+1}$
    - Intégration sur $[x_i, x_{i+2}]$
    - n intervalles, mais 2n+1 points



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

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

  - Que l'on écrit :

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

- **La méthode de Simpson une <u>bonne méthode</u>**
  - 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

- **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,…,2n-1 (base des polynômes de degré ≤ 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

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

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

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

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

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

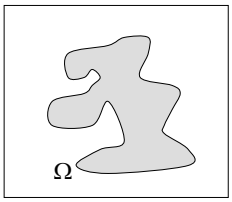
$$\int_{-1}^{1} f(x)dx = f(-\sqrt{\frac{1}{3}}) + f(\sqrt{\frac{1}{3}})$$

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

•131

- **Mêmes idées mais**
  - Beaucoup plus de calculs
  - Problème du domaine d'intégration

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

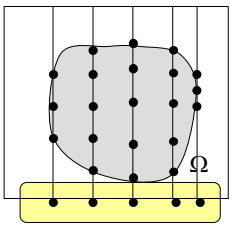


- **On essaie de se ramener à :**

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

  - Le cas idéal fréquent :

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



- **Méthodes classiques**
  - Quadrature de Gauss multivariables
  - Généralisation des formules précédentes

•132

- **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)*

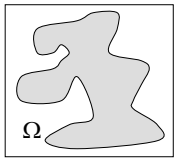
$$E(f(U)) = \lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} f(x_i)$$

- **En dimension supérieure (ici 2):**
  - On place Ω dans un rectangle *R*
  - On remplace *f* par *f**

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

  - On tire aléatoirement *n* valeurs sur *R*

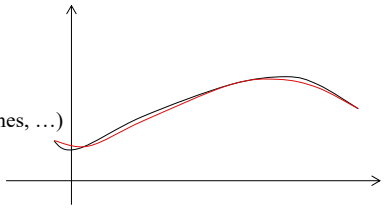
$$\iint_\Omega f(x,y)dxdy = aire(R).[\frac{1}{n}\sum_{i=1}^{n} f^*(x_i,y_i)]$$



Cinquième partie : Interpolation - Approximation - Lissage : notions

Problème

- **Soit une fonction f connue analytiquement ou en tout point**
  - Remplacer f par Φ* t.q. || f - Φ*||
    - 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

•135

Problème

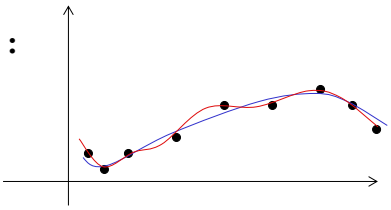
- **On se ramène toujours à un problème discret**

n points de l'espace considéré

On cherche à faire passer une courbe Φ* :

• par ces points (interpolation)

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



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

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

- **L'interpolation est un problème classique**
  - Plus contraint
  - Plus oscillant (données erronées)

•136

- **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**
  - $P(x_i)=y_i$ i=1,…,n
  - n coefficients inconnus de P : $P(x)=a_{n-1}x^{n-1}+a_{n-2}x^{n-2}+\ldots. +a_0$
  - Système linéaire (n,n) : connu pour être instable
    - conditionnement de la matrice élevé (type Van der Monde)
    - Possible pour des systèmes de degrés limités

•137

Dernière partie : Conclusion

•138

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

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. Jedrzejewski, Springer, 2 édition, 2006**