Calcul Formel et Numérique, INFO-F-205

Maarten Jansen

en collaboration avec G. Bontempi

Version février 2017

Renseignements pratiques, ressources

- Page web du cours
  `http://homepages.ulb.ac.be/~majansen/teaching/INFO-F-205/`
  contient :
  - Syllabus
  - Les transparents
  - Les horaires, les salles, etc.
  - Les fichiers matlab (du cours et des TP)
  - Les énoncés des projets
  - Infos sur l'examen
  - Les coordonnées des instructeurs
- Rendez-vous :
  - fixer par e-mail en spécifiant nom, sujet, durée prévue
  - au moins 2 jours ouvrables à l'avance
  - Pas de rendez-vous, ni des questions par e-mail après le début des examens (le 29 mai 2017)

Les ojectifs du cours

- Etudier les **méthodes numériques** (les algorithmes) de résolution de problèmes d'analyse mathématique et leur application à des problèmes réels.
  (Chap.3,4,5,6)
- Introduction à l'utilisation d'outils pour le calcul numérique sur ordinateur : **MATLAB** (le projet)
- Illustrer la **notion d'erreur** et les problèmes théoriques de **stabilité et convergence** des algorithmes.
  (spécifiquement visée dans le Chap.2, puis appliquée au fil des discussions)

L'examen

- **Répartition des points**
  1. L'examen (écrit) : 18/20
  2. Le projet : 2/20
- **Le projet**
  - Même s'il ne compte que pour 10% des notes, le projet est **obligatoire** ; c-a-d., sans raison acceptable, motivée, et prouvée, en cas d'absence, le projet compte pour 50% du résultat (et donc, le maximum est réduit à 10/20).
    Un travail à temps plein ne vous dispense pas du projet ! !
  - Durée : à peu près une semaine (entre l'annonce et la date limite)
  - La note compte pour les deux examens de l'année académique actuelle.
  - Un report de la note de l'année passée ne peut pas être accordé.
- **L'examen**
  - Ecrit
  - A livre ouvert

Chapitre 1 : Introduction

Maarten Jansen

Aperçu

Définition (Nick Trefethen, Oxford)

Le calcul numérique est une discipline qui traite de la définition, l'analyse et l'implémentation d'algorithmes pour la résolution numérique des problèmes mathématiques continus qui proviennent de la modélisation des phénomènes réels.

Une triple concordance entre

- Problème et modèle (le cadre)
- La solution du problème
- La résolution, la méthode, l'algorithme

A différents niveaux :

- Conception
- Analyse
- Implémentation, mise en œuvre

Au niveau de l'implémentation, exemple 1

Exemple au niveau de la mise en œuvre

Considérons la fonction $f(x) = -\log\left[\sin\left(\frac{1}{x^4} - \frac{1}{\sqrt{x^8+1}}\right)\right]$

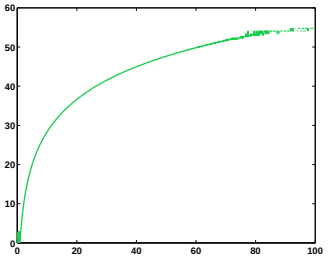
En matlab, nous avons

```
x = (1:1000)/10;
f = -log(sin(1./x.^4-1./sqrt(x.^8+1)));
plot(x,f,'linewidth',2,'color',[0.1 0.8 0.3])
set(gca,'fontsize',14,'fontweight','b')
set(gcf,'color', 'none')
set(gcf,'InvertHardCopy','off')
set(findobj(gcf,'color','white'),'color','none')
print -depsc fonctionbanale.eps
```

Réponse de matlab :

```
Warning: Imaginary parts of complex X and/or Y arguments ignored
```

Et la courbe de $f(x)$:



$$f(x) = -\log\left[\sin\left(\frac{1}{x^4}\left(1 - \frac{1}{\sqrt{1+1/x^8}}\right)\right)\right]$$

Série de Taylor : pour $\varepsilon \approx 0$:

$$f(\varepsilon) \approx f(0) + f'(0)\varepsilon + \frac{f''(0)}{2!}\varepsilon^2 + \cdots$$

- $f(\varepsilon) = \sqrt{1+\varepsilon} \approx 1 + \varepsilon/2$
- $f(\varepsilon) = 1/(1+\varepsilon/2) \approx (1-\varepsilon/2)$
- $f(\varepsilon) = \sin(\varepsilon/2) \approx \varepsilon/2$

Donc : $f(x) \approx -\log\left[\sin\left(\frac{1}{x^4}\cdot\frac{2}{x^8}\right)\right] \approx -\log\left[\frac{2}{x^{12}}\right] = 12\log(x) - \log(2)$

Un autre exemple

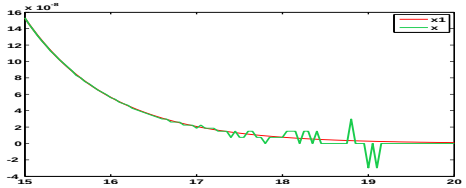
(toujours au niveau de la mise en œuvre)

Nous cherchons la plus grande racine de l'équation du second degré
$y(x) = 0$ où $y(x) = x^2 + 2e^t x - 1$ avec $t \in [15, 20]$

La formule algébrique classique nous amène à une expression théorique de x en fonction de t
$x(t) = -e^t \pm \sqrt{e^{2t} + 1},$
dont nous selectionnons la solution maximale, c.-à-d., celle-ci avec le plus

En matlab, nous construisons la courbe

```
t = 15+(0:100)/20;
x = -exp(t)+sqrt(exp(2*t)+1);
plot(t,x,'linewidth',2,...
    'color',[0.1 0.8 0.3])
```



— Problème numérique à cause d'une soustractions de deux grands nombres
— Implémentation alternative, basée sur le fait que $x_1 \cdot x_2 = c/a$ quand

x_1 et x_2 sont les solutions de $ax^2 + bx + c = 0$:

```
x2 = -exp(t)-sqrt(exp(2*t)+1);
x1 = -1./x2;
plot(t,x1,'linewidth',2,'color','r')
```

— Ou bien : la route de l'approximation

$$x(t) = -e^t + \sqrt{e^{2t}(1 + e^{-2t})} = e^t \cdot \left[-1 + \sqrt{1 + e^{-2t}}\right] \approx e^t \cdot \left[-1 + (1 + e^{-2t}/2\right] = e^{-t}/2$$

Les problèmes au niveau du "design"

Sur le plan de la conception

Analyse de **grands systèmes**, applications

1. Mécanique : conception d'une voiture
2. Analyse du temps, du climat : calculs météorologiques
3. Hydraulique : calculs des marées
4. Electronique

Ordinateurs = discrets, problèmes = continus (intégration, différentiation,...)

→ **Approcher**
— Introduction des erreurs (voir chapitre 2)
— Propagation des erreurs

Les outils classiques

1. Les matrices, les systèmes linéaires : voir chapitre 3
   Matrices souvent creuses
   Exemple : l'algorithme Page Rank (Google) (voir syllabus)
2. Les équation différentielles
   (a) ordinaires : voir chapitre 4
   Exemple : le problème du pendule (voir syllabus)
   (b) partielles : pas dans ce cours

- **Problèmes**
- Solutions
- Algorithmes

Typologie des problèmes

— Problèmes qualitatifs et quantitatifs
— Problèmes implicites et explicites
— Problèmes bien/mal posés
— Problèmes bien/mal conditionnés

Problèmes qualitatifs et quantitatifs

Qualitatifs : comportement des solutions : analyse de stabilité, comportement asymptotique : pour conclusions globales, à long terme.

Quantitatifs : calcul détaillé d'une solution (numérique) : conclusions spécifiques

Problèmes implicites et explicites

Problèmes sous forme explicite

- Exemple : trouver la racine carrée x d'un nombre d : $x = \sqrt{d}$
- Apparance générale : la solution x est une fonction des données d : $x = F(d)$ ou $\vec{x} = F(\vec{d})$

Problèmes sous forme implicite

- Exemple : trouver la plus grande racine x de l'équation algébrique du 2ième degré : $ax^2 + bx + c = 0$.
  Les données sont représentées par le vecteur $\vec{d} = [a\, b\, c]$
- Autre exemple : trouver la solution du système linéaire $A\vec{x} = \vec{b}$ où les données sont représentées par la matrice étendue $\vec{d} = [A|\vec{b}]$
- En général : $F(\vec{x}, \vec{d})$

Dans une certaine mesure, et dans certains cas, la distinction entre implicite et explicite est arbitraire
Exemple : $\vec{x} = A^{-1}\vec{b}$ prend la forme explicite, mais ce probème est identique (de point de vue théorique) au probème implicite d'un système linéaire. Pour la résolution, mieux vaut considérer un algorithm pour le problème implicite équivalent, plutôt que d'encoder la formule explicite littéralement.

Problèmes bien/mal posés

La notion de problème bien/mal posé a été introduite par Hadamard en 1923.

Bien posé

Le problème mathématique $x = F(d)$ est bien posé si la solution x

- existe,
- est unique,
- dépend continûment des données d.

Autrement le problème est dit mal posé.

Exemples des problèmes mal posés

— Le nombre de racines réelles d'un polynôme, en fonction d'un paramètre
— Une catégorie de problèmes souvent mal posés est le cas des problèmes inverses

Un **problème inverse** est un problème qui comprend l'inversion d'un opérateur, où l'application de l'opérateur est relativement simple, tandis que son inversion est compliquée.

Donc, les observations sont fonction des paramètres $\vec{d} = G(\vec{x})$, l'évaluation de $G(\vec{x})$, étant donné $\vec{x}$, est facile, mais nous sommes intéressés en $\vec{x} = G^{-1}(\vec{d})$.

La complication provient (souvent) du fait que plusieurs valeurs de $\vec{x}$ produisent (presque) le même $\vec{d}$.

Exemples des problèmes inverses

- trouver la position d'un obstacle à partir de l'information radar
- mesurer l'épicentre d'un phénomène sismique
- trouver une information sur une scène 3D à partir d'une image 2D (par exemple dans le procédé de la tomographie)
- estimer la température d'un objet à l'instant initial en sachant la température à l'instant final
- trouver la solution d'équations différentielles partielles
- **Déconvolution** des signaux, défloutage ("deblurring") des images

Déconvolution, défloutage

L'observation d'une image ou d'un signal est souvent sujette à des erreurs

- par cause du bruit, souvent un effet aléatoire additif
- par cause d'une distortion, un floutage : un effet linéaire ou non-linéaire

Un modèle linéaire pour le floutage est $d_i = \sum_{j=1}^{n} h_j x_{i-j} + \varepsilon_i$

Le composant ε_i est le bruit

La somme $\sum_{j=1}^{n} h_j x_{i-j}$ est dite **la convolution** du signal $\vec{x}$ avec le filtre $\vec{h}$

Notation : $\vec{d} = \vec{h} * \vec{x} + \vec{\varepsilon}$

Cette convolution si situe dans le cadre plus général $\vec{d} = H \cdot \vec{x} + \vec{\varepsilon}$ où $\vec{\varepsilon}$ représente le bruit.

Sources de complications

— La matrice H est souvent **creuse**, tandis que son inverse, si elle existe est typiquement une matrice **dense**. (Filtre local, inverse global)
— La matrice H est souvent **singulière**, donc pas inversible, c.-à-d., le noyau n'est pas trivial, et donc la solution du système n'est pas unique.
— La situation se complique quand la matrice H reste inconnue : le nombre d'inconnus (=paramètres = $\vec{x}$ et H) dépasse le nombre de données d : le système est (fortement) sous-déterminé.

Résolution : régularisation Vu que la solution n'est pas unique, il nous faut une contrainte additionnelle.

— Parmi les solutions, sélectionner celle-ci avec l'aspect creux maximal
— Sélectionner la solution avec une régularité (p.ex., qui minimise l'intégral $\int |f'(x)|dx$

Dans le reste du cours, nous ne considérerons que des problèmes bien posés.

Le conditionnement d'un problème

- Le conditionnement d'un problème bien posé de la forme $\vec{x} = F(\vec{d})$ mesure la sensibilité de la solution $\vec{x}$ du problème aux changements des données $\vec{d}$.
- Soient $\delta\vec{d}$ une perturbation admissible des données et $\delta\vec{x}$ la modification induite sur la solution du problème $\vec{x} = F(\vec{d})$.
- $\delta\vec{x} = F(\vec{d} + \delta\vec{d}) - F(\vec{d})$
- On appelle **conditionnement relatif** de ce problème la quantité

$$\kappa(\vec{d}) = \lim_{|D|\to 0} \sup_{\delta\vec{d}\in D} \frac{\|\delta\vec{x}\|/\|\vec{x}\|}{\|\delta\vec{d}\|/\|\vec{d}\|} = \lim_{|D|\to 0} \sup_{\delta\vec{d}\in D} \frac{\|F(\vec{d}+\delta\vec{d}) - F(\vec{d})\|/\|F(\vec{d})\|}{\|\delta\vec{d}\|/\|\vec{d}\|}$$

où $\sup$ dénote la borne supérieure,
D est un voisinage de l'origine, le symbole $\|\cdot\|$ désigne la norme vectorielle (ou matricielle).
- Quand $\vec{d} = \vec{0}$, le conditionnement relatif est forcement 0
- Quand $\vec{x} = \vec{0}$, le conditionnement relatif est forcement ∞

Le conditionnement absolu

Quand $\vec{d} = \vec{0}$ ou $\vec{x} = \vec{0}$, il est nécessaire d'introduire le **conditionnement absolu**

$$\kappa_{\text{abs}}(\vec{d}) = \lim_{|D|\to 0} \sup_{\delta\vec{d}\in D} \frac{\|\delta\vec{x}\|}{\|\delta\vec{d}\|}$$

Conditionnement d'un problème différentiable

Si le problème est en forme explicite $x = F(\vec{d})$, et si $F(\cdot)$ est **différentiable** en $\vec{d}$, alors, par définition, $\lim_{\delta\vec{d}\to\vec{0}} \frac{\delta x}{\delta\vec{d}}$ ne dépend pas de la trajectoire de $\delta\vec{d}$ vers $\vec{0}$.

Si nous notons par $F'(d) = \lim_{\delta\vec{d}\to\vec{0}} \frac{\delta x}{\delta\vec{d}}$ sa dérivée (le vecteur ou la matrice des dérivées partielles), le conditionnement relatif peut être écrit de la manière suivante $\kappa(d) = \lim_{\delta d\to 0} \frac{\frac{\|\delta x\|}{\|x\|}}{\frac{\|\delta d\|}{\|d\|}} = \lim_{\delta d\to 0} \frac{\|\delta x\|}{\|\delta d\|}\frac{\|d\|}{\|x\|} = \|F'(d)\|\frac{\|d\|}{\|x\|} = \|F'(d)\|\frac{\|d\|}{\|F(d)\|}$

Si $\vec{d} \in \mathbb{R}^p$ et $\vec{x} = F(\vec{d}) \in \mathbb{R}^q$, la dérivée $F'(\vec{d})$ est en fait la **matrice jacobienne**, c.-à-d.,

$$F'(\vec{d}) = J_F(\vec{d}) = \begin{bmatrix} \cdots & \cdots & \cdots \\ \cdots & \frac{\partial x_i}{\partial d_j} & \cdots \\ \cdots & \cdots & \cdots \end{bmatrix}$$

Si $q = 1$, la matrice jacobienne est le gradient $\nabla F(\vec{d})$
Autrement, les lignes de la matrice sont les gradients $\nabla F_i(\vec{d})$

Exemples

- **Le calcul d'une différence** $\vec{x} = F(\vec{d}) = \vec{d} - \vec{a}$

  (où nous supposons que a est représentée exactement en machine)
  Le conditionnement relatif du problème est

  $\kappa(\vec{d}) = \|F'(\vec{d})\| \frac{\|\vec{d}\|}{\|F(\vec{d})\|} = \frac{\|\vec{d}\|}{\|\vec{d} - \vec{a}\|}$

  Pour des scalaires, il en résulte que $\kappa(d) > M$ si $|d - a|/|d| < 1/M$.
  Donc, le problème est mal conditionné pour les valeurs de d autour de a.

  La soustraction de deux nombres proches est notoirement mal conditionnée

- **L'exponentiation** $x = F(d) = d^r$, où r est positif et grand.

  Alors $\kappa(d) = \|F'(d)\| \frac{\|d\|}{\|F(d)\|} = r\|d^{r-1}\| \frac{\|d\|}{\|d^r\|} = r$

  Le problème est mal conditionné pour toute valeur de d.

Conditionnement : considérations

- Un problème peut avoir un petit conditionnement $\kappa(d)$ pour certaines valeurs de d et un grand conditionnement pour d'autres valeurs.
- Si $\kappa(d)$ est grand pour toute donnée admissible d, le problème est dit **mal conditionné**.
- Un problème **mal posé** est forcement mal conditionné, mais mal conditionné ne veut pas dire mal posé (voir exemples ci-dessus)
- Le conditionnement associé à un problème est une mesure de la difficulté de résolution numérique du problème.
- le fait d'être bien conditionné est une propriété du **problème** qui est indépendante de **l'algorithme** choisi pour le résoudre.

Conditionnement : exemple (1)

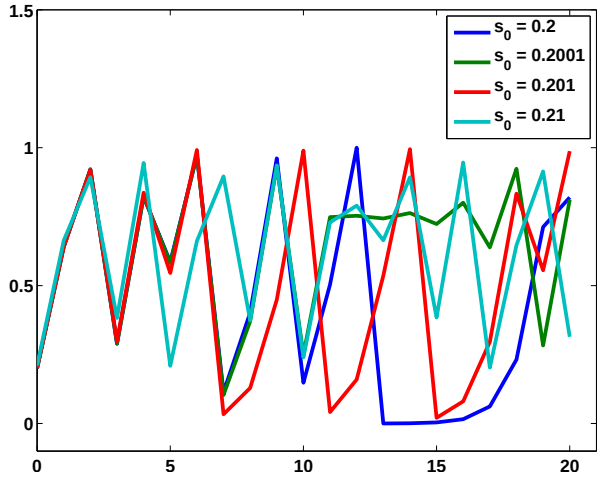
Considerons la série logistique $s_{k+1} = 4s_k(1 - s_k), \qquad 0 \leq s_k \leq 1$

Definissons le problème où $d = s_0$ et $x = s_{20}$. En faisant varier d on remarque qu'une très légère variation suffit à donner après quelques itérations des résultats x très différents.

Notons que ce problème est mal conditionné :

$$\begin{aligned} d = 0.2 &\Rightarrow x = 0.8200 \\ d = 0.2001 &\Rightarrow x = 0.8116 \\ d = 0.201 &\Rightarrow x = 0.9877 \\ d = 0.21 &\Rightarrow x = 0.3143 \end{aligned}$$

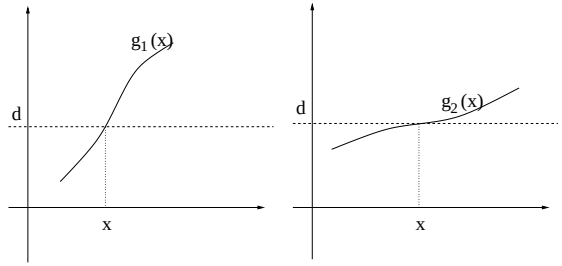
Conditionnement : exemple (1-bis)



Cet exemple est souvent cité pour montrer comment un comportement complexe peut émerger d'une équation dynamique non-linéaire (plus de détails dans INFO-F-305).

Conditionnement : exemple (2)

Considérons les deux problèmes implicites
$F_1(x,d) = g_1(x) - d = 0, \qquad F_2(x,d) = g_2(x) - d = 0$
où $g_1(x)$ et $g_2(x)$ sont

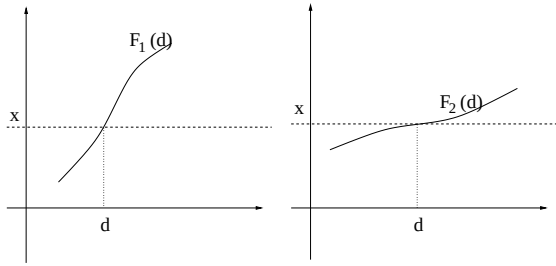


Lequel des deux est le mieux conditionné ?

— Une petite perturbation dans le niveau de d peut entraîner un changement considérable en x, dans le 2ième cas.
— Formellement : $x = g^{-1}(d)$, et donc (en notant $Df(x)$ la dérivée de la fonction $f(x)$)
$\kappa(d) \approx \left|\frac{Dg^{-1}(d)d}{g^{-1}(d)}\right| = \left|\frac{d}{g'(g^{-1}(d))g^{-1}(d)}\right| = \left|\frac{g(x)}{xg'(x)}\right|$
— Une dérivée proche de zero induit un grand conditionnement

Conditionnement : exemple (3)

Considérons les deux problèmes explicites
$x = F_1(d), \qquad x = F_2(d)$
où $F_1(d)$ et $F_2(d)$ sont



Lequel des deux est le mieux conditionné ?

— Une petite perturbation en d peut entraîner un grand changement dans le niveau de x, si la pente de la courbe $F(d)$ est forte
— $\kappa(d) \approx \left|\frac{dg'(d)}{g(d)}\right|$

- Problèmes
- **Solutions**
- Algorithmes

La résolution d'un problème

Deux approches peuvent être envisagées :

- **Résolution symbolique :** ceci utilise les propriétés analytiques et mathématiques du problème pour en dériver la solution x. Malheureusement une solution analytique n'est pas calculable pour tous les problèmes.
- **Résolution numérique :** ceci utilise une méthode numérique pour déterminer la solution x pour une valeur d donnée.

Exemple

— Considérons le problème élémentaire de la recherche des racines d'une équation du second ordre $d_1x^2 + d_2x + d_3 = 0$, où $d = [d_1, d_2, d_3]$.
— La solution analytique est $x = \frac{-d_2 \pm \sqrt{d_2^2 - 4d_1d_3}}{2d_1}$.
— La solution numérique pour $d = [0.1, 1.2, 0.3]$ est $x = -11.74$.

Pour/contre une approche numérique

Une méthode numérique présente des bénéfices aussi bien que des inconvénients par rapport à une solution analytique.

Les avantages tiennent au fait

1. qu'une solution numérique peut être obtenue aussi lorsqu'aucune solution analytique n'est disponible,
2. que la décomposition d'une méthode numérique en une longue série d'opérations arithmétiques élémentaires s'avère être facilement gérable par un ordinateur,
3. qu'une solution analytique, même si elle est disponible, requiert une évaluation numérique, qui en pratique, revient à une reformulation du problème original, cette fois sous forme explicite. Cette formule analytique peut bien être pire conditionnée que la formulation originale, implicite.

A son détriment, il faut mentionner que l'analyse et l'étude d'une solution numérique sont typiquement plus coûteuses.

- Problèmes
- Solutions
- **Algorithmes**

Typologie des algorithmes

— Algorithmes déterministes et stochastiques
— Algorithmes itératifs et directs
— Complexité des algorithmes
— Consistance
— Stabilité ; conditionnement asymptotique
— Convergence

Définition qualitative d'algorithme

Un algorithme est un texte fini qui spécifie l'exécution d'une série finie d'opérations élémentaires, conçue pour résoudre des problèmes d'une classe ou d'un type particulier.

Complexité d'un algorithme

- Plusieurs mesures m_C peuvent être adoptées pour caractériser la complexité d'un algorithme :
  — le nombre d'étapes
  — le temps d'exécution,
  — l'occupation de la mémoire
  — des mesures dépendantes de l'architecture du processeur.
- Dans certains cas l'utilisateur de l'algorithme est intéressé à avoir une idée qualitative du comportement de l'algorithme plutôt que le nombre exact d'opérations.
- Pour ce faire, la notion d'ordre de complexité (ou complexité asymptotique) à été introduite.
- Cette notion consiste à définir une mesure de complexité (par exemple le nombre d'operations arithmétiques) comme une fonction de la **taille** p des données du problème.

Ordre de complexité

— Si la complexité d'un algorithme dépend d'un paramètre p (représentant la taille des données du problème), la **complexité** $C(p)$ d'un algorithme est dite d'ordre $f(p)$ s'il existe deux constantes a et b telles que $m_C = C(p) \leq bf(p)$ pour tout $p \geq a$.

— Ceci est indiqué par la notation conventionnelle "grand O" (ordre de grandeur - order of magnitude) $C(p) = O(f(p))$.

Classes de complexité

Ordre	Classe de complexité	exemple de $C(p)$
$O(1)$	constante	$c \in \mathbb{R}^+$
$O(\log p)$	logarithmique	$c \log p$
$O(p)$	linéaire	$c_1 p + c_0$
$O(p^2)$	quadratique	$c_2 p^2 + c_1 p + c_0$
$O(p^3)$	cubique	$c_3 p^3 + c_2 p^2 + \ldots$
$\vdots$	$\vdots$	$\vdots$
$O(p^m)$, $m \in \mathbb{N}$	polynomiale	$c_m p^m + c_{m-1} p^{m-1} + \ldots$
$O(c^p)$	exponentielle	c^{dp}+polynôme(p)
$O(p!)$	factorielle	$cp!$

Complexité d'opérations arithmétiques

- La somme $\vec{d}_1 + \vec{d}_2$ de deux vecteurs $\vec{d}_1$ et $\vec{d}_2$ de taille p
  p additions, ordre de complexité linéaire
- La somme $A_1 + A_2$ de deux matrices carrées A_1 et A_2 de taille p : p^2 additions, ordre de complexité quadratique
- Le produit scalaire $\vec{d}_1 \cdot \vec{d}_2 = \sum_{i=1}^{p} d_{1,i} d_{2,i}$ de deux vecteurs de taille p : p multiplications plus $p-1$ additions, ordre de complexité linéaire
- Le produit matrice-vecteur, avec une matrice carrée et un vecteur de taille p :
  Cette opération consiste du calcul de p produits scalaires, donc la complexité est d'ordre quadratique
- Le produit de deux matrices carrées A_1 et A_2 de taille p
  Cette opération consiste de p multiplications matrice-vecteur, donc l'ordre de complexité est de $p \times p^2 = p^3$ (cubique)

Complexité d'un problème

- Notons que même si nous pouvons définir la complexité d'un **algorithme**, la notion de complexité d'un **problème** reste floue, car plusieurs algorithmes de complexité différente peuvent être utilisés pour résoudre le même problème.
- En général, nous définissons la **complexité d'un problème** comme étant la complexité de l'algorithme qui a la complexité la plus petite parmi ceux qui résolvent le problème.
- En général, il est fort difficile de preuver l'optimalité d'un algorithme du point de vue de la complexité.

Algorithmes numériques

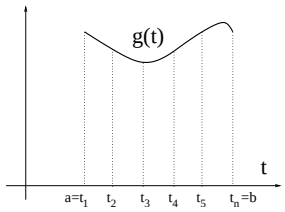
- **definition**(Algorithme numérique) étant donné un problème bien posé $F(x, d) = 0$, nous définissons l'algorithme numérique pour la résolution du problème F par la suite de problèmes approchés $F_1(\vec{x}^{(1)}, \vec{d}^{(1)}), F_2(\vec{x}^{(2)}, \vec{d}^{(2)}), \ldots, F_n(\vec{x}^{(n)}, \vec{d}^{(n)})$ dépendant d'un paramètre n.
- L'idée sous-jacente à la décomposition en sous-problèmes est que la résolution des F_n est plus simple que la résolution de F. Ceci rend possible l'exécution d'un algorithme par une machine.
- Différents algorithmes peuvent résoudre le même problème numérique.
- Il est alors nécessaire de choisir l'algorithme qui présente les meilleurs propriétés numériques et de complexité.

Pour une bonne compréhension

— La notion d'un algorithme numérique est assez générale et abstraite.
— La suite des probèmes approchés ne correspond pas (nécessairement) à la suite des étapes à effectuer dans l'algorithme final

Exemple 1 : intégration numérique

Supposons de connaître la forme analytique de la fonction univariée $g(t)$ et de vouloir calculer l'integral $x = \int_a^b g(t)dt$



L'algorithme, dit **la méthode du point milieu**, est basé sur la suite de problèmes approchés suivante $\vec{x}^{(n)} = \frac{b-a}{n} \sum_{k=1}^{n} g\left(\frac{t_k + t_{k+1}}{2}\right)$

On peut donc identifier $\vec{d}_{(n)} = [t_k, g(t_k)]_{k=1,\ldots,n}$

— Dans ce cas, les problèmes approchés ont des solutions différentes de celle du problème original, mais (en général), plus haute n, plus précise la solution approchée
— Pour l'implémentation, on va sélectionner une seule n telle que la solution du problème approché (l'approximation) soit de qualité suffisante

Exemple 2 : la résolution d'un système linéaire

Considérons le problème mathématique $F(x, d) = 0$ qui correspond à la résolution d'un système linéaire d'ordre 3 : $\begin{cases} 2x_1 + 4x_2 - 6x_3 = -4 \\ x_1 + 5x_2 + 3x_3 = 10 \\ x_1 + 3x_2 + 2x_3 = 5 \end{cases}$ Les données sont représentées par $d = \begin{bmatrix} 2 & 4 & -6 & -4 \\ 1 & 5 & 3 & 10 \\ 1 & 3 & 2 & 5 \end{bmatrix}$ et la solution est $x = [-3, 2, 1]$.

Considérons un algorithme de résolution numérique en 2 étapes où la première étape est représentée par le problème implicite

$F_1(\vec{x}^{(1)}, \vec{d}^{(1)}) = 0 \Leftrightarrow \begin{cases} 2x_1 + 4x_2 - 6x_3 = -4 \\ 3x_2 + 6x_3 = 12 \\ x_2 + 5x_3 = 7 \end{cases}$ où $\vec{d}^{(1)} = \begin{bmatrix} 2 & 4 & -6 & -4 \\ 0 & 3 & 6 & 12 \\ 0 & 1 & 5 & 7 \end{bmatrix}$

et $x^{(1)} = x = [-3, 2, 1]$.

NB : ne pas confondre la solution $x^{(n)}$ (vecteur) de la $n$ème étape de l'algorithme et le $n$ème composant (scalaire) de la solution x du problème.

Soit la deuxième étape représentée par le problème implicite

$F_2(x^{(2)}, \vec{d}^{(2)}) = 0 \Leftrightarrow \begin{cases} 2x_1 + 4x_2 - 6x_3 = -4 \\ 3x_2 + 6x_3 = 12 \\ 3x_3 = 3 \end{cases}$

avec $\vec{d}^{(2)} = \begin{bmatrix} 2 & 4 & -6 & -4 \\ 0 & 3 & 6 & 12 \\ 0 & 0 & 3 & 3 \end{bmatrix}$ et $x^{(2)} = x = [-3, 2, 1]$.

Exemple 2 : considérations

- La construction des problèmes approchés fait partie de la procedure numérique
- Chaque problème intermédiaire a la même solution, qui est la solution du problème original
- Donc, le choix du n n'est pas motivé par la qualité de la solution (qui est constante), mais la complexité du problème réduit

Algorithmes itératifs et directes

- **Rappel : définition générale** Une suite de problèmes approchés $F_1(\vec{x}^{(1)}, \vec{d}^{(1)}), F_2(\vec{x}^{(2)}, \vec{d}^{(2)}), \ldots, F_n(\vec{x}^{(n)}, \vec{d}^{(n)})$
  Deux catégories importantes : algorithmes itératifs et directes
- **Algorithmes directes** Le nombre n est fixe (ou au moins majoré) souvent par une fonction de la taille du probème
- **Algorithmes itératifs**
  — Le nombre n est non-borné (et donc certainement sujet de la conception)
  — Il existe une routine $f(u)$ admissible, indépendant de n, telle que $\vec{x}^{(n)} = f(\vec{x}^{(n-1)})$

Propriétés d'un algorithme numérique

Un algorithme numérique transforme un problème en une série de sous-problèmes.

Les propriétés souhaitées d'un algorithme numérique sont

- **Consistance** : ceci concerne le lien entre le problème original (et sa solution théorique) et les problèmes intermédiaires de l'algorithme (ainsi que leurs solutions théoriques).
- **Stabilité** : ceci concerne la robustesse de la solution d'un algorithme à des perturbations possibles, c.-à-d., le ralation entre la solution théorique et le résultat dans la pratique.
- **Convergence** : ceci concerne la convergence de la solution de l'algorithme vers la solution du problème.

Consistance

— Rappelons nous que $F_n(\vec{x}^{(n)}, \vec{d}^{(n)}) = 0$ signifie que $\vec{x}^{(n)}$ est la solution du $n$ème sous-problème.
— **définition**

> Un algorithme $F_n(\vec{x}^{(n)}, \vec{d}^{(n)})$ est dit **consistant** si $\lim_{n\to\infty} F_n(\vec{x}, \vec{d}^{(n)}) = F(\vec{x}, \vec{d}) = 0$,

c.-à-d., si la solution exacte $\vec{x}$ du problème est parmi les solutions du sous-problème $F_n(\vec{x}^{(n)}, \vec{d}^{(n)}) = 0$ pour $n \to \infty$
— Un algorithme itératif $\vec{x}^{(n+1)} = f(\vec{x}^{(n)})$ est donc consistant si $\vec{x}^{(n)} = x \Rightarrow \vec{x}^{(n+1)} = x$
Cette condition est suffisante mais pas nécessaire
— Attention : être consistant ne signifie pas que $\vec{x}^{(n)}$ converge vers x mais seulement que l'algorithme a été conçu de manière à avoir la solution x parmi les solutions de ses sous-problèmes pour $n \to \infty$.

Consistance forte

définition

> Un algorithme $F_n(\vec{x}^{(n)}, \vec{d}^{(n)})$ est dit fortement consistant si $F_n(x, \vec{d}^{(n)}) = 0$ pour tout n.

Ceci signifie que tous les sous-problèmes F_n ont x parmi leurs solutions.

Exemple : algorithme consistant

— L'intégration du point milieu : L'algorithme est consistant mais pas fortement.
— Annihilation d'une matrice : L'algorithme est fortement consistant.

Stabilité d'un algorithme

Soit $F_n(\vec{x}^{(n)}, \vec{d}^{(n)}) = 0$ un algorithm numérique pour le problème $F(\vec{x}, \vec{d})$, composé d'étapes
$\vec{x}^{(n)} = F^{(n,m)}(\vec{d}^{(n,m-1)})$ où $\vec{d}^{(n,i)} = F^{(n,i)}(\vec{d}^{(n,i-1)})$ et $\vec{d}^{(n,0)} = \vec{d}^{(n)}$

Alors, définissons la résolution en pratique comme le résultat de l'algorithme avec les étapes perturbées $\bar{x}^{(n)} = F^{(n,m)}(\bar{d}^{(n,m-1)}) + \delta\vec{d}^{(n,m)}$ où $\bar{d}^{(n,i)} = F^{(n,i)}(\bar{d}^{(n,i-1)}) + \delta\vec{d}^{(n,i)}$ et $\bar{d}^{(n,0)} = \vec{d}^{(n)}$

définition(Stabilité)

Un algorithme numérique est dit **stable** (ou bien posé) si

1. il existe pour tout n une solution unique $\vec{x}^{(n)}$
2. pour chaque $\epsilon > 0$ il existe un $\eta > 0$ et un n_0 tel que $\|\delta\vec{d}^{(n,i)}\| \leq \eta$ pour tout $i = 1, \ldots, m \Rightarrow \|\bar{x}^{(n)} - \vec{x}^{(n)}\| \leq \epsilon$ pour tout $n > n_0$

Stabilité d'un algorithme itératif

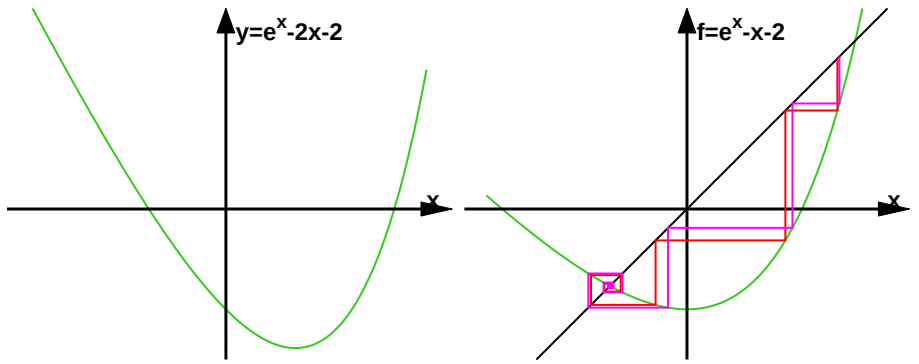
Si l'opération $F^{(n,m)}(\vec{d})$ est indépendante de n et m, on a un algorithme itératif, où $\vec{x}^{(n)} = f(\vec{x}^{(n-1)})$

La notion de stabilité se traduit par la condition qu'il existe une $0 \leq R < 1$ telle que $\|f(\vec{x}_1) - f(\vec{x}_2)\| \leq R\|\vec{x}_1 - \vec{x}_2\|$ dans un voisinage de la solution $\vec{x}$, et si f est différentiable, on a que $\|f'(\vec{x}_1)\| \leq R$

Exemple Supposons que nous cherchons la racine de $x = \cos(x)$ Nous savons que $|\cos'(x)| = |\sin(x)| < 1$ quand $x \neq k\pi$ où $k \in \mathbb{Z}$, donc l'algorithm itératif $x^{(n)} = \cos(x^{(n-1)})$ est stable.

Exemple Supposons que nous cherchons la racine positive de $e^x - 2x - 2 = 0$ et nous proposons l'algorithme $x^{(n)} = e^{x^{(n-1)}} - x^{(n-1)} - 2$. Cet algorithme est consistant, mais il n'est pas stable autour la solution positive. Une petite perturbation peut induire des résultats complètement différentes.

Illustration de stabilité et d'instabilité



Conditionnement asymptotique relatif

— Nous appelons conditionnement asymptotique relatif de la méthode numérique la quantité $\kappa^{\text{num}}(\vec{d}^{(n)}) = \lim_{k\to\infty} \sup_{n\geq k} \kappa_n(\vec{d}^{(n)})$, où

$$\kappa_n(\vec{d}^{(n)}) = \sup_{\delta\vec{d}^{(n)}\in D_n} \frac{\|\delta\vec{x}^{(n)}\|/\|\vec{x}^{(n)}\|}{\|\delta\vec{d}^{(n)}\|/\|\vec{d}^{(n)}\|}.$$

— L'algorithme numérique est dit bien conditionné si κ^{num} est "petit" pour toutes données $\vec{d}^{(n)}$ et mal conditionné sinon.

Exemple

— Considérons le problème mathématique $2x = 1$

— et les deux algorithmes itératives suivants, où $\vec{x}^{(0)} = 1/2 + \delta$

1. $\vec{x}^{(n)} = \vec{d}^{(n)}\vec{x}^{(n-1)} + \frac{1}{4}, \qquad \vec{d}^{(n)} = \frac{1}{2}$
2. $\vec{x}^{(n)} = \vec{d}^{(n)}\vec{x}^{(n-1)} - \frac{1}{2}, \qquad \vec{d}^{(n)} = 2$

— Tous les deux sont consistants mais uniquement le premier algorithme est stable. (Algorithme itératif avec dérivée au-dessous de 1)

— Dans le premier cas uniquement, un petit changement $\delta\vec{d}^{(n)}$ des données entraîne un changement borné de la solution $\vec{x}^{(n)}$ pour tous n (script MATLAB `s_stable2.m` et `s_unstable2.m`).

Exemple : algorithme instable

- Considérons le problème mathématique $x(d) = \int_0^1 z^d e^{-z} dz$, où la solution $x(n)$ pour un certain $n \in \mathbb{N}$ est requise.
- On peut montrer que
  $e^{-1} = x(0) > x(1) > x(2) > \ldots > x(n-1) > x(n) > \ldots > 0$
- Considérons un algorithme basé sur l'intégration par parties. Chaque étape consiste en $x(n) = n \cdot x(n-1) - e^{-1}$ où $x(0) = 1 - e^{-1}$
- **En matlab**
  (attention au fait qu'en matlab, un vecteur `x` ne peut pas avoir un composant `x(0)`, donc `x(1)` en matlab correspond à $x(0)$)

```
D = 20;
x = zeros(1,D+1); x(1) = 1-1/exp(1);

for d=1:D,
   x(d+1) = d*x(d)-1/exp(1);
end
```

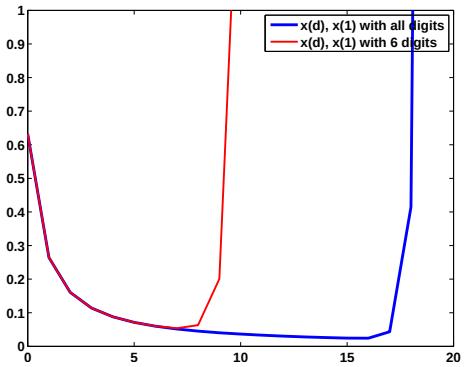

Exemple : algorithme instable

- Le conditionnement κ_n étant une fonction linéaire de n, on déduit que l'algorithme est instable.
- On peut montrer que deux itérations de l'algorithme, ayant des conditions initiales très proches, divergent. (ceci est un exemple de la théorie du chaos)
- **Remarque**
  Les résultats sur la page suivante sont différents si on remplace `1/exp(1)` par `exp(-1)` pour le calcul de $e^{-1} = 1/e$. Cette observation démontre, une fois de plus, la sensitivité de l'algorithme.

Exemple : algorithme instable (II)

Pour illustrer, nous effectuons le script matlab avec la valeur initiale, $1 - 1/e$, encodée en machine avec 6 chiffres :

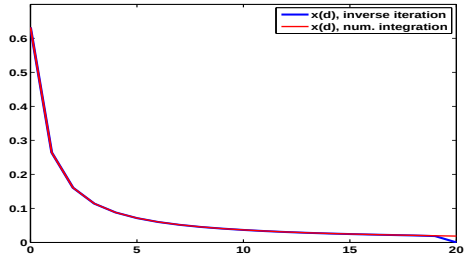
```
x(1) = round(x(1)*1.e6)*1.e-6;
```



Il est important aussi de remarquer que la solution calculée $x(n)$ ne respecte pas les contraintes analytiques.

Exemple : algorithme stable

- Le deuxième algorithme est basé sur un principe différent. Au lieu d'itérer pour n croissant, on itère pour n décroissant selon la formule $x(n-1) = (x(d) + 1/e)/d$ où $x(n) = 0$ pour un n suffisamment grand.
- On peut montrer que cet algorithme est stable.



- La figure prend note de la stabilité en comparaisant le résultat des itérations avec le résultat d'une intégration numérique pour chaque valeur de d séparément. (routine qui est plus compliquée de point de vue nombres de calculs)

Convergence d'un algorithme

Soit $\bar{x}^{(n)} = \vec{x}^{(n)} + \delta\vec{x}^{(n)}$ la suite des solutions approchées produite par l'algorithme suite aux perturbations $\{\delta\vec{d}^{(i)}\}$, $i = 1, \ldots, n$ des données.

— **definition**(Convergence) Un algorithme est convergent si $\lim_{n\to\infty, \delta\vec{d}^{(n)}\to 0} F_n(x + \delta\vec{x}^{(n)}, \vec{d}^{(n)} + \delta\vec{d}^{(n)}) = 0$ $\lim_{n\to\infty, \delta\vec{d}^{(i)}\to 0} \bar{x}^{(n)} = \lim_{n\to\infty, \delta\vec{d}^{(i)}\to 0} \vec{x}^{(n)} + \delta\vec{x}^{(n)} = x$ ou, en d'autres termes, si pour chaque $\epsilon > 0$ il existe $\eta > 0$ et un n_0 tels que pour tous $n > n_0$ $\|\delta\vec{d}^{(i)}\| \leq \eta \Rightarrow \|\bar{x}^{(n)} - x\| \leq \epsilon$

— Les concepts de stabilité et convergence sont fortement liés.

— Les méthodes numériques que nous allons considérer satisfont les conditions du théorème de Lax-Richtmyer (ou théorème d'équivalence) selon lequel pour un algorithme numérique consistant, la stabilité est équivalente à la convergence.