UNIVERSITÉ LAVAL

MAT-2910: Analyse numérique pour ingénieurs

Thomas Briffard

Janvier 2020

GIREF, Université Laval

Qui suis-je : Thomas Briffard.

Me contacter : **thomas.briffard@giref.ulaval.ca**.

Où me trouver : VCH-3858.

Disponibilités : Mercredi 13h30-14h30 & Jeudi 15h30-16h30, m'envoyer un courriel pour prise de rendez-vous en dehors des disponibilités.

Manuel : Analyse numérique pour ingénieurs, André Fortin (5^{eme} édition de préférence).

Logiciel : Matlab [1].

Site du cours : monportail.ulaval.ca

- Infos générales,
- Sujet du devoir, exercices,
- Documents d'aide pour Matlab,...

CDA (PLT-2576): Dépannage individuel (gratuit), aide aux exercices, voir horaires sur le **site du CDA**.

[1] **Banque de programmes Matlab.**

Évaluations :

- 2 Mini-tests (en ligne, 2 x 10%), 2 examens (2 x 30%), 2 devoirs avec Matlab (2 x 10%),
- Il faut une moyenne d'au moins 50% au total, et 50% de moyenne à l'ensemble constitué des deux examens,
- Devoirs en équipe de 2 (vous sera confirmé), 3 semaines pour les rendre. Gestion des équipes par le portail.

Fonctionnement du cours: «squelette» du cours sur acétates, exemples et compléments au tableau en classe.

Conflit d'horaire, demande de reprise, accommodement : **communiquer le plus tôt possible !**

La simulation numérique est devenue un outil indispensable dans l'industrie de pointe, permettant entre autres de:

- Réduire les coûts de prototypage,
- Avoir accès à des mesures physiques très difficiles à mesurer expérimentalement.

Pierre-Louis Lions (Médaillé Fields, 1994) :

> *En quelques décennies, les **simulations numériques** sont devenues un **outil privilégié** d'investigation dans les sciences et les technologies. Elles ont pour but de **reproduire par le calcul** le comportement d'un système décrit par un modèle très souvent constitué d'équations aux dérivées partielles. (…) L'essor des simulations numériques renforce donc la **nécessité de l'étude mathématiques (analyse) de ces équations et de leur résolution numérique**...*

Ce que vous savez déjà faire analytiquement :

- Résolution de certaines EDO, ex : $2y' = -3xy \Rightarrow$ séparation des variables : $y = Ce^{-\frac{3}{4}x^2}$
- Dérivation, intégration,...

Pour ces exemples, il est possible de faire les calculs analytiquement, et donc d'obtenir une solution exacte, mais cela n'est pas toujours possible...

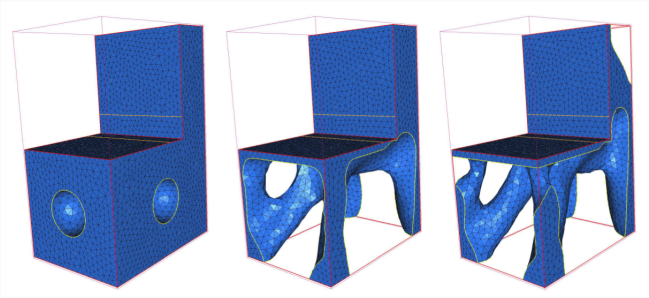
Ce que nous allons voir :

- Des méthodes numériques permettant de résoudre avec un ordinateur des problèmes mathématiques répandus dans la «vraie vie» (système linéaire, EDO, intégrales, interpolation,...)
- Analyser la pertinence, précision des algorithmes ainsi que le coût et la complexité de mise en oeuvre,
- Chercher à évaluer l'erreur commise par l'utilisation d'une méthode numérique.

Pour terminer l'introduction, un exemple de simulation numérique pour de l'optimisation de forme.

L'idée est de chercher à modifier la géométrie d'un objet afin de trouver la forme «optimale» répondant à certains critères.

Autrement dit: quelle est la répartition de matière idéale dans un volume donné soumis à certaines contraintes ?



Itérations du processus d'optimisation de forme

Ce qui donne, une fois utilisé par l'industrie[2]:

[2] **"Light Rider"**

Chapitre 1: Analyse d’erreurs

On souhaite simuler des phénomènes physiques : obtenir une **solution approchée**, **approximation**, d'une solution exacte le plus souvent inconnue.

Plusieurs sources d'erreurs vont intervenir, à différents niveaux:

- **Erreur de modélisation :** par exemple en négligeant certains termes physiques,
- **Erreur de représentation en machine:** pour un ordinateur il est impossible de représenter de manières exactes des nombres tels que $\frac{1}{3}, \pi, \ldots \rightarrow$ on procédera à l'arrondi, ou troncature,
- **Erreur de troncature:** certaines opérations ($\log$, $\exp$, $\sin$,$\cos$) sont approchées par une **somme finie** d'opérations élémentaires (+,-,x,/). Cependant, le plus souvent, pour être exacte, la somme devrait être infinie...
  Ce dernier type d'erreur interviendra notamment lors de l'étude des **développements de Taylor.**

Quelques catastrophes dues à une mauvaise utilisation de l'outil numérique :
http://ta.twi.tudelft.nl/users/vuik/wi211/disasters.html

Deux définitions pour commencer. Une mesure **quantitative**...

Définition (Erreur absolue)

Soit x un nombre réel, et x^* une approximation de ce nombre. L'erreur absolue est définie par

$$E_a(x^*) = |x - x^*|$$

...ainsi qu'une mesure **qualitative**

Définition (Erreur relative)

Soit x un nombre réel, et x^* une approximation de ce nombre. L'erreur relative est définie par

$$E_r(x^*) = \frac{|x - x^*|}{|x|} = \frac{E_a(x^*)}{|x|}$$

En multipliant E_r par 100, on obtient l'erreur relative en %.

⚠ En pratique, x n'est pas connu !

On exhibera le plus souvent une **borne supérieure** de l'erreur absolue, notée Δx, *i.e*

$$E_a(x^*) = |x - x^*| \leq \Delta x$$

Par abus de langage, **on confondra la borne Δx avec $E_a(x^*)$.**

Pour l'erreur relative E_r, on a alors

$$E_r(x^*) = \frac{|x - x^*|}{|x|} \leq \frac{\Delta x}{|x|}$$

et si Δx est suffisamment «petit», on dira que

$$E_r(x^*) \approx \frac{\Delta x}{|x^*|}$$

Remarque(s)

$|x - x^*| \leq \Delta x \iff -\Delta x \leq x - x^* \leq \Delta x \iff x^* - \Delta x \leq x \leq x^* + \Delta x$

Autrement dit, $x^* = x \pm \Delta x$, Δx représente ainsi l'incertitude.

En conclusion, **sauf si l'on connaît à la fois x et x^***, on utilisera:

- Δx comme valeur pour l'erreur absolue,
- $\frac{\Delta x}{|x^*|}$ comme valeur pour l'erreur relative.

Pour mesurer la précision d'une approximation, on introduit la notion de chiffres significatifs (C.S).

Définition (Chiffres significatifs)

Si l'erreur absolue vérifie $\Delta x \leq 0,5 \times 10^m$, alors le chiffre dans x^* correspondant à la m^e puissance de 10 est dit *significatif* et tous ceux à sa gauche (correspondant aux puissances de 10 supérieures à m) le sont aussi. On arrête le compte au dernier chiffre non nul.

Si le chiffre correspondant à la m^e puissance de 10 dans x^* est nul ainsi que tous ceux à sa gauche, on dit qu'il n'y a aucun chiffre significatif.

Inversement, on estime l'erreur absolue (Δx) à partir du nombre de chiffres significatifs de la manière suivante:

• Si une approximation x^* possède n chiffres significatifs, on commence à compter à partir du premier chiffre non nul à gauche, et l'erreur absolue est inférieure à 0,5 fois la puissance de 10 correspondant au dernier chiffre significatif.

☢ Attention !

Si x^* possède n chiffres significatifs, cela ne veut pas dire que x^* possède n chiffres exacts ! **Mais n chiffres dont on contrôle l'erreur.**

Représentation en virgule flottante: Soit x un nombre réel. La représentation en virgule flottante de x dans **la base b** sera

$$\begin{aligned} x = \pm m \times b^l &= \pm 0, d_1 d_2 \cdots d_n \cdots \times b^l \\ &= \pm (d_1 \times b^{-1} + d_2 \times b^{-2} + \cdots + d_n \times b^{-n} + \cdots) \times b^l \end{aligned}$$

où

- $m = 0, d_1 d_2 \cdots d_n \cdots$ est la **mantisse**, par convention on prendra $m < 1$.
- b est la **base**, $b \in \mathbb{N}$,
- l est l'**exposant**, $l \in \mathbb{Z}$.

De plus, les entiers d_i dans l'écriture de la mantisse vérifient:

- $0 \leq d_i \leq b-1,\ i = 2, 3, \cdots$
- $0 < d_1 \leq (b-1)$, cela permet d'assurer l'unicité de la représentation. On parlera de virgule flottante à **mantisse normalisée**.

Dans ce cas, on peut montrer

$$\frac{1}{b} \leq m < 1$$

à noter que l'écriture de 0 devient une exception (car la mantisse ne s'annule jamais !).

- Le choix de la base b est arbitraire, cependant il y a des choix naturels :
  - b=10, base décimale : utilisée par les humains,
  - b=2, base binaire, ou b=16, base hexadécimale : utilisées par les ordinateurs.

- **Quelle que soit la base choisie**, les nombres irrationnels ($\pi, \sqrt{2}, \cdots$) auront toujours une mantisse de longueur infinie,
- L'utilisation de mantisse infinie sera bien évidemment impossible sur un ordinateur, où le nombre de bits pour représenter un nombre réel est limité. Ainsi, on recourt à la troncature ou à l'arrondi pour réduire la mantisse à N chiffres. On aura donc

$$m = d_1 \times b^{-1} + d_2 \times b^{-2} + \cdots + d_N \times b^{-N}$$

La représentation de certains réels sur ordinateur sera donc erronée. De plus les opérations élémentaires utilisant ces nombres seront également erronées.

Troncature et arrondi:

- Troncature à N chiffres: on coupe la mantisse au-delà de la position N.
- Arrondi à N chiffres: On ajoute (5 en décimal, 1 en binaire) à la position $N+1$, puis on fait une troncature à N chiffres.

Définition (Précision machine)

On appelle précision machine (notée ε_m) la plus grande erreur relative commise en représentant un nombre réel sur ordinateur en utilisant la troncature:

$$\varepsilon_m = \max_x \frac{\Delta x}{|x|} = \max_x E_r(x).$$

En utilisant l'arrondi, la plus grande erreur relative est $\frac{\varepsilon_m}{2}$.

Il est possible de montrer que l'on a (p. 12-13)

$$\varepsilon_m \leq b^{1-N}$$

N étant le nombre de chiffres dans la mantisse.

- En pratique, on ne fera pas de distinctions entre ε_m et b^{1-N},
- Norme IEEE-754 (p 14-17): standardisation de la représentation. Garantis l'uniformité du comportement entre les ordinateurs.

  Précision machine [3] :
  - $2^{1-24} = 0,119 \times 10^{-6}$ en simple précision (32 bits)
  - $2^{1-53} = 0,222 \times 10^{-15}$ en double précision (64 bits)

Conclusion: En général les nombres ne seront plus représentés exactement. Il y aura des erreurs dans le résultat d'opérations arithmétiques de base [4] !

[3] voir `script_precision_machine.m` pour le calcul de la précision machine.

[4] voir `script_somme.m`.

Définition (Notation flottante)

Soit $x \in \mathbb{R}$. On note $fl(x)$ sa **représentation en virgule flottante décimal** à n chiffres

$$fl(x) = \pm 0, d_1 d_2 d_3 \cdots d_n \times 10^l$$

le dernier chiffre d_n dépend du procédé retenu pour éliminer les derniers chiffres, à savoir la troncature ou l'arrondi. **On utilisera de préférence l'arrondi**.

On utilise la base décimale, cependant les effets décrits par la suite valent également pour les autres bases !

Principe de fonctionnement des opérations élémentaires: On doit représenter les deux opérandes en notation flottante, effectuer l'opération de la façon habituelle et exprimer le résultat en notation flottante.

Opérations élémentaires:

- $x + y \longrightarrow fl(fl(x) + fl(y))$
- $x - y \longrightarrow fl(fl(x) - fl(y))$
- $x \div y \longrightarrow fl(fl(x) \div fl(y))$
- $x \times y \longrightarrow fl(fl(x) \times fl(y))$

- $+/-$: on devra faire un «décalage » de la mantisse lorsque les exposants ne sont pas les mêmes. Cela impliquera d'ajouter des zéros dans le nombre le plus petit (pour ramener son exposant au niveau du plus grand nombre).
- $\times/\div$: la loi des exposants permet de faire la somme (différence) des exposants et uniquement le produit (la division) des mantisses pour obtenir le résultat.

☢ Attention ! Opérations risquées

- Le décalage de la mantisse peut se révéler **très dangereux si on additionne des nombres dont les ordres de grandeur sont très différents**.
- La troncature ou l'arrondi pouvant se faire sur certains termes intermédiaires, **l'associativité et la distributivité** n'existe plus: **l'ordre des opérations** devient important pour le résultat.
- Il peut être dangereux de soustraire des **nombres très proches**: perte de précision (chiffres significatifs).

Réelle source de problèmes en pratique !

Suivant le problème, il existe certaines méthodes de contournement : additionner les termes en les plaçant par ordre croissant, somme de Kahan (p. 22), . . .

Conclusion :

- Les opérations élémentaires sont sources d'erreurs,
- Pas de solution miracle répondant à tous les problèmes énoncés auparavant !
- On essayera de **minimiser le nombre d'opérations**. À ce titre, le nombre d'opérations élémentaires en virgule flottante sert souvent de point de comparaison entre différents algorithmes.

Voyons l'**algorithme de Horner** permettant de réduire le nombre d'opérations pour l'évaluation d'un polynôme.

Algorithme de Horner : multiplications imbriquées.
Soit le polynôme p de degré n, $p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$.

En regroupant judicieusement les termes, on peut écrire:

$$\begin{aligned} p(x) &= a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_n x^n \\ &= a_0 + x(a_1 + a_2 x + a_3 x^2 + \cdots + a_{n-1} x^{n-2} + a_n x^{n-1}) \\ &= a_0 + x(a_1 + x(a_2 + a_3 x + \cdots + a_{n-1} x^{n-3} + a_n x^{n-2})) \\ &= \cdots \\ &= a_0 + x(a_1 + x(a_2 + x(a_3 + \cdots + x(a_{n-1} + a_n x) \cdots))) \end{aligned}$$

- Approche directe : au plus $\frac{n(n+1)}{2} + n$ opérations élémentaires.
- Horner : au plus $2n$ opérations élémentaires.

Algorithme de Horner, pseudo-code `script_Horner.m`

Données :

- Les coefficients a_i d'un polynôme p de degré n,
- Une abscisse x.

Résultat : la variable r qui contiendra la valeur du polynôme p évaluée en x

$r = a_n$;
pour $i = n, n-1, \cdots, 2, 1$ **faire**
  $r = a_{i-1} + xr$;
fin
$p(x) = r$;

Tout cela concerne l'erreur de représentation. Qu'en est-il de l'erreur de troncature ?

$\rightarrow$ Développement de Taylor

- Quel est le polynôme de degré n, noté $\boldsymbol{P_n(x)}$, qui donne la meilleure approximation d'une **fonction f donnée**, cela **au voisinage d'un point x_0, également donné** ?

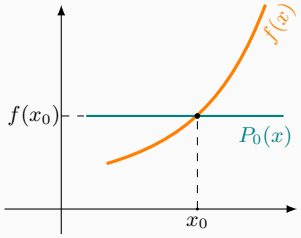
→ Le polynôme (ou formule) de Taylor nous donne une méthode de construction des polynômes $P_n(x)$, **lorsque la fonction f est suffisamment dérivable.**

Définition (Polynôme de Taylor)

Le polynôme de Taylor de degré n de la fonction $f(x)$, autour du point x_0 est défini par

$$\begin{aligned} P_n(x) =& f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \\ & \frac{f'''(x_0)}{3!}(x - x_0)^3 + \cdots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n \\ =& \sum_{i=0}^{n} \frac{f^{(i)}(x_0)}{i!}(x - x_0)^i \end{aligned}$$

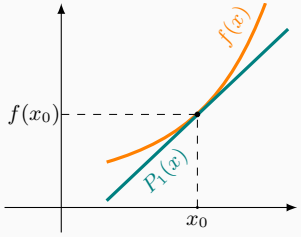
Note: la fonction f doit être n fois dérivable en x_0



- $n = 0$

$P_0(x) = f(x_0)$.

Cela signifie que l'on approche f par une constante, un polynôme de degré 0.

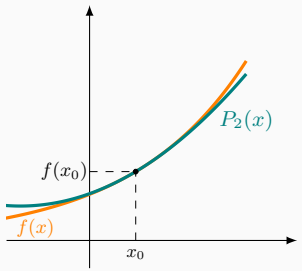


- $n = 1$

$P_1(x) = f(x_0) + f'(x_0)(x - x_0)$.

Cela signifie que l'on approche f par une droite, un polynôme de degré 1.

Note: $P_1(x)$ correspond à l'équation de la tangente à la courbe f en x_0



- $n = 2$

$$P_2(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2}(x - x_0)^2.$$

Cela signifie que l'on approche f par une quadratique, un polynôme de degré 2.

Approcher une fonction f par un polynôme de Taylor P_n va induire une erreur[5], donnée par l'expression

$$|f(x) - P_n(x)|$$

et dite **erreur de troncature**. La question est maintenant :

Suis-je en mesure de quantifier cette erreur ?

[5] voir `script_Taylor.m` pour exemples numériques avec Matlab.

Théorème (Développement de Taylor-Lagrange)

Soit f une fonction $\boldsymbol{n+1}$ **fois dérivable** autour de x_0, et $P_n(x)$ le polynôme de Taylor de degré n autour de x_0. On a **l'égalité** suivante

$$f(x) = P_n(x) + R_n(x)$$

R_n est «l'écart» entre f et P_n, défini par

$$R_n(x) = \frac{f^{(n+1)}(\xi_x)}{(n+1)!}(x - x_0)^{n+1}$$

où ξ_x est un nombre réel, dépendant de x, compris entre x et x_0, *i.e*

- $x \leq \xi_x \leq x_0$ si $x < x_0$
- $x_0 \leq \xi_x \leq x$ si $x > x_0$

L'erreur de troncature est ainsi définie par

$$|f(x) - P_n(x)| = |R_n(x)| = \left| \frac{f^{(n+1)}(\xi_x)}{(n+1)!}(x - x_0)^{n+1} \right| = \left| \frac{f^{(n+1)}(\xi_x)}{(n+1)!} \right| \left| (x - x_0)^{n+1} \right|$$

Remarque(s)

À n fixé, l'erreur augmente si x s'éloigne de x_0 et diminue si x se rapproche de x_0. Il est donc préférable d'avoir x_0 proche des points où l'on souhaite évaluer la fonction f...

On cherche à évaluer l'erreur de troncature

$$|R_n(x)| = |f(x) - P_n(x)| = \left| \frac{f^{(n+1)}(\xi_x)}{(n+1)!}(x - x_0)^{n+1} \right|$$

Cependant ξ_x n'est pas connu !!! On sait seulement qu'il existe et qu'il dépend de n, x, x_0, f.

Afin d'estimer l'erreur, le but du jeu va donc être de chercher à se «débarrasser» de la variable ξ_x.

Pour cela, on va chercher à majorer le terme $\left|f^{(n+1)}(\xi_x)\right|$, *i.e* trouver $M \in \mathbb{R}^+$ tel que

$$\left|f^{(n+1)}(\xi_x)\right| \leq M$$

On aura ainsi $|R_n(x)| = |f(x) - P_n(x)| \leq \frac{M}{(n+1)!}\left|(x-x_0)^{n+1}\right|$. On contrôle ainsi le terme d'erreur.

Formulation en h du développement de Taylor :

On utilise souvent une autre forme du développement de Taylor, obtenu en effectuant le changement de variable $\boldsymbol{h = x - x_0}$. On obtient ainsi

$$f(x) = f(x_0 + h) = P_n(x_0 + h) + R_n(x_0 + h) = \tilde{P}_n(h) + \tilde{R}_n(h)$$

avec

$$\tilde{P}_n(h) = f(x_0) + f'(x_0)h + \frac{f''(x_0)}{2!}h^2 + \frac{f'''(x_0)}{3!}h^3 + \cdots + \frac{f^{(n)}(x_0)}{n!}h^n$$

$$\tilde{R}_n(h) = \frac{f^{(n+1)}(\xi_h)}{(n+1)!}h^{n+1}$$

avec ξ_h compris entre x_0 et $x_0 + h$.

ξ_h compris entre x_0 et $x_0 + h$:

- Si $h > 0 \rightarrow \ \xi_h \in [x_0, x_0 + h] \iff x_0 \leq \xi_h \leq x_0 + h$
- Si $h < 0 \rightarrow \ \xi_h \in [x_0 + h, x_0] \iff x_0 + h \leq \xi_h \leq x_0$

Avec la formulation en h on cherchera une majoration du terme d'erreur R_n de la forme

$$|R_n(x_0 + h)| = \left| \frac{f^{(n+1)}(\xi_h)}{(n+1)!} \right| |h|^{n+1} \leq \frac{M}{(n+1)!} |h|^{n+1}$$

Encore une fois, on cherchera une majoration de la fonction $|f^{(n+1)}|$ ne faisant plus apparaitre ξ_h.

Au final, on a donc les deux expressions équivalentes suivantes:

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n + \frac{f^{(n+1)}(\xi_x)}{(n+1)!}(x - x_0)^{n+1}$$

$$f(x_0 + h) = f(x_0) + f'(x_0)h + \frac{f''(x_0)}{2!}h^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}h^n + \frac{f^{(n+1)}(\xi_h)}{(n+1)!}h^{n+1}$$

Attention !

Il faut savoir manipuler les deux formulations du développement de Taylor, et ne pas les mélanger !

Définition (Notation de Landau - grand ordre)

Une fonction $e(h)$ est un grand ordre de h^n au voisinage de 0, on notera $e(h) = O(h^n)$ s'il existe une constante C telle que, au voisinage de 0,

$$|e(h)| \leq Ch^n$$

- **Lorsque h est assez petit**, la fonction $O(h^n)$ décroît comme Ch^n,
- Plus n est grand, plus la décroissante est rapide,
- Une fonction $O(h^p)$ est obligatoirement $O(h^m)$ pour $m < p$.
- Comment déterminer le grand ordre d'une fonction ? Pour h petit,

$$e\left(\frac{h}{2}\right) \approx C\left(\frac{h}{2}\right)^n = C\frac{h^n}{2^n} \Rightarrow \frac{e(h)}{e\left(\frac{h}{2}\right)} \approx 2^n$$

à titre d'exemple, si $e(h) = O(h^n)$ représente une erreur, en divisant h par 2, l'erreur sera divisée par 2^n.

Définition (Ordre d'une approximation)

Une approximation dont le **terme d'erreur** est un grand ordre de h^n ($O(h^n)$) est dite **approximation d'ordre** n.

• Suivant cette définition, le polynôme de Taylor de degré n sera généralement, mais pas toujours, une approximation d'ordre $n+1$, *i.e*

$$R_n(x) = O((x-x_0)^{n+1}) \text{ ou } R_n(h) = O(h^{n+1}).$$

Un contre exemple est le développement de Taylor d'ordre 5 de $f(x) = \sin(x)$ autour de 0 : il suffit de calculer le polynôme de Taylor de degré 3 pour avoir une approximation d'ordre 5.

• NE PAS CONFONDRE **le degré** du polynôme P_n **avec l'ordre** du développement de Taylor

- Le degré est la puissance de h (ou $(x-x_0)$) la plus grande dans le polynôme P_n,
- L'ordre est la puissance de h (ou $(x-x_0)$) intervenant dans le terme d'erreur R_n, et sert à indiquer la qualité de l'approximation.

Avec la notion de grand ordre, il est donc possible d'écrire les développements de Taylor de la façon suivante ($n = 2$ pour illustrer):

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + O((x - x_0)^3)$$

Formulation en h:

$$f(x_0 + h) = f(x_0) + f'(x_0)h + \frac{f''(x_0)}{2!}h^2 + O(h^3)$$

✓ La formulation en h est pratique pour évaluer facilement la fonction f en un point proche de x_0.

Voyons une application directe du développement de Taylor:
Soit x^* une approximation de x, f une fonction connue. Que peut-on dire de $|f(x) - f(x^*)|$?

→ On parlera de **propagation d'erreur**, où comment l'incertitude sur x se répercute sur le résultat d'un calcul.

Propagation d'erreur :
Supposons connu une certaine approximation x^* (un paramètre physique par exemple), avec Δx connu. On souhaite évaluer la précision du calcul $f(x^*)$.

L'erreur absolue est définie par

$$\Delta f = |f(x) - f(x^*)| = |f(x^* \pm \Delta x) - f(x^*)|$$

Mais par développement de Taylor de f en x^*, on a

$$f(x^* \pm \Delta x) = f(x^*) \pm f'(x^*)\Delta x + O((\Delta x)^2)$$

On a ainsi

$$\Delta f \approx |f'(x^*)|\Delta x$$

à partir de l'erreur absolue sur x^*, on peut donc estimer l'erreur absolue sur $f(x^*)$.

Le développement de Taylor se généralise pour les fonctions de plusieurs variables.

Soit $f(x, y)$ une fonction de deux variables, à valeurs réelles que l'on suppose suffisamment différentiable. Le développement de Taylor d'ordre 2 de la fonction $f(x, y)$ autour du point (x_0, y_0) est défini par

$$f(x_0 + h_1, y_0 + h_2) = f(x_0, y_0) + \frac{\partial f}{\partial x}(x_0, y_0)h_1 + \frac{\partial f}{\partial y}(x_0, y_0)h_2 + O(h_1^2) + O(h_2^2),$$

Pour une fonction de trois variables $f(x, y, z)$, on aura autour de (x_0, y_0, z_0)

$$f(x_0 + h_1, y_0 + h_2, z_0 + h_3) = f(x_0, y_0, z_0) + \frac{\partial f}{\partial x}(x_0, y_0, z_0)h_1 + \frac{\partial f}{\partial y}(x_0, y_0, z_0)h_2 + \frac{\partial f}{\partial z}(x_0, y_0, z_0)h_3 + O(h_1^2) + O(h_2^2) + O(h_3^2)$$

Dans le cadre de ce cours, on n'explicitera pas le terme d'erreur pour les fonctions de plusieurs variables.

Cela nous fournit une formule pour la propagation d'erreur dans le cadre de fonctions à plusieurs variables

$$(x, y, z) = (x^*, y^*, z^*) \pm (\Delta x, \Delta y, \Delta z)$$

$$\begin{aligned}\Delta f = |f(x, y, z) - f(x^*, y^*, z^*)| \simeq & \left|\frac{\partial f}{\partial x}(x^*, y^*, z^*)\right| \Delta x \\ & + \left|\frac{\partial f}{\partial y}(x^*, y^*, z^*)\right| \Delta y \\ & + \left|\frac{\partial f}{\partial z}(x^*, y^*, z^*)\right| \Delta z\end{aligned}$$

Propagation d'erreur absolue pour les opérations élémentaires:

$$\begin{aligned}f(x, y) = x + y \quad &\longrightarrow \quad \Delta f \simeq |\Delta x| + |\Delta y| \\ f(x, y) = x - y \quad &\longrightarrow \quad \Delta f \simeq |\Delta x| + |\Delta y| \\ f(x, y) = x \times y \quad &\longrightarrow \quad \Delta f \simeq |y^*||\Delta x| + |x^*||\Delta y| \\ f(x, y) = x \div y \quad &\longrightarrow \quad \Delta f \simeq \frac{|y^*||\Delta x| + |x^*||\Delta y|}{|y^*|^2}\end{aligned}$$

Je dois être capable de :

- Calculer/estimer l'erreur absolue et relative d'un nombre,
- Établir les chiffres significatifs partant de l'erreur absolue et réciproquement,
- Faire une représentation en virgule flottante avec mantisse à n chiffres par troncature ou par arrondi,
- Normaliser une mantisse et décaler la mantisse le cas échéant,
- Faire des opérations élémentaires en virgule flottante à n chiffres,
- Reconnaître les opérations dangereuses et comprendre l'importance de l'ordre dans les opérations élémentaires,
- Utiliser la méthode de Horner,
- Construire un polynôme de Taylor de degré n ainsi que son terme d'erreur,
- Estimer le reste, approximer l'erreur absolue et obtenir le nombre de chiffres significatifs de mon approximation,
- Comprendre et savoir calculer l'ordre d'une méthode,
- Savoir calculer la propagation d'erreur pour une fonction d'une ou de plusieurs variables et ainsi calculer l'erreur/chiffres significatifs d'une évaluation de fonction.

Chapitre 2: Équations non linéaires

Objectif du chapitre: Pour une fonction f donnée, on souhaite trouver r tel que

$$f(r) = 0$$

On dira qu'un tel r est **une racine** de la fonction f.

- On travaillera avec des fonctions f continues,
- Dans certains cas, il est facile de trouver analytiquement les racines. Pour des fonctions polynomiales, il existe par exemple des formules pour des degrés ≤ 4. Mais rien pour les degrés supérieurs et les fonctions quelconques,
- On cherche une méthode systématique, s'appuyant le moins possible sur les particularités de la fonction f.

Ce chapitre s'attache à construire des méthodes permettant de trouver les racines d'une fonction quelconque.

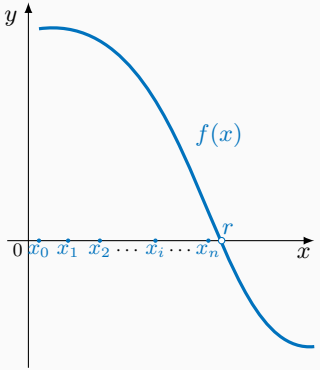
Un peu de jargon pour commencer...

Définition (Méthode itérative)

Une méthode itérative est une méthode dans laquelle on applique de manière répétitive (en boucle) certaines opérations algorithmiques. Une méthode itérative se compose en général de **donnée(s) initiale(s)** et de **critère(s) d'arrêt(s)**.

- Une méthode itérative procède par **itérations**, une itération correspondant à un passage dans la boucle,
- Le critère d'arrêt est une condition vérifiée à chaque itération permettant l'arrêt ou non de la méthode,
- Une méthode itérative va générer une suite d'approximation $x_0, x_1, ..., x_n$, on parle aussi d'itérés, on va avoir une approximation par itération,
- Une méthode itérative s'oppose à une méthode directe, résolvant le problème en une seule étape,

- Une méthode itérative est convergente si la suite d'approximation converge ($\lim_{n\to\infty} |x_n - r| = 0$),
- Une méthode itérative est divergente si la suite d'approximation ne converge pas (la limite est infinie ou n'existe pas, ex: $x_n = (-1)^n$).



Critère d'arrêt, comment le choisir ? Dans un monde idéal on arrêterait lorsque

$$|x_n - r| = 0.$$

La probabilité que cela arrive est voisine de 0, notamment à cause des erreurs de représentation, troncature,...

Quelques critères d'arrêts usuels:

- Critères de convergence : critères définissant une solution acceptable

$$|x_{n+1} - x_n| \leq TOL, \quad \left|\frac{x_{n+1} - x_n}{x_n}\right| \leq TOL, \quad |f(x_n)| \leq TOL$$

- Critères de divergence : critères définissant une instabilité, l'impossibilité d'atteindre une solution acceptable

$$\left|\frac{x_{n+1} - x_n}{x_n}\right| \geq TOL, \quad \left|\frac{x_{n+1} - x_n}{x_n - x_{n-1}}\right| \geq TOL, \quad n \geq n_{max}$$

  Les tolérances TOL et le nombre maximal d'itération n_{max} sont des **paramètres utilisateur**. Ils sont à ajuster suivant la précision souhaitée sur l'approximation.

Méthode de la bissection (méthode de dichotomie):

Repose sur l'observation suivante : de part et d'autre d'une racine, une fonction continue f va changer de signe et passer du positif au négatif ou vice-versa. Cela inspire la méthode suivante

- Choisir un intervalle de départ $[x_1, x_2]$ où la fonction f possède un changement de signe, c'est-à-dire où $f(x_1) \times f(x_2) < 0$
- Approximer la racine par le point milieu $x_m = \frac{x_1+x_2}{2}$,
- Choisir entre $[x_1, x_m]$ et $[x_m, x_2]$ l'intervalle qui possède encore un changement de signe. Recommencer la deuxième étape avec ce nouvel intervalle

L'idée est donc de réduire la longueur de l'intervalle contenant la racine par deux à chaque itération, d'où le nom de la méthode.

Algorithme de la bissection, pseudo-code `script_bissection.m`

Données : Une fonction f continue et un intervalle $[x_1, x_2]$ **contenant un changement de signe** de f, *i.e* $\boldsymbol{f(x_1) \times f(x_2) < 0}$. Une tolérance d'arrêt $\boldsymbol{\varepsilon_a}$, un nombre max d'itérations $\boldsymbol{n_{max}}$, la précision machine $\boldsymbol{\varepsilon_m}$.

Résultat : La variable x_m qui contiendra une approximation de la racine r.

$x_m = \frac{x_1+x_2}{2}$;
tant que $\frac{|x_2-x_1|}{2|x_m|+\varepsilon_m} > \varepsilon_a$ *et* $n \leq n_{max}$ **faire**
  si $f(x_1) \times f(x_m) < 0$ **alors**
    $x_2 = x_m$
  sinon
    si $f(x_m) \times f(x_2) < 0$ **alors**
      $x_1 = x_m$
    fin
  fin
  $x_m = \frac{x_1+x_2}{2}$; $n = n + 1$;
fin

• Pourquoi le critère d'arrêt suivant ?

$$\frac{|x_2 - x_1|}{2|x_m| + \varepsilon_m} \leq \varepsilon_a$$

→ À chaque étape, la racine r se trouve soit dans l'intervalle $[x_1, x_m]$ ou $[x_m, x_2]$, tous deux de longueur $\frac{x_2 - x_1}{2}$, et donc

$$\Delta r = |r - x_m| < \frac{x_2 - x_1}{2}$$

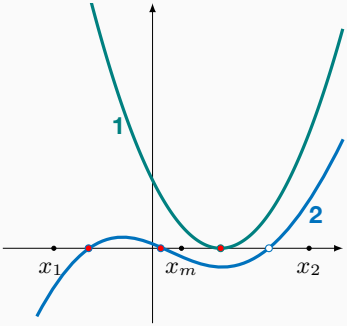
Autrement dit, $\frac{|x_2 - x_1|}{2|x_m|}$ est une approximation de l'erreur relative.

• Et pourquoi ajoute t'on ε_m, la précision machine, au dénominateur ?

→ Pour éviter la division par 0 ! En effet, il y a le risque que x_m devienne 0 au cours des itérations (0 machine).

Est-ce que cela fonctionne tout le temps ? **Non !** Soit $I = [x_1, x_2]$ l'intervalle de départ

1. Si la fonction $f(x)$ est tangente à l'axe des x, elle ne présente pas de changement de signe.
2. Si la fonction $f(x)$ admet un nombre pair (ou impair) de racine dans l'intervalle I, alors on va «rater» des racines.



→ Le choix de l'intervalle de départ est crucial !

Il est possible de connaître à l'avance le nombre d'itérations pour garantir que l'erreur absolue Δr soit inférieure à une quantité E donnée.

Soit $L = x_2 - x_1$, longueur de l'intervalle de départ.
Longueur de l'intervalle après une itération : $\frac{L}{2}$.
Longueur de l'intervalle après n itérations : $\frac{L}{2^n}$, et donc, après n itérations

$$\Delta r < \frac{L}{2^n}$$

Ainsi, pour garantir que l'erreur absolue soit inférieure à E donnée, il suffit de trouver $n \in \mathbb{N}$ tel que

$$\frac{L}{2^n} \leq E \Leftrightarrow \ln\left(\frac{L}{E}\right) \leq n\ln(2) \Leftrightarrow n \geq \frac{\ln(L) - \ln(E)}{\ln(2)}$$

n étant un entier on prend l'entier le plus petit vérifiant la dernière inégalité.

Remarque(s)

La majoration de l'erreur absolue dépend uniquement de l'itération et de la longueur de l'intervalle de départ, mais pas de la fonction f !

Avantage de la bissection :

- Fiable, converge dés qu'il y a changement de signe de la fonction.

Désavantages de la bissection :

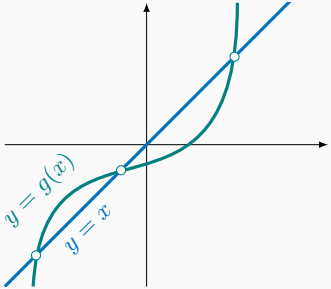
- Converge lentement,
- Exige une connaissance *a priori* de la position des racines.

Les méthodes que nous allons voir par la suite :

- Ne nécessiteront pas la connaissance d'un intervalle I où la fonction change de signe,
- Seront plus rapide,
- Mais la convergence ne sera plus assurée à 100%.

Définition (Point fixe)

Soit g une fonction à valeur réelle. Un point r tel que $\boldsymbol{g(r) = r}$ est appelé point fixe de la fonction g.



Graphiquement, un point fixe de g est un point où la fonction g intersecte la droite d'équation $y = x$.

Il y a un lien étroit entre racine et point fixe. En effet, si r est une racine de la fonction f, alors r est un point fixe des fonctions suivantes

$$g_{-}(x) = x - f(x) \rightarrow \; g_{-}(r) = r - f(r) = r$$
$$g_{+}(x) = x + f(x) \rightarrow \; g_{+}(r) = r + f(r) = r$$

Réciproquement, si r est un point fixe de g, alors

$$r - g(r) = 0,$$

r est donc une racine de la fonction $f(x) = x - g(x)$.

$\rightarrow$ **La recherche d'un point fixe est similaire à la recherche d'une racine.**

Le problème qui nous intéresse principalement est $f(r) = 0$. Mais partant de f, il est possible de construire une grande variété de fonctions de points fixes g.

La logique sera la suivante:

$f \rightarrow$ «fabrication» fonction point fixe $g \rightarrow$ application de la méthodes des points fixes à g pour trouver r tel que $g(r) = r \iff f(r) = 0$.

Voyons à présent en quoi consiste la méthode des points fixes.

Méthode du point fixe:

- Point initial x_0 donné,
- Faire pour $n \geq 0$: $\boldsymbol{x_{n+1} = g(x_n)}$.

Algorithme du point fixe, pseudo-code `script_points_fixes.m`

Données :

- Une fonction g continue et un point de départ x_0
- Une tolérance d'arrêt ε_a et un nombre max d'itérations n_{max}
- la précision machine ε_m.

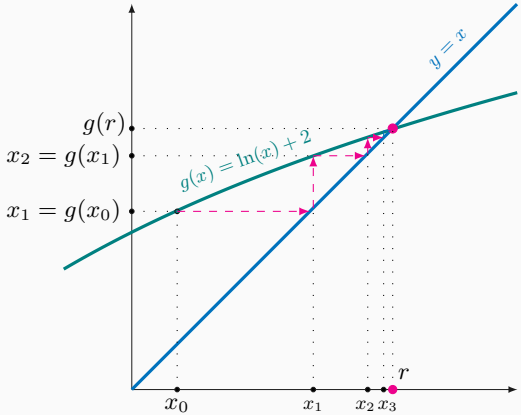
Résultat : la variable x_n qui contiendra une approximation du point fixe r

1 **tant que** $\frac{|x_{n+1}-x_n|}{|x_{n+1}|+\varepsilon_m} > \varepsilon_a$ *et* $n \leq n_{max}$ **faire**
2 $\quad x_{n+1} = g(x_n)$;
3 $\quad n = n + 1$;
4 **fin**

Méthode du point fixe:

- Point initial x_0 donné,
- Faire pour $n \geq 0$: $\boldsymbol{x_{n+1} = g(x_n)}$.

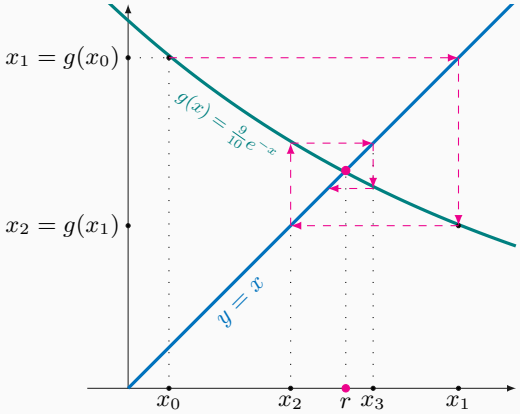
Exemple 1: $g(x) = \ln(x) + 2$



Méthode du point fixe:

- Point initial x_0 donné,
- Faire pour $n \geq 0$: $\boldsymbol{x_{n+1} = g(x_n)}$.

Exemple 2: $g(x) = \frac{9}{10}e^{-x}$



Quand la méthode fonctionne-t-elle ? *i.e* sous quelles conditions a t-on

$$x_n \to\ r = g(r)$$

Soit r un point fixe de g. L'erreur à l'itération n est définie par

$$e_n = x_n - r$$

Et dire que $x_n \to\ r = g(r)$ revient à dire que $e_n \to\ 0$.

Pour que cela se produise, l'erreur doit donc diminuer d'itérations en itérations. Or l'erreur à l'itération $n+1$ est définie par

$$e_{n+1} = x_{n+1} - r = g(x_n) - r = g(x_n) - g(r)$$

Mais on peut également écrire $x_n = r + \underbrace{x_n - r}_{e_n} = r + e_n$, et on a ainsi

$$e_{n+1} = g(r + e_n) - g(r)$$

Effectuons un développement de Taylor de la fonction g autour de r (pour e_n petit).

$$g(r+e_n) = g(r) + g'(r)e_n + \frac{g''(r)}{2}e_n^2 + \frac{g'''(r)}{6}e_n^3 + ...$$

Ainsi,

$$e_{n+1} = g(r+e_n) - g(r) = g'(r)e_n + \frac{g''(r)}{2}e_n^2 + \frac{g'''(r)}{6}e_n^3 + ...$$

Si l'on néglige les termes de degré supérieur à 2 en e_n, on obtient

$$\boldsymbol{e_{n+1} \approx g'(r)e_n} \approx g'(r)^2 e_{n-1} \approx ... \approx g'(r)^{n+1} e_0 = g'(r)^{n+1}(x_0 - r)$$

L'erreur à l'itération $n+1$ ne pourra donc diminuer, en valeur absolue, que si

$$\boldsymbol{-1 < g'(r) < 1 \iff |g'(r)| < 1}$$

En pratique:

- Si $|g'(r)| < 1$ et $g'(r) \neq 0$, la méthode converge vers r, r est dit **point fixe attractif**,
- Si $-1 < g'(r) < 0$, le signe de l'erreur alternera, les valeurs de x_n oscilleront de part et d'autre de r. Mais l'erreur diminue, et donc la méthode converge vers r,
- Si $|g'(r)| > 1$, l'erreur augmente à chaque itération, la méthode diverge, r est dit **point fixe répulsif**,
- Si $|g'(r)| = 1$, c'est un **cas indéterminé**, la convergence dépendra de x_0 et de g.

☢ Attention !

La convergence est également assujettie au choix de la valeur initiale x_0. Un mauvais choix de x_0 peut résulter en un algorithme divergent même si $|g'(r)| < 1$.

$\rightarrow$ L'ensemble des valeurs x_0 pour lesquelles x_n tend vers r est appelé **bassin d'attraction**.

Plus $g'(r)$ est petit plus l'erreur diminue vite, et plus la convergence est rapide. On dit que $|g'(r)|$ est le **taux de convergence** de la méthode des points fixes.

Le cas limite est $g'(r) = 0$. Que se passe-t-il dans ce cas ?
Reprenons notre développement de l'erreur à l'itération $n+1$

$$e_{n+1} = \cancel{g'(r)e_n} + \frac{g''(r)}{2}e_n^2 + \frac{g'''(r)}{6}e_n^3 + ...$$

Ainsi l'erreur à l'itération $n+1$ est cette fois proportionnelle à e_n^2. Dans ce cas la convergence dépendra du point de départ x_0. En effet

$$e_{n+1} \approx \frac{g''(r)}{2}e_n^2 \approx \frac{g''(r)^3}{8}e_{n-1}^4 \approx ... \approx e_0 \left(\frac{g''(r)}{2}e_0\right)^{2^{n+1}-1}$$

De même, si $g''(r) = 0$, on aura $e_{n+1} \approx \frac{g'''(r)}{6}e_n^3$, et la convergence dépendra encore du point de départ x_0.

De manière générale, on introduit la notion de convergence à l'ordre p.

Définition (Ordre de convergence)

La méthode des points fixes converge à l'ordre p si

$$|e_{n+1}| \approx C\,|e_n|^p$$

où C est une constante. Une convergence **d'ordre 1 est dite linéaire, d'ordre 2 quadratique, d'ordre 3 cubique**,...

En résumé:

- Si $|g'(r)| < 1$ et $g'(r) \neq 0$, la méthode des points fixes **converge à l'ordre 1 (linéaire)**,
- Si $|g'(r)| = 0$ la convergence est **au moins d'ordre 2 (quadratique) mais dépend de x_0**. L'ordre sera déterminé par l'ordre de la dérivée de g ne s'annulant pas (ordre 2 si $g''(r) \neq 0$, ...).

À noter que l'on souhaite en général l'ordre de convergence le plus élevé possible, pourquoi ?
Supposons $|e_0| = |x_0 - r| = 0.1$, alors on a pour une convergence linéaire

$$|e_1| = |x_1 - r| \approx C_{lin}|x_0 - r| = 0.1 \times C_{lin}$$

et pour une convergence quadratique

$$|e_1| = |x_1 - r| \approx C_{quad}|x_0 - r|^2 = 0.01 \times C_{quad}$$

$\rightarrow$ Comment mesurer l'ordre de convergence en pratique ?

Si l'on connaît r, on peut calculer e_{n+1} et e_n, et l'on doit avoir :

- Pour une convergence linéaire (ordre 1) :

$$\left|\frac{e_{n+1}}{e_n}\right| \rightarrow |g'(r)| \neq 0$$

- Pour une convergence quadratique (ordre 2) :

$$\left|\frac{e_{n+1}}{e_n}\right| \rightarrow |g'(r)| = 0 \text{ et } \left|\frac{e_{n+1}}{e_n^2}\right| \rightarrow \frac{|g''(r)|}{2} \neq 0$$

Comme en pratique r n'est pas connu, on utilise des approximations pour e_n. Deux méthodes :

- Si la méthode converge, on peut remplacer r par l'approximation obtenue à la dernière itération N de l'algorithme.

  $$x_N \approx r \text{ et donc } |e_n| \approx |x_n - x_N| = E_n$$

  Le calcul de l'erreur est donc fait *a posteriori*, une fois la valeur de x_N obtenue.

- Si l'on souhaite une approximation de l'erreur pendant les itérations de l'algorithme, on peut prendre l'approximation suivante, de moins bonne qualité

  $$E_n = x_{n+1} - x_n \approx e_n = x_n - r$$

Dans les deux cas, pour établir l'ordre de convergence, on regardera

$$\left|\frac{E_{n+1}}{E_n}\right| \to |g'(r)| \neq 0 \Rightarrow \text{ convergence linéaire}$$

$$\left|\frac{E_{n+1}}{E_n}\right| \to |g'(r)| = 0, \left|\frac{E_{n+1}}{E_n^2}\right| \to \frac{|g''(r)|}{2} \neq 0 \Rightarrow \text{ convergence quadratique}$$

Méthode des points fixes appliquée à la fonction $g(x) = e^{-x}$, $x_0 = 0$			
n	x_n	$\lvert E_n \rvert = \lvert x_{n+1} - x_n \rvert$	$\left\lvert \frac{E_{n+1}}{E_n} \right\rvert$
0	$0,0000000$	$1,000\ 0000$	$0,6321 \times 10^{+0}$
1	$1,0000000$	$0,6321 \times 10^{+0}$	$0,5131$
2	$0,3678794$	$0,3243 \times 10^{+0}$	$0,5912$
3	$0,6922006$	$0,1917 \times 10^{+0}$	$0,5517$
4	$0,5004735$	$0,1058 \times 10^{+0}$	$0,5753$
5	$0,6062435$	$0,6085 \times 10^{-1}$	$0,5623$
6	$0,5453957$	$0,3422 \times 10^{-1}$	$0,5698$
7	$0,5796123$	$0,1950 \times 10^{-1}$	$-$
⋮	⋮	⋮	⋮
14	$0,5669089$	$0,3673 \times 10^{-3}$	$0,5672$
15	$0,5672762$	$0,2083 \times 10^{-3}$	$0,5671$
⋮	⋮	⋮	⋮
35	$0,5671433$	$0,2469 \times 10^{-8}$	$0,5671$

$\rightarrow$ **Convergence à l'ordre 1 (linéaire)**, $|g'(r)| \approx 0.5671$

À partir d'une méthode de point fixe convergeant à l'ordre 1, est-il possible d'obtenir une méthode convergeant à l'ordre 2 ? → **Algorithme de Steffenson**

Part de l'observation suivante

$$e_{n+1} \approx g'(r)e_n \text{ et } e_{n+2} \approx g'(r)e_{n+1} \Rightarrow \frac{e_{n+2}}{e_{n+1}} \approx \frac{e_{n+1}}{e_n}$$

c'est-à-dire

$$\frac{x_{n+2} - r}{x_{n+1} - r} \approx \frac{x_{n+1} - r}{x_n - r} \tag{1}$$

En isolant r dans (1) on obtient

$$r \approx \frac{x_{n+2}x_n - x_{n+1}^2}{x_{n+2} - 2x_{n+1} + x_n}$$

Cette formule étant numériquement instable, on lui préférera

$$r \approx x_n - \frac{(x_{n+1} - x_n)^2}{x_{n+2} - 2x_{n+1} + x_n}$$

Cette dernière formule s'appelle **formule d'extrapolation de Aitken**. L'extrapolation de Aitken s'applique à n'importe quelle suite $(x_n)_{n\in\mathbb{N}}$ de réels, et permet d'en accélérer la convergence.

L'extrapolation de Aitken couplé à la méthode des points fixes nous donne **l'algorithme de Steffenson :**

- Point initial x_0 donné,
- Faire pour $n \geq 0$: $x_{n+1} = x_n - \frac{(g(x_n)-x_n)^2}{g(g(x_n))-2g(x_n)+x_n}$.

Algorithme de Steffenson, pseudo-code `script_Steffenson.m`

Données :

- Une fonction g continue et un point de départ x_0
- Une tolérance d'arrêt ε_a, un nombre max d'itérations n_{max} et la précision machine ε_m.

Résultat : La variable x_{n+1} qui contiendra une approximation du point fixe r

1 **tant que** $\frac{|x_{n+1}-x_n|}{|x_{n+1}|+\varepsilon_m} > \varepsilon_a$ *et* $n \leq n_{max}$ **faire**
2 $\quad a = g(x_n)$; $b = g(a)$;
3 $\quad x_{n+1} = x_n - \frac{(a-x_n)^2}{b-2a+x_n}$;
4 $\quad n = n + 1$;
5 **fin**

- L'algorithme de Steffenson revient à appliquer la méthode de points fixes à la fonction h définie par

$$h(x) = x - \frac{(g(x) - x)^2}{g(g(x)) - 2g(x) + x}$$

- Dans l'algorithme précédent, les variables a et b peuvent être vues comme des variables temporaires, écrasées à chaque itération,
- Si la méthode des points fixes est d'ordre 1 pour g, alors la méthode de Steffenson sera d'ordre 2. Si la convergence pour g est déjà d'ordre 2, alors on peut avoir avoir convergence d'ordre 3 !
- La méthode de Steffenson est un peu plus sensible aux instabilités numériques (soustractions dangereuses),
- La convergence va toujours dépendre du point de départ x_0.

$\rightarrow$ Est-il possible, sans accélérer la suite $(x_n)_{n\in\mathbb{N}}$, d'avoir une méthode d'ordre 2 ?

La **méthode de Newton** est une méthode **quadratique**, que l'on peut interpréter de différentes façons.

Première interprétation : Approche par correction (développement de Taylor)

Rappel: on cherche à résoudre $f(x) = 0$,

- On choisit un point de départ x_0
- On cherche une correction δ_0 sur le point x_0 telle que:

$$f(x_0 + \delta_0) = 0.$$

- A l'aide du développement de Taylor de f en x_0:

$$0 = f(x_0 + \delta_0) = f(x_0) + f'(x_0)\delta_0 + O(\delta_0^2),$$

en supposant δ_0 petit, on néglige $O(\delta_0^2)$, et en supposant $f'(x_0) \neq 0$, on obtient la correction

$$\delta_0 = -\frac{f(x_0)}{f'(x_0)}$$

- Prendre $x_1 = x_0 + \delta_0 = x_0 - \frac{f(x_0)}{f'(x_0)}$ et recommencer à partir de x_1.

Deuxième interprétation : Point fixe

On introduit la fonction:

$$g(x) = x - \frac{f(x)}{f'(x)}.$$

Alors $g(x) = x$ implique $f(x) = 0$.

$\rightarrow$ La méthode de Newton est donc une **méthode de point fixe avec une fonction g particulière**.

On peut lui appliquer les résultats vus pour les points fixes:

- Convergence quadratique si $f'(r) \neq 0$ car:

$$g'(r) = \frac{f(r)f''(r)}{f'(r)^2} = 0$$

- Approximation du terme d'erreur (développement de Taylor) :

$$e_{n+1} \approx \frac{g''(r)}{2} e_n^2 = \frac{f''(r)}{2f'(r)} e_n^2$$

Remarque(s)

Si $f''(r) = 0$, la convergence est au moins cubique.

Algorithme de Newton :

- Point x_0 initial donné.
- Faire $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$.

Algorithme de Newton, pseudo-code `script_Newton.m`

Données :

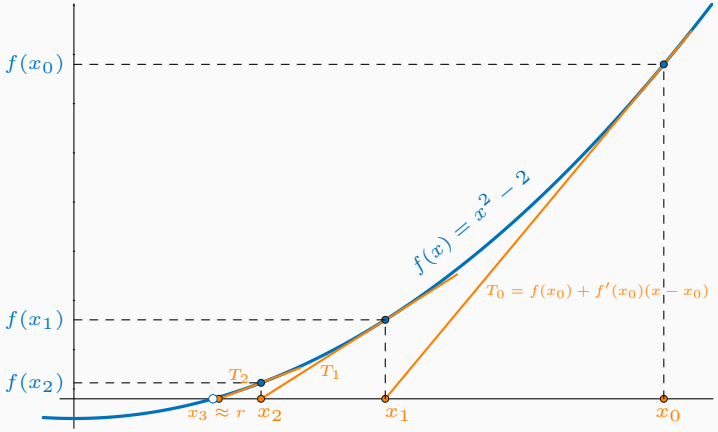
- Une fonction f deux fois différentiables et un point de départ x_0.
- Une tolérance d'arrêt ε_a et un nombre max d'itérations n_{max}
- la précision machine ε_m.

Résultat : la variable x_{n+1} qui contiendra une approximation de la racine r

1 **tant que** $\frac{|x_{n+1}-x_n|}{|x_{n+1}|+\varepsilon_m} > \varepsilon_a$ *et* $n \leq n_{max}$ **faire**
2 $\quad x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$;
3 $\quad n = n + 1$;
4 **fin**

Interprétation géométrique : $f(x_0 + \delta_0) = f(x_0) + f'(x_0)\delta_0 \rightarrow$ équation de la tangente à f en x_0.

- Partir d'un point $(x_0, f(x_0))$,
- Trouver l'intersection de la tangente en $(x_0, f(x_0))$ avec l'axe des abscisses $\rightarrow x_1$,
- Boucler en utilisant le nouveau point $(x_1, f(x_1))$.



En résumé, si f est deux fois dérivable avec $f'(r) \neq 0$ et si x_0 est assez près de la racine r, la méthode de Newton sera bien définie et **dans le pire des cas, convergente à l'ordre 2**.

Que se passe-t-il quand $f'(r) = 0$? $\rightarrow$ Souvent le signe d'une racine multiple

Définition (Multiplicité d'une racine)

Une racine r de f est de multiplicité m si on peut écrire f sous la forme:

$$f(x) = (x - r)^m h(x)$$

avec $\lim_{x \to r} h(x) = h(r) \neq 0$

Proposition (Caractérisation d'une racine multiple)

Une racine r de f est de multiplicité m si et seulement si:

$$f(r) = f'(r) = \cdots = f^{(m-1)}(r) = 0 \text{ et } f^{(m)} \neq 0.$$

Si r est une racine de multiplicité m la convergence devient linéaire $(g'(r) \neq 0)$.

En effet, partant de

$$g'(r) = \frac{f(r)f''(r)}{f'(r)^2} \tag{2}$$

et en remplaçant f par $f(x) = (x-r)^m h(x)$ dans (2), on obtient pour g' (cf. p.76 du manuel pour développements complets)

$$\boldsymbol{g'(r) = \frac{h(r)^2 m(m-1)}{m^2 h(r)^2} = \frac{m(m-1)}{m^2} = 1 - \frac{1}{m}}$$

- Pour $m = 1$, la racine est simple et on retrouve le résultat attendu, c'est-à-dire $g'(r) = 0$ et donc convergence quadratique,
- Pour $m \geq 2$, on a convergence linéaire,

$$\left|\frac{e_{n+1}}{e_n}\right| \to 1 - \frac{1}{m}, \quad \left|\frac{e_{n+1}}{e_n^2}\right| \to \infty$$

  Le taux de convergence est $1 - \frac{1}{m}$, plus m sera grand, plus la convergence sera lente.

Pour résumer :

- $f'(r) \neq 0$: racine simple, convergence au moins quadratique,
- $f'(r) = 0$: racine multiple, convergence linéaire.

Comment retrouver une convergence dans le cas d'une racine multiple ?

Pour retrouver une convergence quadratique, on peut appliquer la méthode de Newton à:

$$h(x) = \frac{f(x)}{f'(x)}$$

Il est possible de montrer (p.77 du manuel) que la racine multiple r de f est une racine simple de la fonction h.
L'algorithme de Newton devient alors

$$x_{n+1} = x_n - \frac{f(x_n)f'(x_n)}{(f'(x_n))^2 - f(x_n)f''(x_n)}$$

$\rightarrow$ nécessite le calcul d'une dérivée en plus !

Autres approches dans le cas de racines multiples :

- Si l'on connait la valeur de la multiplicité m (très rare en pratique), en peut prendre (ex. 12 p.89)

$$g(x) = x - m\frac{f(x)}{f'(x)},$$

- Appliquer l'extrapolation de Aitken à Newton.

Remarque(s)

Le point de départ x_0 peut être loin de r et l'algorithme tout de même converger (avec un taux moindre que quadratique au début). En ce sens, la méthode de Newton est une méthode relativement robuste.

Conclusions:

- Méthode quadratique si $f'(r) \neq 0$ et f deux fois dérivables,
- Nécessite la dérivabilité de f et le calcul de f',
- La nature de f peut ralentir ou empêcher la convergence. En présence d'une racine multiple, la convergence devient linéaire. Les méthodes pour retrouver un taux quadratique peuvent nécessiter :
  - la connaissance de la multiplicité,
  - des calculs supplémentaires,
  - une plus grande régularité de la fonction.

L'algorithme de Newton nécessite d'être en mesure de calculer la dérivée de la fonction f

$\rightarrow$ Que faire si l'on ne connaît pas la dérivée f' ?

On introduit la méthode de la sécante, apparentée à la méthode de Newton, qui s'affranchit du calcul de dérivée. C'est une méthode à deux points.

On calcule la valeur du point x_{n+1}:

$$x_{n+1} = x_n - \frac{f(x_n)(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}$$

Cela revient à changer dans la méthode de Newton $f'(x_n)$ par $\frac{f(x_n)-f(x_{n-1})}{x_n - x_{n-1}}$, coefficient directeur de la sécante entre $(x_n, f(x_n))$ et $(x_{n-1}, f(x_{n-1}))$

- Ce n'est **pas une méthode de point fixe**.
- C'est une méthode à **deux pas**.
- Les deux points x_0 et x_1 pour l'initialisation devront être proches de r.
- **La convergence n'est pas quadratique**. Si $f'(r) \neq 0$ et $f''(r) \neq 0$, l'ordre de la méthode est de $\frac{1+\sqrt{5}}{2} \rightarrow$ **méthode dite superlinéaire**.

$$\left|\frac{e_{n+1}}{e_n}\right| \rightarrow 0, \quad \left|\frac{e_{n+1}}{e_n^2}\right| \rightarrow \infty$$

Algorithme de la sécante

- Points x_0 et x_1 initiaux donnés.
- Faire $x_{n+1} = x_n - \frac{f(x_n)(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}$

Algorithme de la sécante, pseudo-code `script_Secante.m`

Données :

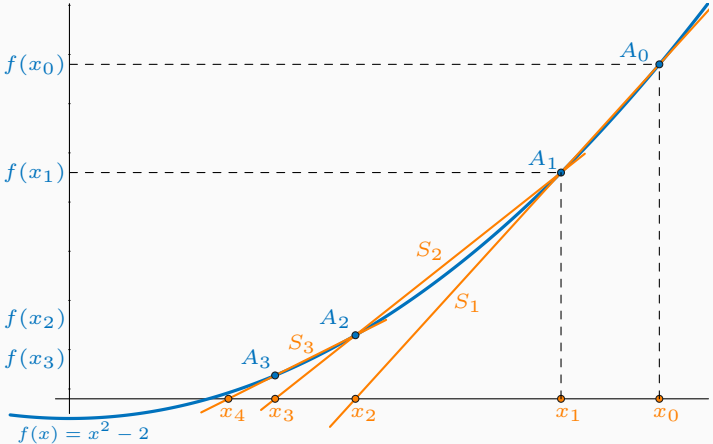
- Une fonction f continue et deux points de départ x_0, x_1.
- Une tolérance d'arrêt ε_a et un nombre max d'itérations n_{max}
- la précision machine ε_m.

Résultat : la variable x_{n+1} qui contiendra une approximation de la racine r

1 **tant que** $\frac{|x_{n+1} - x_n|}{|x_{n+1}| + \varepsilon_m} > \varepsilon_a$ *et* $n \leq n_{max}$ **faire**
2 $\quad x_{n+1} = x_n - \frac{f(x_n)(x_n - x_{n-1})}{(f(x_n) - f(x_{n-1})}$,
3 $\quad n = n + 1$;
4 **fin**

Interprétation géométrique :

- Partir de deux points $A_0 = (x_0, f(x_0))$ et $A_1 = (x_1, f(x_1))$,
- Trouver l'intersection de (A_0A_1) et l'axe des abscisses $\rightarrow x_2$,
- Boucler en utilisant les points $(x_2, f(x_2))$ et $(x_1, f(x_1))$.



Conclusion chapitre 2

Je dois être capable de :

- Maîtriser la bissection et son terme d'erreur,
- Distinguer une racine et un point fixe,
- Construire la fonction $g(x)$ dont le point fixe correspond à la racine de l'équation $f(x) = 0$,
- Caractériser un point fixe (répulsif, $\cdots$) et je comprends ce qu'est un bassin d'attraction,
- Établir la convergence ou non d'une méthode de point fixe,
- Comprendre et je sais déterminer l'ordre d'une méthode de point fixe,
- Utiliser l'extrapolation de Aitken,
- Connaître les particularités que constituent les méthodes de:
  - Steffenson (construction, utilisation et ordre de convergence à connaitre),
  - Newton (utilisation et ordre de la méthode à connaitre),
  - La sécante (utilisation et ordre de la méthode à connaitre),
- Saisir la notion de racine multiple et son impact sur la méthode de Newton,
- Comprendre que toutes ces méthodes sont sensibles à la position du point de départ x_0,
- Connaître une ou deux méthodes corrigeant la perte d'ordre dans le cas de Newton,
- Analyser le comportement d'une méthode en me basant sur un tableau de résultat: comportement normal ou anomalie imprévue, convergence, ordre.

Chapitre 3: Systèmes d’équations algébriques

De nombreuses méthodes numériques conduisent à la résolution d'un système matriciel de la forme

$$Ax = b.$$

L'inconnue est le vecteur x, tandis que la matrice A et le second membre b sont connus.

Le but de ce chapitre est de voir des méthodes de résolution «automatique» des systèmes, qui soient le plus efficace possible.

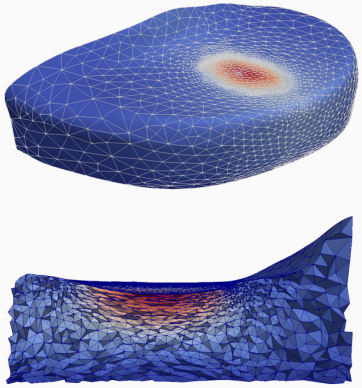
Aujourd'hui, la taille du vecteur x, représentant les inconnues, dépasse souvent les milliers, et peut même dépasser le milliard d'inconnues[6] !

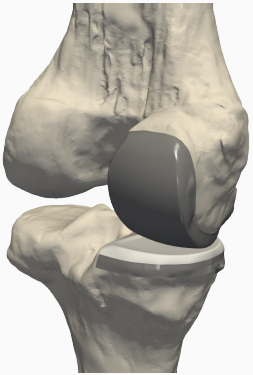
$\rightarrow$ Que peut représenter ce vecteur x en pratique ?

[6] **Calcul à 4.6 milliards d'inconnues**

Exemples de simulations numériques de la «vraie vie» :

1) Simulation du contact prothèse-fémur [7]

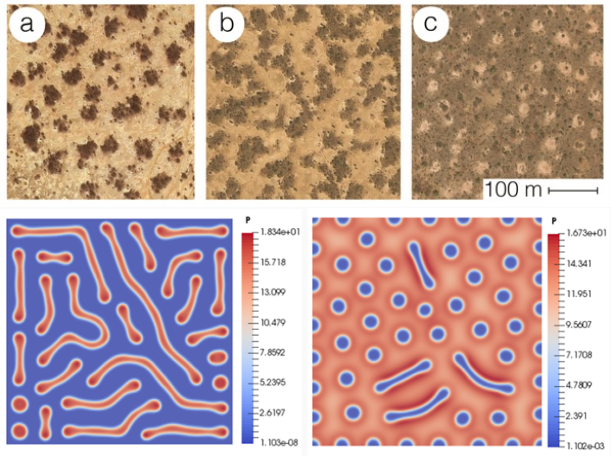




Le vecteur x peut représenter une variable physique (déplacement, pression,...) calculée à chaque sommet d'un maillage.

[7] Prothèse de genou, Bodycad

2) Croissance de plante dans un milieu semi-aride. Le vecteur x peut par exemple représenter la densité de plante présente, calculée sur une grille fine [8]



[8] Aerial images of flat terrain vegetation patterns in Sudan. 1a. Spot (11.6280, 27.9177). 1b. Labyrinth (11.1024, 27.8228). 1c. Gap patterns (10.7549, 28.5955). Images © Google, DigitalGlobe. Image credit: Karna Gowda.

Moral:

- De nombreuses méthodes numériques conduisent à la construction d'un système matriciel, souvent de grande taille.
- Nécessité de se munir de méthodes de résolutions efficaces.

Plan de bataille du chapitre:

- Étude des systèmes dit linéaires (élimination de Gauss, factorisation LU, Cholesky)
- Normes et conditionnement de matrices,
- Généralisation de la méthode de Newton pour résoudre les systèmes d'équations non linéaires.

Pour les systèmes linéaires, nous allons uniquement étudier les **méthodes dites directes**, i.e donnant la solution en un nombre fini et prédéterminé d'opérations.

Des **méthodes itératives**, nécessitant en théorie un nombre infinie d'opérations, existent également pour résoudre ces systèmes.

Pour les systèmes d'équations non-linéaires, l'utilisation de méthodes itératives devient obligatoire.

De manière générale, un système de m **équations linéaires** à n inconnues, est un système pouvant être écrit sous la forme suivante :

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \cdots + a_{3n}x_n = b_3 \\ \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\ a_{m1}x_1 + a_{m2}x_2 + a_{m3}x_3 + \cdots + a_{mn}x_n = b_m \end{cases} \tag{3}$$

où $\boldsymbol{x_1, x_2, \cdots, x_n}$ sont **les inconnues** et les $\boldsymbol{a_{ij}}$ **les coefficients** du système.

Le système (3), où $m = n$, s'écrit sous la forme matricielle $Ax = b$

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}}_{b}$$

En pratique, la matrice A et le second membre b seront connus, il s'agira de déterminer le vecteur $x = (x_1, x_2, \cdots, x_n)$.

Pour le cours, **on aura toujours $n = m$**, donc autant d'inconnues que d'équations.

On étudiera donc les systèmes où la matrice A est carrée.

Rappelons quelles conditions garantissent l'existence d'une solution au système.

Rappel

Les propositions équivalentes suivantes garantissent l'existence d'une solution au système matriciel $Ax = b$ (avec A matrice carrée)

- La matrice A est inversible, c-à-d A^{-1} existe ($AA^{-1} = A^{-1}A = I$),
- Le déterminant de la matrice A, noté $\det(A)$ est non nul,
- La matrice A est de rang maximal (nombre d'inconnues=nombre d'équations),
- Le système $Ax = 0$ admet seulement la solution nulle.

Dans ce chapitre, on traitera uniquement le cas où l'une des propositions précédente est vérifiée. On considérera donc toujours le cas où A est inversible (non singulière), on écrira $x = A^{-1}b$.

✓ En pratique, on ne calculera jamais explicitement A^{-1}. Calculer l'inverse d'une matrice n'est pas numériquement efficace ! On va plutôt travailler sur le système linéaire de départ afin de déterminer x.

Comment résoudre de **manière systématique** un système linéaire ?

Regardons d'abord un cas particulier de matrices, les matrices diagonales.

Définition (Matrice diagonale)

La matrice $A = (a_{ij})_{1 \leq i,j \leq n}$ est dite **diagonale**, si ses entrées sont nulles en dehors de sa diagonale (i.e. si $i \neq j$ alors $a_{ij} = 0$).
Une telle matrice est donc de la forme

$$A = \begin{pmatrix} a_{11} & 0 & \dots & \dots & 0 \\ 0 & a_{22} & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \dots & \dots & 0 & a_{nn} \end{pmatrix}$$

On a alors, $\det(A) = a_{11} \times a_{22} \times ... \times a_{nn} = \Pi_{i=1}^{n} a_{ii}$

Dans ce cas, la résolution du système $Ax = b$ est immédiate,

$$x_i = \frac{b_i}{a_{ii}}, \ \ i = 1, \cdots, n$$

Définition (Matrice triangulaire)

La matrice $A = (a_{ij})_{1 \leq i,j \leq n}$ est dite **triangulaire inférieure** si tous les a_{ij} sont nuls pour $i < j$. Une matrice triangulaire inférieure est donc de la forme

$$A = \begin{pmatrix} a_{11} & 0 & 0 & 0 & \dots & 0 \\ a_{21} & a_{22} & 0 & 0 & \dots & 0 \\ a_{31} & a_{32} & a_{33} & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-1\,1} & a_{n_1\,2} & a_{n-1\,3} & \dots & a_{n-1\,n-1} & 0 \\ a_{n\,1} & a_{n\,2} & a_{n\,3} & \dots & a_{n\,n-1} & a_{n\,n} \end{pmatrix}$$

Une matrice est dite **triangulaire supérieure** si tous les a_{ij} sont nuls pour $j < i$ (transposée d'une matrice triangulaire inférieure). Dans les deux cas, $\det(A) = a_{11} \times a_{22} \times ... \times a_{nn} = \Pi_{i=1}^{n} a_{ii}$

La résolution du système $Ax = b$ avec A triangulaire inférieure ou supérieure est également aisée. On effectue une **descente triangulaire** ou **remontée triangulaire**, selon le cas.

- A triangulaire inférieure $\rightarrow$ descente triangulaire

$$x_1 = \frac{b_1}{a_{11}}, \quad x_i = \frac{\left(b_i - \sum_{k=1}^{i-1} a_{ik}x_k\right)}{a_{ii}}, \ i = 2, 3, \cdots, n. \tag{4}$$

- A triangulaire supérieure $\rightarrow$ remontée triangulaire

$$x_n = \frac{b_n}{a_{nn}}, \quad x_i = \frac{\left(b_i - \sum_{k=i+1}^{n} a_{ik}x_k\right)}{a_{ii}}, \ i = n-1, n-2, \cdots, 2, 1. \tag{5}$$

☣ Attention !

Les formules (4) et (5) sont valides si les a_{ii} sont tous non nuls. Dans le cas contraire, cela signifie que $\det(A) = 0$ est donc que la matrice n'est pas inversible. Le système $Ax = b$ n'admet alors pas de solution unique.

Combien coûte (en nombre d'opérations) une descente (resp. remontée) triangulaire ?

- $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ multiplications/divisions $\rightarrow\ O(n^2)$
- $\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$ additions/soustractions $\rightarrow\ O(n^2)$

Les méthodes de descentes et remontées triangulaires sont très génériques, mais nécessitent que A soit triangulaire.

L'idée va donc être de transformer une matrice carrée quelconque en matrice triangulaire.

$\rightarrow$ **Élimination de Gauss**

L'élimination de Gauss consiste à effectuer des opérations élémentaires sur les lignes de la matrice A.

But: Introduire des zéros sous la diagonale de la matrice A afin de la rendre triangulaire supérieure.

On note l_i la ligne i de la matrice A. Les trois opérations élémentaires sont les suivantes :

- $(l_i \leftarrow \lambda l_i)$: remplacer la ligne i par un multiple d'elle-même,
- $(l_i \leftrightarrow l_j)$: intervertir la ligne i et j,
- $(l_i \leftarrow l_i + \lambda l_j)$: remplacer la ligne i par la ligne i plus un multiple de la ligne j.

Chacune de ces opérations revient à multiplier la matrice A par une matrice inversible W.

Remarque(s)

En multipliant par une telle matrice inversible W, la solution du système de départ n'est pas modifiée

$$WAx = Wb \iff W^{-1}WAx = W^{-1}Wb \iff Ax = b$$

- $(l_i \leftarrow \lambda l_i)$: W est une matrice ressemblant à la matrice identité, sauf pour le coefficient W_{ii} qui vaut λ,
- $(l_i \leftrightarrow l_j)$: W est une matrice où la ligne i et la ligne j de la matrice identité sont permutées,
- $(l_i \leftarrow l_i + \lambda l_j)$: W est une matrice ressemblant à la matrice identité, sauf pour le coefficient W_{ij} qui vaut λ.

En pratique, dans l'élimination de Gauss, on ne multiplie pas par les matrices W, on fait plutôt les opérations élémentaires sur la matrice dite augmentée, *i.e* la matrice que l'on obtient en ajoutant le membre de droite b à la matrice A, c'est-à-dire:

$$\left(\begin{array}{cccc|c} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{array}\right)$$

Les opérations élémentaires sont faites **à la fois sur la matrice A et le second membre b**.

Une fois la matrice A rendu triangulaire supérieure, il ne reste plus qu'à effectuer une remontée triangulaire afin d'obtenir la solution x.

Avantage : permet de traiter des matrices quelconques,

Inconvénient : **si l'on change le second membre b, on doit tout recommencer !**

Dans l'élimination de Gauss, on effectue les opérations directement sur A et b, les matrices des opérations élémentaires ne sont pas conservées.

$\rightarrow$ On «jette» de l'information en effectuant l'élimination de Gauss !

La solution va être de conserver ces matrices.

Soit $T = W_N W_{N-1} \cdots W_1$ la matrice représentant la suite d'opérations élémentaires rendant la matrice A triangulaire supérieure. Cette matrice ne dépend pas de b et l'on a

$$TAx = Tb$$

Posons $U = TA$, U est la matrice triangulaire supérieure, et

$$T^{-1}Ux = T^{-1}Tb = b$$

Autrement dit, $T^{-1}U = A$, et posons

$$L = T^{-1} = (W_N W_{N-1} \cdots W_1)^{-1} = W_1^{-1} W_2^{-1} \cdots W_N^{-1}$$

L est la matrice triangulaire inférieure qui a permis d'éliminer les termes non nuls sous la diagonale de la matrice A dans l'élimination de Gauss.

On a donc la décomposition $A = LU$, cependant **cette décomposition n'est pas unique**. Elle le sera en ajoutant des contraintes supplémentaires sur L ou U.

Théorème (Décomposition LU)

Si $A \in M_n(\mathbb{R})$, telle que les sous matrices d'ordre $1 \leq k \leq n$ (sous matrices principales):

$$A^k = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1k} \\ a_{21} & a_{22} & \dots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k1} & a_{k2} & \dots & a_{kk} \end{pmatrix}$$

soient inversibles ($\det(A^k) \neq 0$, $1 \leq k \leq n$). Alors il existe une matrice triangulaire supérieure U («Upper»), et une matrice triangulaire inférieure L («Lower»), telle que

$$A = LU$$

Si l'on fixe la diagonale de L (ou de U) avec des 1, la décomposition est unique.

On a donc le choix dans la matrice où l'on place les 1 sur la diagonale. La factorisation LU sera différente suivant que l'on place les 1 sur la diagonale de L ou la diagonale de U.

En quoi cela nous aide-t-il à résoudre $Ax = b$?

$$Ax = b \iff LUx = b$$

Posons $Ux = y$. **La résolution de $Ax = b$ se fait alors en deux étapes :**

1. Résoudre, par une descente, $Ly = b \ \ (\rightarrow \ y)$
2. Résoudre, par une remontée, $Ux = y \ \ (\rightarrow \ x)$

C'est la **méthode de résolution par décomposition LU**.

- Si l'on change le second membre b, on peut réutiliser la factorisation LU de la matrice A. Il suffira à nouveau d'effectuer une descente puis une remontée pour trouver x,
- L'hypothèse $\det(A^k) \neq 0$ permet de s'assurer qu'il n'y aura pas de pivots nuls pendant la décomposition,
- Lorsque les 1 sont sur la diagonale de L ($L_{ii} = 1$), il s'agit de la **décomposition de Doolittle** (ce que fait Matlab par défaut). Lorsque les 1 sont sur la diagonale de U ($U_{ii} = 1$), il s'agit de la **décomposition de Crout**.

Que faire en cas de pivots nuls ? → On réalise une **permutation des lignes** (exemple 3.33 p.115). Cela permet alors d'avoir un théorème un peu plus général.

Théorème (Décomposition $PA = LU$)

Soit A une matrice inversible. Il existe P une matrice produit de matrice de permutations, L triangulaire inférieure et U triangulaire supérieure telles que:

$$PA = LU, \text{ avec } |\det(P)| = 1$$

En pratique:

- Une permutation de ligne est toujours faite avant de faire les opérations pour annuler les termes de la colonne.
- Même lorsque les permutations de lignes ne sont pas «nécessaire», il est courant d'en réaliser afin que le pivot (l_{ii} ou u_{jj}) soit de valeur absolue la plus grande possible.

Voyons comment déterminer de façon automatique les matrices L et U, pour en tirer un algorithme implémentable.

Algorithme de décomposition LU (Crout): Supposons connues les $j-1$ premières colonnes de L et les $j-1$ premières lignes de U, on veut calculer la ligne j de U et la colonne j de L

- Calcul du pivot:

$$l_{jj} = a_{jj} - \sum_{k=1}^{j-1} l_{jk} u_{kj}$$

- Calcul de la colonne j de L pour $j+1 \leq i \leq n$:

$$l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj}$$

- Calcul de la ligne j de U pour $j+1 \leq i \leq n$:

$$u_{ji} = \frac{1}{l_{jj}} \left(a_{ji} - \sum_{k=1}^{j-1} l_{jk} u_{ki} \right)$$

Je vous **déconseille fortement** de retenir ces formules !

Il suffit d'écrire ce que l'on «veut». Exemple sur une matrice 4×4 et la décomposition de Crout.

$$\begin{pmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} \end{pmatrix} \begin{pmatrix} 1 & u_{12} & u_{13} & u_{14} \\ 0 & 1 & u_{23} & u_{24} \\ 0 & 0 & 1 & u_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

1. Première colonne de L : (lignes de L) $\times$ (première colonne de U)

$$l_{11} = a_{11}, \quad l_{21} = a_{21}, \quad l_{31} = a_{31}, \quad l_{41} = a_{41}.$$

$$\begin{pmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} \end{pmatrix}$$

2. Première ligne de U : (première ligne de L) $\times$ (colonnes de U)

$$l_{11}u_{12} = a_{12} \Rightarrow u_{12} = a_{12}/l_{11}, \quad u_{13} = a_{13}/l_{11}, \quad u_{14} = a_{14}/l_{11},$$

$$\begin{pmatrix} 1 & u_{12} & u_{13} & u_{14} \\ 0 & 1 & u_{23} & u_{24} \\ 0 & 0 & 1 & u_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3. Deuxième colonne de L : (lignes de L) $\times$ deuxième colonne de U

$$l_{22} = a_{22} - l_{21}u_{12}, \quad l_{32} = a_{32} - l_{31}u_{12}, \quad l_{42} = a_{42} - l_{41}u_{12}.$$

$$\begin{pmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} \end{pmatrix}$$

4. Deuxième ligne de U : (deuxième ligne de L) $\times$ (colonnes de U)

$$u_{23} = (a_{23} - l_{21}u_{13})/l_{22}, \quad u_{24} = (a_{24} - l_{21}u_{14})/l_{22}.$$

$$\begin{pmatrix} 1 & u_{12} & u_{13} & u_{14} \\ 0 & 1 & u_{23} & u_{24} \\ 0 & 0 & 1 & u_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

5. Troisième colonne de L : (lignes de L) $\times$ troisième colonne de U

$$l_{33} = a_{33} - l_{31}u_{13} - l_{32}u_{23}, \quad l_{43} = a_{43} - l_{41}u_{13} - l_{42}u_{23}.$$

$$\begin{pmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} \end{pmatrix}$$

6. Troisième ligne de U : (troisième ligne de L) $\times$ (quatrième colonne de U)

$$u_{34} = (a_{34} - l_{31}u_{14} - l_{32}u_{24})/l_{33}.$$

$$\begin{pmatrix} 1 & u_{12} & u_{13} & u_{14} \\ 0 & 1 & u_{23} & u_{24} \\ 0 & 0 & 1 & u_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

7. Quatrième colonne de L : (lignes de L) $\times$ (quatrième colonne de U)

$$l_{44} = a_{44} - l_{41}u_{14} - l_{42}u_{24} - l_{43}u_{34}.$$

$$\begin{pmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} \end{pmatrix}$$

Pour limiter l'espace mémoire utilisé, la décomposition LU peut être stockée dans la matrice A (la matrice A est détruite au cours de l'opération)[9].

Définition (LU compacte)

La matrice LU compacte est la matrice:

$$\begin{pmatrix} l_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ l_{21} & l_{22} & u_{23} & \dots & u_{2n} \\ l_{31} & l_{32} & l_{33} & \dots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \dots & l_{nn} \end{pmatrix}$$

[9] voir `LU_Crout.m` et `LU_Doolittle.m` pour implémentations de la factorisation LU. Voir `script_LU.m` pour un exemple numérique.

Nombre d'opérations pour la décomposition LU

- nombre d'additions/soustractions : $\frac{2n^3-3n^2+n}{6} \rightarrow O(\frac{2n^3}{6})$
- nombre de divisions/multiplications : $\frac{n^3-n}{3} \rightarrow O(\frac{n^3}{3})$

Nombre d'opérations pour les remontées

- nombre d'additions/soustractions : $n^2 - n \rightarrow O(n^2)$
- nombre de divisions/multiplications : $n^2 \rightarrow O(n^2)$

Est-ce toujours raisonnable ?

En faisant 10^{12} opérations par seconde, il faudra :

- 185 heures $\approx$ 7.7 jours pour 10^6 inconnues

La décomposition LU permet de calculer facilement le déterminant de A, pour la décomposition de Crout, on aura:

$$\det(A) = \det(L)\det(U) = \det(L) = \prod_{i=1}^{n} l_{ii}$$

Et pour Doolittle,

$$\det(A) = \det(L)\det(U) = \det(U) = \prod_{i=1}^{n} u_{ii}$$

✓ Même si la matrice A est une matrice creuse, *i.e* une matrice comportant de nombreux zéros, il n'y a aucune raison que la matrice compacte résultant de la factorisation LU soit également creuse.

Définition (Matrice à diagonale strictement dominante)

Une matrice A est à **diagonale strictement dominante** par lignes si:

$$\forall i,\ |a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}|$$

ou par colonnes si:

$$\forall j,\ |a_{jj}| > \sum_{i=1, i\neq j}^{n} |a_{ij}|$$

Théorème

Si A est à diagonale strictement dominante par colonne ou par ligne:

- A est inversible.
- L'élimination de Gauss et la décomposition LU se font sans permuter de lignes

Conclusion factorisation LU:

- Si $\det(A) = 0$, le système n'a pas de solution unique,
- Peu efficace pour n grand,
- Si tous les déterminants des sous-matrices principales A^k de A **sont non nuls, alors la décomposition n'exige pas de permutations**. De même si A est à diagonale strictement dominante,
- Si on a seulement A inversible ($\det(A) \neq 0$) alors on a l'existence d'une décomposition de la forme $PA = LU$, avec P une matrice de permutation.
- En pratique, on stocke la factorisation directement dans la matrice A. Attention, il est possible que la factorisation LU prenne plus de place en mémoire que la matrice A elle-même !

Voyons à présent la décomposition pour un type particulier de matrice.

Définition (Matrice Symétrique Définie Positive (SDP))

Soit A une matrice symétrique ($A = A^t$). A est dite définie positive si

$$Ax \cdot x = x^t Ax > 0, \quad \forall x \neq 0$$

Théorème (Factorisation de Cholesky)

Soit A une **matrice symétrique**, les propositions suivantes sont équivalentes:

- A est définie positive,
- Les déterminants de toutes les sous matrices principales A^k de A sont **strictement positifs**,
- (les valeurs propres de A sont réelles et strictement positives),
- Il existe une factorisation dite de **Cholesky** de A : $\boldsymbol{A = LL^t}$, L étant une matrice triangulaire inférieure dont les termes sur la diagonale (l_{ii}) sont strictement positifs.

En pratique, on cherche donc une matrice L, triangulaire inférieure telle que

$$\begin{pmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} \end{pmatrix} \begin{pmatrix} l_{11} & l_{21} & l_{31} & l_{41} \\ 0 & l_{22} & l_{32} & l_{42} \\ 0 & 0 & l_{33} & l_{43} \\ 0 & 0 & 0 & l_{44} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

Comme pour LU, on se sert directement de la matrice A pour déterminer les coefficients de L.

Quelques considérations sur la factorisation de Cholesky:

- Si $A = LL^t$, alors $\det(A) = \det(L)\det(L^t) = \det(L)^2 = (\Pi_{i=1}^n l_{ii})^2$
- La condition $l_{ii} > 0$ dans le théorème précédent, sert à assurer l'unicité de la factorisation,
- On ne construit et ne stocke qu'une matrice triangulaire inférieure L,
- La résolution de $Ax = b$ se fait toujours en deux étapes, comme pour LU. Seule l'étape de décomposition coûte moins cher.

☢ Attention !

Il faut bien faire attention à vérifier les hypothèses du théorème avant d'effectuer la décomposition de Cholesky, c-à-d vérifier que A est symétrique et définie positive.

Avant de se lancer dans les calculs pour voir si une matrice est «Choleskysable», peut-on éliminer d'office certaines matrices ?

Supposons que A est «Choleskysable», alors A est symétrique et $A = LL^t$

$$\Rightarrow a_{ii} = \sum_{k=1}^{n} l_{ik}^2 > 0$$

Autrement dit, si la matrice A possède au moins un coefficient négatif sur sa diagonale, alors la décomposition de Cholesky ne sera pas applicable.

Proposition

Soit A une matrice **symétrique** et a **diagonale strictement dominante**, dont les **termes diagonaux sont strictement positifs**, alors A admet une factorisation de Cholesky.

Attention !

Si la matrice A ne vérifie pas les conditions de la proposition, on ne peut rien conclure. Ce sont des conditions suffisantes mais pas nécessaires.

Surprise: les calculs pour la décomposition et la résolution ne seront pas exacts ! Deux sources d'erreurs:

- **Erreurs liées à la représentation** de A et b, Exemple:

$$\begin{pmatrix} 1 & 2 \\ 1.1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 10 \\ 10.4 \end{pmatrix}$$

La solution exacte est $x = (4, 3)$. Si dans la matrice on prend 1.05 au lieu de 1.1, la solution exacte est $x = (8, 1)$.

→ **Mauvaise stabilité** par rapport aux variations des coefficients[10].

✓ En pratique la représentation des nombres sur ordinateur fera en sorte que A et b ne seront pas représentés exactement, ce qui peut donc avoir de grandes répercussions sur la solution du système.

- **Erreurs liées à la méthode de résolution** et provenant des calculs en arithmétique flottante (ex: divisions et soustractions dangereuses pendant les remontées ou descentes triangulaires).

[10] Voir le fichier `script_stabilite.m` pour un exemple numérique.

Solutions partielles pour limiter les erreurs:

- Dans LU faire des permutations de lignes pour choisir le plus grand pivot.
- Mettre les lignes à l'échelle, on divise chaque ligne par le plus grand coefficient de la ligne en valeur absolue,

$$a_{ij} \leftarrow \frac{a_{ij}}{\max_j a_{ij}}, \quad i = 1, \ldots, n.$$

Cependant, pas de solutions miracles. On va se définir des outils pour chercher à mesurer ces erreurs.

Objectifs :

- Mesurer l'écart entre la solution numérique et la solution exacte,
- Mesurer la sensibilité du système à des perturbations.

Comment mesurer la distance entre deux vecteurs ?

Définition (Norme vectorielle)

Une norme est une application de $\mathbb{R}^n$ dans $\mathbb{R}^+$ qui à $x \in \mathbb{R}^n$ associe $\|x\|$ telle que:

- $\|x\| \geq 0$ et $\|x\| = 0 \Rightarrow x = 0$.
- Si $\alpha \in \mathbb{R}$, $\|\alpha x\| = |\alpha| \, \|x\|$
- $\|x + y\| \leq \|x\| + \|y\|$

Des exemples de normes:

- Norme 1 : $\|x\|_1 = \sum_{k=1}^{n} |x_k| = |x_1| + |x_2| + \cdots + |x_n|$
- Norme 2 : $\|x\|_2 = \sqrt{\sum_{k=1}^{n} |x_k|^2} = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$
- Norme infinie : $\|x\|_\infty = \max_k |x_k| = \max\{|x_1|, |x_2|, \ldots, |x_n|\}$

Comment mesurer la distance entre deux matrices ?

Définition (Norme matricielle)

Une norme matricielle est une application de $M_n(\mathbb{R})$ dans $\mathbb{R}^+$ qui à $A \in M_n(\mathbb{R})$ associe $\|A\|$ telle que:

- $\|A\| \geq 0$ et $\|A\| = 0 \Rightarrow A = 0$.
- Si $\alpha \in \mathbb{R}$, $\|\alpha A\| = |\alpha| \, \|A\|$
- $\|A + B\| \leq \|A\| + \|B\|$
- $\|AB\| \leq \|A\|\|B\|$

Définition (Norme matricielle induite ou subordonnée)

Pour une norme vectorielle donnée $\| \cdot \|_i$, la norme matricielle induite est définie par:

$$\|A\|_i = \sup_{\|x\|_i \neq 0} \frac{\|Ax\|_i}{\|x\|_i} = \sup_{\|x\|_i = 1} \|Ax\|_i$$

Exemples de normes matricielles usuelles:

- La norme matricielle induite par la norme 1 (max sur la somme, en valeurs absolues, par colonne):

$$\|A\|_1 = \max_{1\leq j\leq n} \sum_{i=1}^{n} |a_{ij}| = \max_{1\leq j\leq n} \{|a_{1j}| + |a_{2j}| + \cdots + |a_{nj}|\}$$

- La norme matricielle induite par la norme infinie (max sur la somme, en valeurs absolues, par ligne):

$$\|A\|_\infty = \max_{1\leq i\leq n} \sum_{j=1}^{n} |a_{ij}| = \max_{1\leq i\leq n} \{|a_{i1}| + |a_{i2}| + \cdots + |a_{in}|\}$$

✓ On a $\|A\|_1 = \|A^t\|_\infty$, et donc si A symétrique $\|A\|_1 = \|A\|_\infty$

- Norme de Frobenius (norme induite par aucune norme):

$$\|A\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}^2}$$

Définition (Normes compatibles)

Une norme vectorielle $\|\cdot\|_v$ et une norme matricielle $\|\cdot\|_M$ sont compatibles si pour tout $x \in \mathbb{R}^n$ et $A \in M_n(\mathbb{R})$:

$$\|Ax\|_v \leq \|A\|_M \|x\|_v$$

Théorème (Normes compatibles)

- Toutes les normes matricielles induites sont compatibles avec leurs normes vectorielles,

$$\|Ax\|_1 \leq \|A\|_1 \|x\|_1, \quad \|Ax\|_\infty \leq \|A\|_\infty \|x\|_\infty,$$

- La norme de Frobenius est compatible avec la norme euclidienne,

$$\|Ax\|_2 \leq \|A\|_F \|x\|_2,$$

En pratique, on utilise souvent la norme 1 ou la norme ∞, car plus simple à calculer.

Erreur absolue et relative pour la résolution de $Ax = b$: Soit x^* la solution numérique et x la solution exacte. Le **vecteur de l'erreur** est défini par

$$e = x - x^*.$$

On souhaite que la norme de ce vecteur, $\|e\|_v$ soit la plus faible possible. Pour chaque x^*, on peut également associé le vecteur

$$r = b - Ax^*.$$

r est appelé **le vecteur résidu**. On a immédiatement

$$r = b - Ax^* = Ax - Ax^* = A(x - x^*) = Ae. \tag{6}$$

En utilisant (6), et en utilisant le fait que les normes sont compatibles, on montre le résultat suivant

Proposition

On contrôle l'erreur relative due à l'algorithme de résolution de $Ax = b$ via l'inégalité suivante

$$\frac{1}{\|A\|_M \|A^{-1}\|_M} \frac{\|r\|_v}{\|b\|_v} \leq \frac{\|e\|_v}{\|x\|_v} \leq \|A\|_M \|A^{-1}\|_M \frac{\|r\|_v}{\|b\|_v}.$$

La quantité $\|A\|_M \|A_M^{-1}\|$ permet de mesurer l'effet de la matrice sur la résolution du système et porte un nom particulier

Définition (Conditionnement)

Le conditionnement d'une matrice A inversible pour la norme matricielle $\|\cdot\|_M$ est défini par:

$$\mathrm{cond}_M(A) = \|A\|_M \|A^{-1}\|_M$$

- Le conditionnement n'est définie que pour les matrices inversibles,
- Le conditionnement **dépend de la norme choisie**,
- $\mathrm{cond}_M(A) \geq 1$,
- **Problème** : le conditionnement fait intervenir A^{-1} qui en règle général est très difficile à calculer.

L'inégalité précédente s'écrit alors

$$\frac{1}{\mathrm{cond}_M(A)} \frac{\|r\|_v}{\|b\|_v} \leq \frac{\|e\|_v}{\|x\|_v} \leq \mathrm{cond}_M(A) \frac{\|r\|_v}{\|b\|_v}. \tag{7}$$

Concrètement, l'inégalité permet de dire que **dans le pire des cas**, l'erreur relative sera

$$\frac{\|e\|_v}{\|x\|_v} = \mathrm{cond}_M(A)\frac{\|r\|_v}{\|b\|_v}.$$

Il est donc préférable que le **conditionnement soit petit**, idéalement $\mathrm{cond}_M(A) \approx 1$. Un «grand» conditionnement indique qu'il est possible que l'erreur soit importante.

$\rightarrow$ Plus le conditionnement de A est grand, plus on a intérêt à avoir un petit résidu, on devra alors être attentif à la qualité de la méthode de résolution.

L'inégalité suppose que la matrice A et le second membre b sont représentés exactement en machine, ce qui n'est pas le cas !

Cela nous amène à **l'erreur de représentation**.

Erreur de représentation: L'analyse est faite seulement sur A, mais le raisonnement est le même sur b.

Soit ΔA la matrice des erreurs de représentation sur A:

$$A^* = A + \Delta A$$

et x^* la solution du système perturbé, $A^*x^* = b$. Alors on a le résultat suivant,

Proposition

Soit x^* la solution de $(A + \Delta A)x^* = b$ et x la solution exacte de $Ax = b$

$$\frac{\|x - x^*\|_v}{\|x^*\|_v} = \frac{\|e\|_v}{\|x^*\|_v} \leq \text{cond}_M(A) \frac{\|\Delta A\|_M}{\|A\|_M}$$

Que nous dit cette inégalité ?

- Si $\text{cond}_M(A)$ est grand, une petite variation de A pourrait résulter en une grande variation sur la solution du système.
- Inversement, si $\text{cond}_M(A)$ est petit, une petite variation de A entraîne une petite variation sur la solution du système.

Il reste un point à voir, comment calculer $\mathrm{cond}_M(A)$ **sans inverser** A^{-1} ? En utilisant l'inégalité (7), on obtient **une borne minimale**

$$\mathrm{cond}_M(A) \geq \max\left(\frac{\|e\|_v\|b\|_v}{\|r\|_v\|x\|_v}, \frac{\|r\|_v\|x\|_v}{\|e\|_v\|b\|_v}\right)$$

✓ La borne dépend de b, x et la méthode de résolution. Le conditionnement ne dépend que de A.

En pratique, x est inconnu ! $\rightarrow$ on se construit un problème[11].

1. On choisit y quelconque et on calcul $b = Ay$,
2. On calcul y^* avec la méthode de résolution désirée,
3. On calcul $r = b - Ay^*$ et $e = y - y^*$,
4. On calcul le $\max$

La borne obtenue dépend du point y choisi arbitrairement, de la méthode de résolution et de la norme choisie. Mais cela donne toujours «une idée» du conditionnement. C'est le mieux que l'on puisse faire sans calculer A^{-1}.

[11] Voir le fichier `script_borne_conditionnement.m` pour exemple numérique.

Conclusion sur le conditionnement:

- On a toujours $\mathrm{cond}_M(A) \geq 1$.
- Le conditionnement permet de contrôler l'erreur (relative), il ne dit rien, a priori, sur la valeur de l'erreur relative,
- Un conditionnement petit assure un bon contrôle de l'erreur de notre solution,
- Avoir un petit conditionnement, n'assure pas une erreur relative petite,
- Un conditionnement grand indique qu'il faut être attentif à l'algorithme choisi et aux possibles propagations d'erreurs,
- Un grand conditionnement n'assure pas que l'erreur relative sera grande,
- On évite, en pratique, de calculer A^{-1} pour calculer le conditionnement. On préférera se servir de la borne inférieure trouvée.

BONUS ! Pas à l'examen

La notion de conditionnement est lié à la notion de stabilité d'un problème,

Définition (Problème bien posé)

Un problème est dit **bien posé (ou stable)** si la solution **existe**, est **unique**, et si **la solution dépend continûment des données**.

Le dernier point de la définition signifie que pour un problème stable, une petite perturbation des données (le second membre b ou la matrice A), doit entraîner une petite perturbation de la solution (le vecteur x).

Lorsque ce n'est pas le cas (une petite perturbation des données entraîne une grande variation de la solution), on dit que le problème est mal posé, ou instable.

Si le conditionnement de A est «petit» (et A inversible), la résolution de $Ax = b$ est un problème stable par rapport aux perturbations des données.

BONUS ! Pas à l'examen

Si l'on sait que notre matrice A est mal conditionnée, que peut-on faire ?
$\rightarrow$ On transforme le système de départ ! Soit S une matrice inversible, alors

$$Ax = b \iff SAx = Sb$$

Le but est donc de trouver une matrice S telle que $\mathrm{cond}_M(SA)$ soit meilleur que $\mathrm{cond}_M(A)$ (c'est-à-dire plus petit).
$\rightarrow$ On dit que S est une **matrice de préconditionnement** (à gauche).

Trouver une bonne matrice de préconditionnement reste en général un problème délicat...

On fera cependant toujours en sorte que S préserve les «bonnes propriétés» de la matrice A, telle que la symétrie.

Voyons à présent la résolution d'un **système d'équations non linéaires**. **Notation**:

- Soit $(f_i)_{1\leq i\leq n}$ n fonctions de n variables $(x_1, x_2, \ldots, x_n)$, différentiables,
- On définit pour $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$, $F(x) = (f_1(x), f_2(x), \cdots, f_n(x))$.
- On veut résoudre $F(x) = 0$. On note $r = (r_1, r_2, \ldots, r_n)$ la solution (racine).

En terme de système, cela revient à trouver $x = (x_1, \ldots, x_n)$ tel que:

$$\begin{cases} f_1(x_1, x_2, \cdots, x_n) = 0 \\ f_2(x_1, x_2, \cdots, x_n) = 0 \\ \qquad\qquad\qquad\qquad \vdots \\ f_n(x_1, x_2, \cdots, x_n) = 0 \end{cases}$$

Mis à part la méthode de la bissection, les méthodes du chapitre 2 peuvent être adaptées pour résoudre ce système. On présente ici la méthode la plus utilisée: la **méthode de Newton**.

Principe: On part de $x^0 = (x_1^0, x_2^0, \dots, x_n^0)$ - idéalement proche de la racine - et on cherche une correction $\delta x^0 = (\delta x_1^0, \delta x_2^0, \dots, \delta x_n^0)$ telle que pour tout i, $f_i(x^0 + \delta x^0) = 0$.
Cela nous donne, en réalisant un développement de Taylor à l'ordre 1,

$$\begin{cases} 0 = f_1(x^0 + \delta x^0) \approx f_1(x^0) + \delta x_1^0 \dfrac{\partial f_1}{\partial x_1}(x^0) + \delta x_2^0 \dfrac{\partial f_1}{\partial x_2}(x^0) + \cdots + \delta x_n^0 \dfrac{\partial f_1}{\partial x_n}(x^0) \\ 0 = f_2(x^0 + \delta x^0) \approx f_2(x^0) + \delta x_1^0 \dfrac{\partial f_2}{\partial x_1}(x^0) + \delta x_2^0 \dfrac{\partial f_2}{\partial x_2}(x^0) + \cdots + \delta x_n^0 \dfrac{\partial f_2}{\partial x_n}(x^0) \\ \vdots \\ 0 = f_n(x^0 + \delta x^0) \approx f_n(x^0) + \delta x_1^0 \dfrac{\partial f_n}{\partial x_1}(x^0) + \delta x_2^0 \dfrac{\partial f_n}{\partial x_2}(x^0) + \cdots + \delta x_n^0 \dfrac{\partial f_n}{\partial x_n}(x^0) \end{cases}$$

Soit, sous forme matricielle,

$$\vec{0} = F(x^0) + \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(x^0) & \frac{\partial f_1}{\partial x_2}(x^0) & \dots & \frac{\partial f_1}{\partial x_n}(x^0) \\ \frac{\partial f_2}{\partial x_1}(x^0) & \frac{\partial f_2}{\partial x_2}(x^0) & \dots & \frac{\partial f_2}{\partial x_n}(x^0) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1}(x^0) & \frac{\partial f_n}{\partial x_2}(x^0) & \dots & \frac{\partial f_n}{\partial x_n}(x^0) \end{pmatrix} \begin{pmatrix} \delta x_1^0 \\ \delta x_2^0 \\ \vdots \\ \delta x_n^0 \end{pmatrix}$$

Définition (Matrice jacobienne)

On note $J(x^i)$, la matrice dite jacobienne définie par

$$J(x^i) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(x^i) & \frac{\partial f_1}{\partial x_2}(x^i) & \dots & \frac{\partial f_1}{\partial x_n}(x^i) \\ \frac{\partial f_2}{\partial x_1}(x^i) & \frac{\partial f_2}{\partial x_2}(x^i) & \dots & \frac{\partial f_2}{\partial x_n}(x^i) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1}(x^i) & \frac{\partial f_n}{\partial x_2}(x^i) & \dots & \frac{\partial f_n}{\partial x_n}(x^i) \end{pmatrix} = \begin{pmatrix} \left(\nabla f_1(x^i)\right)^t \\ \left(\nabla f_2(x^i)\right)^t \\ \vdots \\ \left(\nabla f_n(x^i)\right)^t \end{pmatrix}$$

On cherche alors une correction δx^0 telle que

$$\vec{0} = F(x^0) + J(x^0)\delta x^0 \iff J(x^0)\delta x^0 = -F(x^0)$$

$-F(x^0)$ est appelé le **résidu** en x^0. La résolution de ce système matriciel nous donne la correction $\delta x^0 = -J(x^0)^{-1}F(x^0)$ à apportée à x^0.

On ne calcul pas l'inverse, on résout le système !

On recommence ensuite la démarche à partir de $x^1 = x^0 + \delta x^0$.

Algorithme de Newton pour les systèmes:

Algorithme de Newton, pseudo-code `script_Newton_systeme.m`

Données :

- Des fonctions f_i, $1 \leq i \leq n$ deux fois différentiables et un point de départ $x^0 = (x_0, x_1, \ldots, x_n)$. $F(x^0) = (f_1(x^0), \ldots, f_n(x^0))$
- Une tolérance d'arrêt ε_a et un nombre max d'itérations n_{max}
- la précision machine ε_m.

Résultat : le vecteur x^{i+1} qui contiendra une approximation de la racine $r = (r_1, \ldots, r_n)$

1 **tant que** $\frac{\|x^{i+1} - x^i\|}{\|x^{i+1}\| + \varepsilon_m} > \varepsilon_a$ *et* $n \leq n_{max}$ **faire**
2 Résoudre $J(x^i)\delta x^i = -F(x^i)$;
3 $x^{i+1} = x^i + \delta x^i$;
4 $i = i + 1$;
5 **fin**

- L'inverse de la matrice jacobienne $J_F(x^0)^{-1}$ joue le rôle que jouait $\frac{1}{f'(x_0)}$ en dimension 1,
- À chaque étape, on résout un système linéaire de dimension n. À chaque itération il faut que $J(x^i)$ soit inversible,
- La convergence est quadratique si $J_F(r)$ est inversible. Sinon elle est linéaire (on retrouve le comportement 1D dans le cas des racines multiples),
- Pour étudier la convergence on fera les mêmes tableaux qu'en 1D, en remplaçant les valeurs absolues par des normes vectorielles,
- Il est primordial (plus qu'en 1D) que x^0 soit proche de r sous peine de divergence possible.
- Il faut aussi calculer la matrice jacobienne d'une fonction à plusieurs variables ce qui peut être très fastidieux.

Des variantes:

- **Quasi-Newton** : On ne calcule pas la matrice jacobienne à chaque itération. On garde la même pendant plusieurs itérations (on économise le calcul de dérivées et si factorisation LU utilisée, on réduit le nombre de factorisation).
- **Newton modifiée**: On remplace les dérivées partielles par des approximations précises, pour $h > 0$ petit:

$$\frac{\partial f_i}{\partial x_j}(x) = \frac{f_i(x_1, \cdots, x_j + h, \cdots, x_n) - f_i(x_1, \cdots, x_j - h, \cdots, x_n)}{2h}$$

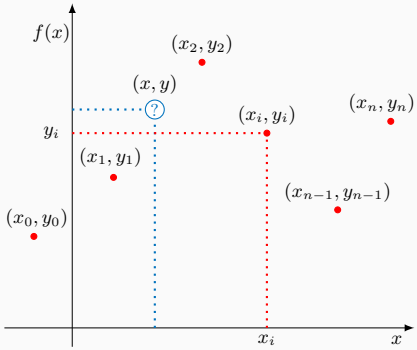
Je dois être capable de :

- Faire une méthode d'élimination de Gauss d'un système linéaire et définir les matrices équivalentes aux opérations élémentaires,
- Faire une décomposition et une résolution par LU et Cholesky,
- Distinguer Doolittle et Crout,
- Reconnaître des matrices permettant d'appliquer ou d'exclure Cholesky,
- Comprendre la notion de conditionnement d'une matrice,
- Connaître le théorème sur le conditionnement,
- Appliquer le théorème de conditionnement pour caractériser une solution et son erreur relative,
- Construire une borne inférieure du conditionnement et je comprends ses limitations,
- Calculer une matrice Jacobienne,
- Utiliser la méthode de Newton pour les systèmes non linéaires, et comprend ses limitations et ses variantes.

Chapitre 5: Interpolation

Principe: étant donné un ensemble de $n+1$ points $(x_i, y_i) = (x_i, f(x_i))$, $i = 0, \ldots, n$, on souhaite construire une fonction continue $p(x)$ passant par ces $n+1$ points, c-à-d telle que

$$p(x_i) = y_i, \quad i = 0, \ldots, n.$$



Les abscisses x_i sont appelés **noeuds d'interpolation**. Les points (x_i, y_i) sont appelés **points d'interpolation**, ou points de collocation.

Autrement dit, si l'on ne connaît que les points d'interpolation $(x_i, f(x_i))$, est-on capable d'obtenir une approximation de la fonction f pour une valeur de x différentes des x_i ?

But: Comprendre le comportement général d'un phénomène à partir de résultats expérimentaux,

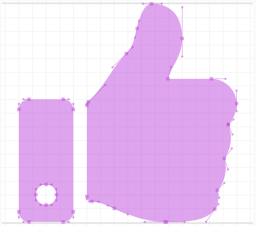
Domaines d'applications très divers: CAO, synthèse d'images, métrologie, géologie, typographie,...

Ce que nous allons voir:

- Interpolation de Lagrange,
- Interpolation de Newton,
- Splines cubiques.

Ce que nous n'allons pas voir, mais également utilisé en pratique:

- Courbes de Bézier, B-Splines, NURBS,
- Krigeage,
- Moindres carrés,
- ...



La solution va être de créer un polynôme de degré suffisamment élevé et passant par les points d'interpolation. On parle de **polynôme d'interpolation**, ou polynôme de collocation.

Un rappel d'algèbre:

Théorème (fondamental de l'algèbre)

Soit p_n un polynôme de degré n,

$$p_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x_n, \quad a_n \neq 0.$$

p_n possède exactement n racines (réelles ou complexes).

et une conséquence...

Proposition

Il existe un **unique** polynôme d'interpolation p_n de degré **au plus n** passant par $n+1$ points d'interpolation $(x_i, y_i)_{0 \leq i \leq n}$, où $x_i \neq x_j$ pour $i \neq j$.

Définition (Interpolant)

Si l'on suppose que les ordonnées y_i proviennent d'une fonction f, c'est-à-dire $y_i = f(x_i)$, alors l'unique polynôme p_n passant par les points $(x_i, f(x_i))$ est appelé **l'interpolant** de f.

Attention !

L'interpolant de f passant par $n+1$ points d'interpolation peut être un polynôme de degré inférieur à n (exemple: imaginez les ordonnées $f(x_i)$ alignées sur une droite !).

On a donc unicité du polynôme d'interpolation, mais a-t-on toujours existence de ce polynôme ?

$\rightarrow$ S'obtient par construction.

On a $n+1$ points (x_i, y_i), $0 \leq i \leq n$, on souhaite trouver p_n tel que

$$\begin{cases} p_n(x_0) = a_0 + a_1 x_0 + a_2 x_0^2 + \cdots + a_n x_0^n = y_0 \\ p_n(x_1) = a_0 + a_1 x_1 + a_2 x_1^2 + \cdots + a_n x_1^n = y_1 \\ \quad \vdots \qquad\qquad\qquad\qquad \vdots \\ p_n(x_n) = a_0 + a_1 x_n + a_2 x_n^2 + \cdots + a_n x_n^n = y_n \end{cases}$$

Cela revient à trouver $(a_0, a_1, \ldots, a_n)$ tels que

$$\underbrace{\begin{pmatrix} 1 & x_0 & x_0^2 & x_0^3 & \ldots & x_0^n \\ 1 & x_1 & x_1^2 & x_1^3 & \ldots & x_1^n \\ 1 & x_2 & x_2^2 & x_2^3 & \ldots & x_2^n \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & x_n^3 & \ldots & x_n^n \end{pmatrix}}_{V} \underbrace{\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}}_{a} = \underbrace{\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}}_{y}$$

Cela est donc équivalent à résoudre le système $Va = y$. La matrice V est appelée **matrice de Vandermonde**.

Quelques propriétés de la matrice de Vandermonde:

- La matrice de Vandermonde est inversible si et seulement si les abscisses sont distinctes, *i.e* $x_i \neq x_j$ pour $i \neq j$.

  $\rightarrow$ Le polynôme d'interpolation existe seulement si la matrice V est inversible, donc **quand les abscisses x_i sont distinctes**,
- Le conditionnement de la matrice augmente fortement quand n augmente,
- La résolution se comporte mal quand les abscisses sont proches ou petites,
- Si le nombre de points d'interpolation ou leurs positions changent, il faut tout recommencer !

Conclusion: Cette approche est utile uniquement du point de vue théorique, car elle donne une condition d'existence du polynôme d'interpolation. On ne l'utilisera pas numériquement.

Première approche plus efficace : **l'interpolation de Lagrange**

Étant donné $(n+1)$ points (x_i, y_i), $0 \leq i \leq n$. Supposons que l'on soit capable de construire $n+1$ polynômes $L_i(x)$ de **degré n**, tels que

$$L_i(x_j) = \begin{cases} 1 \text{ si } i = j \\ 0 \text{ si } i \neq j \end{cases}$$

Comme toute combinaison linéaire de polynômes de degré n est un polynôme de degré au plus n, le polynôme

$$L(x) = y_0 L_0(x) + y_1 L_1(x) + \cdots + y_n L_n(x) = \sum_{i=0}^{n} y_i L_i(x)$$

est un polynôme de degré au plus n. De plus, $\forall\, 0 \leq j \leq n$

$$\begin{aligned} L(x_j) &= y_0 L_0(x_j) + y_1 L_1(x_j) + \cdots + y_j L_j(x_j) + \cdots + y_n L_n(x_j) \\ &= 0 + 0 + \cdots + 0 + y_j + 0 + \cdots + 0 = y_j \end{aligned}$$

Le polynôme L et donc de degré n et passe par tous les points d'interpolation.

$$\rightarrow \; L \text{ est le polynôme recherché}$$

Comment construire les polynômes L_i, $0 \leq i \leq n$?

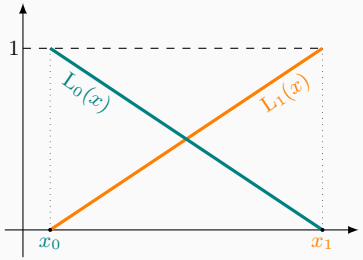
- $n = 1 \rightarrow$ deux points

On cherche deux polynômes de degré 1, L_0 et L_1 passant par les deux points (x_0, y_0) et (x_1, y_1) et vérifiant

$$\begin{cases} L_0(x_0) = 1 \\ L_0(x_1) = 0 \end{cases} \qquad \begin{cases} L_1(x_0) = 0 \\ L_1(x_1) = 1 \end{cases}$$

Posons

$$L_0(x) = \frac{(x - x_1)}{(x_0 - x_1)} \text{ et } L_1(x) = \frac{(x - x_0)}{(x_1 - x_0)}.$$



Alors,

$$L_0(x_0) = \frac{(x_0 - x_1)}{(x_0 - x_1)} = 1,$$

$$L_0(x_1) = \frac{(x_1 - x_1)}{(x_0 - x_1)} = 0.$$

De même, $L_1(x_0) = 0, L_1(x_1) = 1$.

Ainsi, le polynôme d'interpolation de degré 1 passant par (x_0, y_0) et (x_1, y_1) est

$$\begin{aligned} L(x) &= y_0 L_0(x) + y_1 L_1(x) \\ &= y_0 \frac{(x - x_1)}{(x_0 - x_1)} + y_1 \frac{(x - x_0)}{(x_1 - x_0)} \end{aligned}$$

- $n = 2 \rightarrow$ trois points

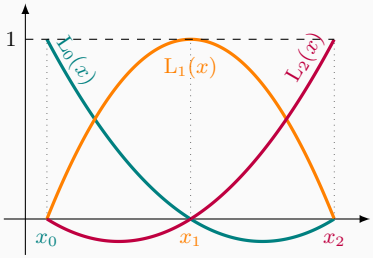
On cherche trois polynômes de degré 2, L_0, L_1, L_2 passant par les points (x_0, y_0), (x_1, y_1), (x_2, y_2) et vérifiant

$$\begin{cases} \mathrm{L}_i(x_i) = 1 & \forall\, i = 1, 2, 3 \\ \mathrm{L}_i(x_j) = 0 & \forall\, i \neq j \end{cases} \tag{8}$$

Posons

$$L_0(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)}, \qquad L_1(x) = \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)},$$

$$L_2(x) = \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}$$



Encore une fois, on a bien

$$L_0(x_0) = 1, L_0(x_1) = 0, L_0(x_2) = 0$$

$$L_1(x_0) = 0, L_1(x_1) = 1, L_1(x_2) = 0$$

$$L_2(x_0) = 0, L_2(x_1) = 0, L_2(x_2) = 1$$

Ainsi,

$$\begin{aligned} L(x) &= y_0 L_0(x) + y_1 L_1(x) + y_2 L_2(x) \\ &= y_0 \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} + y_1 \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} + y_2 \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} \end{aligned}$$

• pour $n+1$ points d'interpolation, on aura

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{(x - x_j)}{(x_i - x_j)}, \ i = 0, \ldots, n.$$

Ces polynômes vérifient bien

$$L_i(x_j) = \begin{cases} 1 \text{ si } i = j \\ 0 \text{ si } i \neq j \end{cases}$$

et le polynôme d'interpolation sera alors [12]

$$p_n(x) = \sum_{i=0}^{n} y_i L_i(x).$$

Inconvénient : si l'on rajoute un point d'interpolation, il faut recalculer tous les polynômes L_i.

[12] voir `script_Interpolation_Lagrange.m` pour exemples numériques.

Interpolation de Newton : se base sur une écriture différente du polynôme recherché.

Lagrange: $p_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$
Newton :

$$\begin{aligned} p_n(x) &= a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \dots \\ &\quad + a_{n-1}(x - x_0)(x - x_1)(x - x_2)\dots(x - x_{n-2}) \\ &\quad + a_n(x - x_0)(x - x_1)(x - x_2)\dots(x - x_{n-1}) \\ &= p_{n-1}(x) + a_n(x - x_0)(x - x_1)(x - x_2)\dots(x - x_{n-1}) \end{aligned}$$

On souhaite toujours que notre polynôme passe par les points d'interpolation (x_i, y_i), $0 \leq i \leq n$, c'est-à-dire $p_n(x_i) = y_i$.

- $i = 0 \rightarrow\ p_n(x_0) = y_0 = a_0 \Rightarrow a_0 = y_0$
- $i = 1 \rightarrow\ p_n(x_1) = y_1 = a_0 + a_1(x_1 - x_0) \Rightarrow a_1 = \frac{y_1 - y_0}{x_1 - x_0}$
- $i = 2 \rightarrow\ p_n(x_2) = y_2 = a_0 + a_1(x_2 - x_0) + a_2(x_2 - x_0)(x_2 - x_1)$

$$\Rightarrow a_2 = \frac{1}{(x_2 - x_0)(x_2 - x_1)} \left((y_2 - a_0) - a_1(x_2 - x_0)\right)$$

- $i = \dots$

Chaque polynôme dépend du polynôme précédent, **les coefficients a_i sont liés entre eux**. On va voir comment les calculer de façon efficace (et automatique).

Définition (Premières différences divisées)

On appellera première différence divisée, notée $f[x_i, x_{i+1}]$ le rapport

$$f[x_i, x_{i+1}] = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}$$

On souhaite exprimer tous les coefficients a_i en fonction de différences divisées.

Regardons comment s'écrit $p_1(x)$:
On a $a_1 = \frac{y_1 - y_0}{x_1 - x_0} = f[x_0, x_1]$, ainsi le polynôme p_1 s'écrit

$$p_1(x) = a_0 + a_1(x - x_0) = y_0 + f[x_0, x_1](x - x_0)$$

Et on a bien $p_1(x_0) = y_0$ et $p_1(x_1) = y_0 + \frac{y_1 - y_0}{\cancel{x_1 - x_0}}\cancel{(x_1 - x_0)} = y_1$.

$\rightarrow p_1$ est l'unique polynôme d'interpolation de degré 1 passant par les points (x_0, y_0) et (x_1, y_1).

Regardons ce qu'il se passe pour p_2

$$\begin{aligned} p_2(x) &= a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) \\ &= y_0 + f[x_0, x_1](x - x_0) + a_2(x - x_0)(x - x_1) \\ &= p_1(x) + a_2(x - x_0)(x - x_1) \end{aligned}$$

On a vu

$$\begin{aligned} a_2 &= \frac{1}{(x_2 - x_0)(x_2 - x_1)} \left((y_2 - a_0) - a_1(x_2 - x_0)\right) \\ &= \frac{1}{(x_2 - x_0)(x_2 - x_1)} \left((y_2 - y_0) - \frac{y_1 - y_0}{x_1 - x_0}(x_2 - x_0)\right) \end{aligned}$$

que l'on peut également écrire (après quelques calculs)

$$a_2 = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0}$$

Cette dernière forme nous amène à définir les énièmes différences divisées.

Définition (Énièmes différences divisées)

On appelle deuxième différence divisée de la fonction f, notée $f[x_i, x_{i+1}, x_{i+2}]$, le rapport

$$f[x_i, x_{i+1}, x_{i+2}] = \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i}$$

Plus généralement, la p-ième différence divisée de f, notée $f[x_i, x_{i+1}, \ldots, x_{i+p}]$ est définie par

$$f[x_i, x_{i+1}, \ldots, x_{i+p}] = \frac{f[x_{i+1}, x_{i+2}, \ldots, x_{i+p}] - f[x_i, x_{i+1}, \ldots, x_{i+p-1}]}{x_{i+p} - x_i}$$

Avec cette notation, on a $a_2 = f[x_0, x_1, x_2]$ et donc

$$p_2(x) = y_0 + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1)$$

Encore une fois, on a $p_2(x_0) = y_0, p_2(x_1) = y_1, p_2(x_2) = y_2$.

$\rightarrow$ p_2 est l'unique polynôme d'interpolation de degré 2 passant par les trois points $(x_0, y_0), (x_1, y_1), (x_2, y_2)$.

Remarque(s)

$$p_2(x) = y_0 + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1)$$
$$= p_1(x) + f[x_0, x_1, x_2](x - x_0)(x - x_1)$$

p_2 s'obtient par l'ajout d'un terme de degré 2 à p_1. Cela veut dire que si l'on a déjà calculé p_1 et que l'on ajoute un point d'interpolation (passant de 2 à 3), on peut facilement obtenir p_2.

On aurait envie de dire

$$p_3(x) = y_0 + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1)$$
$$+ f[x_0, x_1, x_2, x_3](x - x_0)(x - x_1)(x - x_2)$$

Et l'on peut ! Ce que venons de voir pour p_1 et p_2 se généralise aux polynômes de degré n.

Le théorème ci-dessous formalise ce que nous venons de voir [13]

Théorème (Interpolation de Newton)

L'unique polynôme de degré n passant par $n+1$ points d'interpolation (x_i, y_i), $0 \leq i \leq n$ est donné par

$$\begin{aligned} p_n(x) &= a_0 N_0(x) + a_1 N_1(x) + \cdots + a_n N_n(x) \\ &= p_{n-1}(x) + a_n N_n(x) \end{aligned}$$

avec $N_0(x) = 1$, $N_i(x) = (x - x_0)(x - x_1)\ldots(x - x_{i-1})$, $1 \leq i \leq n$ et

$$a_0 = f[x_0] = f(x_0),\ a_i = f[x_0, x_1, x_2, \ldots, x_i],\ 1 \leq i \leq n$$

Les polynômes N_i sont les **polynômes de Newton**.

Le calcul des différences divisées $f[x_0, x_1, x_2, \ldots, x_i]$ peut être pénible si l'on s'y prend mal. On va voir une façon astucieuse et efficace de les calculer.

[13] voir `script_Interpolation_Newton.m` pour exemples numériques.

On construit une **table des différences divisées.** Calcul colonne par colonne. Exemple avec 4 points d'interpolation:

Table de différences divisées

x_i	$f(x_i)$	$f[x_i, x_{i+1}]$	$f[x_i, x_{i+1}, x_{i+2}]$	$f[x_i, x_{i+1}, x_{i+2}, x_{i+3}]$
x_0	$f(x_0)$			
		$f[x_0, x_1]$		
x_1	$f(x_1)$		$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0}$	
		$f[x_1, x_2]$		$f[x_0, x_1, x_2, x_3]$
x_2	$f(x_2)$		$f[x_1, x_2, x_3]$	
		$f[x_2, x_3]$		
x_3	$f(x_3)$			

✓ Il n'est pas nécessaire de placer les points d'interpolation par abscisse croissante.

La diagonale de la table correspond aux coefficients a_i du polynôme d'interpolation de Newton.

$$p_3(x) = y_0 + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + f[x_0, x_1, x_2, x_3](x - x_0)(x - x_1)(x - x_2)$$

L'ordre des points n'a pas d'importance, par unicité du polynôme on doit obtenir le même résultat **à la fin.**

Table de différences divisées

x_i	$f(x_i)$	$f[x_i, x_{i+1}]$	$f[x_i, x_{i+1}, x_{i+2}]$	$f[x_i, x_{i+1}, x_{i+2}, x_{i+3}]$
x_0	$f(x_0)$			
		$f[x_0, x_1]$		
x_1	$f(x_1)$		$f[x_0, x_1, x_2]$	
		$f[x_1, x_2]$		$f[x_0, x_1, x_2, x_3]$
x_2	$f(x_2)$		$f[x_1, x_2, x_3]$	
		$f[x_2, x_3]$		
x_3	$f(x_3)$			

$$p_3(x) = y_3 + f[x_2, x_3](x - x_3) + f[x_1, x_2, x_3](x - x_3)(x - x_2) + f[x_0, x_1, x_2, x_3](x - x_3)(x - x_2)(x - x_1)$$
$$= y_0 + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + f[x_0, x_1, x_2, x_3](x - x_0)(x - x_1)(x - x_2)$$

Quelques observations/conclusions:

- Si l'on utilise tous les points d'interpolation, on a unicité du polynôme (mais les polynômes intermédiaires ne seront pas forcément les mêmes).
- Si l'on ajoute un point d'interpolation (augmentant de 1 le degré du polynôme d'interpolation), **on ne recommence pas tout**. Il suffit de rajouter une ligne dans notre table des différences divisées, et de rajouter un terme de degré $n+1$ à notre polynôme de degré n déjà calculé.
- Par unicité du polynôme passant par les $n+1$ points d'interpolation, que vous utilisiez les polynômes de Lagrange ou les polynômes de Newton, vous devez obtenir le même résultat final !

Comment évaluer l'erreur commise en approximant une fonction f par son interpolant p_n ?

$\rightarrow$ Étude de l'erreur d'interpolation

L'erreur d'interpolation E_n est définie par

$$E_n(x) = f(x) - p_n(x)$$

Puisque $p_n(x_i) = y_i = f(x_i)$, il est clair que l'erreur d'interpolation est nulle aux points d'interpolation ($E_n(x_i) = 0$) !

On cherche à évaluer l'erreur pour x «quelconque» dans l'intervalle $[x_0, x_n]$.

Dans la suite, on suppose $x_0 < x_1 < \cdots < x_n$.

Soit x^* tel que $x^* \neq x_i$, $i = 0, \ldots, n$ et $x^* \in [x_0, x_n]$.
Considérons l'interpolation de Newton et ajoutons le point $(x^*, f(x^*))$ à notre table de différences divisées. Par définition de l'interpolation on a

$$f(x^*) = p_{n+1}(x^*) = p_n(x^*) + f[x_0, x_1, \ldots, x_n, x^*](x - x_0) \ldots (x - x_n)$$

Cela nous donne

$$E_n(x^*) = f(x^*) - p_n(x^*) = f[x_0, x_1, \ldots, x_n, x^*](x - x_0)\ldots(x - x_n)$$

Si je sais évaluer $f[x_0, x_1, \ldots, x_n, x^*]$ alors je connais l'erreur d'interpolation.

Problème: on ne connaît pas $f(x^*)$! on ne peut donc pas calculer la différence divisée.

Solution: il est possible de montrer que pour $x^* \in [x_0, x_n]$ (et en supposant f suffisamment régulière)

$$f[x_0, x_1, \ldots, x_n, x^*] = \frac{f^{(n+1)}(\xi_{x^*})}{(n+1)!}, \text{ pour } \xi_{x^*} \in [x_0, x_n]$$

On caractérise donc l'erreur en fonction de la dérivée de la fonction f.

Théorème (Erreur d'interpolation)

Soit $x_0 < x_1 < \cdots < x_n$ les abscisses des points d'interpolation. Si f est $n+1$ fois dérivable sur $[x_0, x_n]$, alors $\forall\, x \in [x_0, x_n], \exists\, \xi_x \in [x_0, x_n]$ tel que

$$E_n(x) = f(x) - p_n(x) = \frac{f^{(n+1)}(\xi_x)}{(n+1)!}(x - x_0)(x - x_1)\dots(x - x_n)$$

- Par unicité du polynôme, l'erreur d'interpolation est la même que vous utilisiez les polynômes de Newton ou Lagrange (ou même Vandermonde).
- En pratique, ξ_x n'est pas connu (dans la vraie vie, pas plus que f !)
- Comme pour les développements de Taylor, dans le cas ou l'on connait la fonction f, on a recourt à une majoration de la dérivée,

$$|E_n(x)| \leq \frac{1}{(n+1)!} \max_{x_0 \leq \xi_x \leq x_n} |f^{(n+1)}(\xi_x)||(x - x_0)\dots(x - x_n)|$$

- Si on ne connaît pas la fonction f, la majoration est impossible. Dans ce cas, si on connait un point d'interpolation $(x_{n+1}, f(x_{n+1}))$ supplémentaire, on fait l'approximation suivante

$$f[x_0, x_1, \ldots, x_n, x_{n+1}] \approx \frac{f^{(n+1)}(\xi_x)}{(n+1)!}$$

et alors

$$E_n(x) \approx f[x_0, \ldots, x_n, x_{n+1}](x - x_0) \ldots (x - x_n)$$
$$\Rightarrow \boldsymbol{E_n(x) \approx p_{n+1}(x) - p_n(x)}$$

+ : facile à calculer,
- : pas toujours d'une grande précision...

- Si l'on peut choisir les points d'interpolation x_i, on a tout intérêt à les prendre **proche** de l'abscisse x où l'on souhaite interpoler notre fonction.

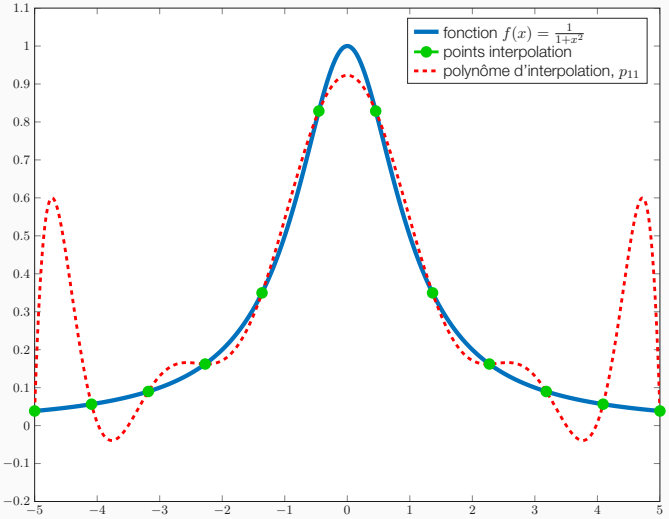
De manière général, le terme d'erreur E_n est un polynôme de degré $n+1$, **pouvant fortement osciller**.

$\rightarrow$ Phénomène de Runge : pour certaines fonctions, l'erreur augmente quand le nombre de points d'interpolation augmente.

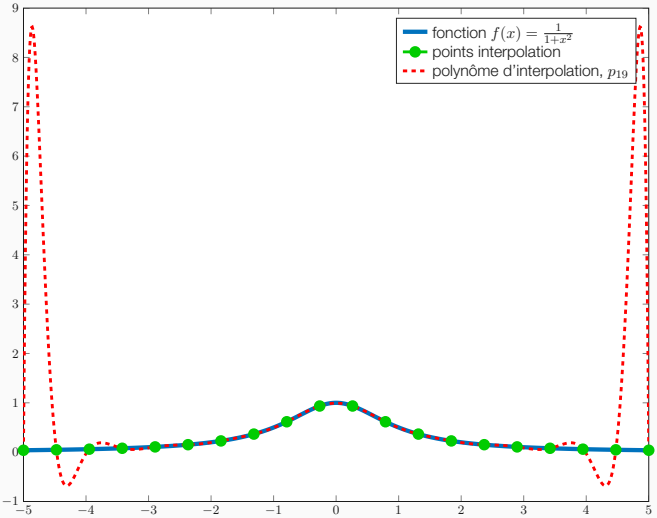
Pour illustrer le phénomène, considérons

$$f(x) = \frac{1}{1+x^2}$$

et regardons le comportement de son polynôme d'interpolation p_n, pour différentes valeurs de n.



Phénomène de Runge, $n = 11$ (12 points d'interpolation).



Phénomène de Runge, $n = 19$ (20 points d'interpolation).

→ **L'erreur augmente** avec le nombre de points !

Conclusion: L'interpolation de Lagrange/Newton n'est pas adaptée à toutes les fonctions f.

$\rightarrow$ Il est parfois peu prudent d'utiliser un grand nombre de point si l'on ne connaît pas le comportement de la fonction.

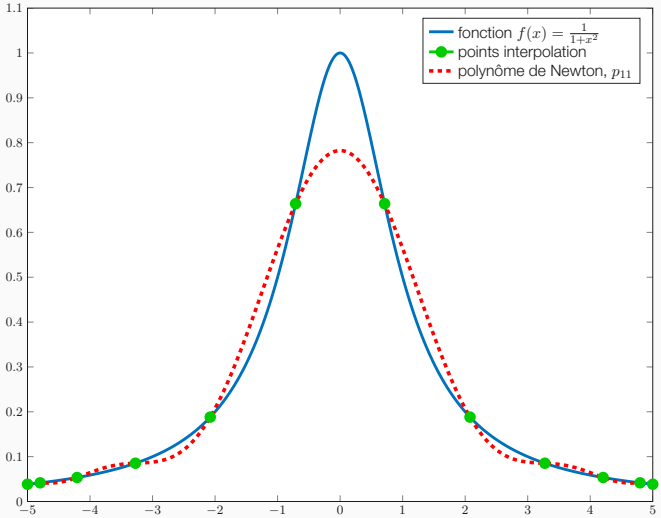
Solution partielle: mieux positionner les abscisses des points d'interpolation, de sorte à minimiser l'expression

$$|(x - x_0)(x - x_1) \dots (x - x_n)|$$

apparaissant dans le terme d'erreur E_n. Comment ?
$\rightarrow$ Abscisses de Tchebychev

$$x_i = \frac{a+b}{2} - \frac{b-a}{2} \cos\left(\frac{i\pi}{n}\right), \; i = 0, \dots, n.$$

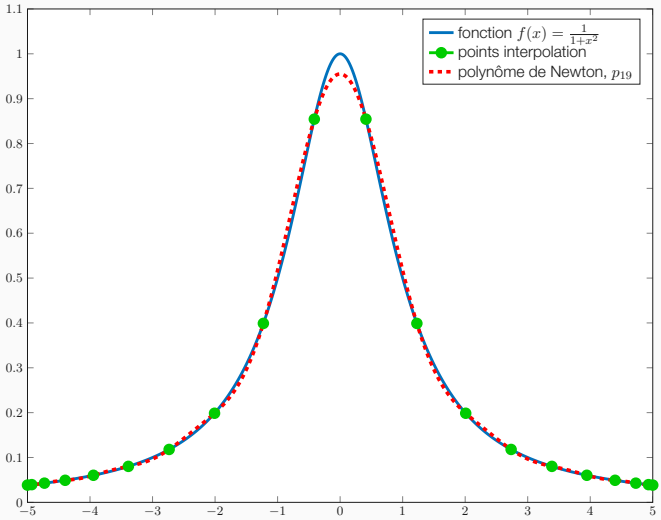
L'intervalle $[a, b]$ étant l'intervalle sur lequel on cherche à interpoler la fonction.



Phénomène de Runge, $n = 11$ (12 points d'interp.), abscisses de Tchebychev.

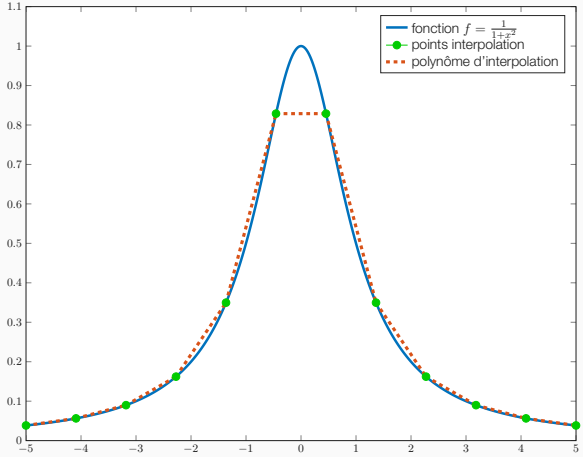
→ Toujours quelques oscillations présentent...

...que l'on parvient a atténuer en augmentant le nombre de points d'interpolation.



Phénomène de Runge, $n = 19$ (20 points d'interp.), abscisses de Tchebychev.

Autre solution: interpolation linéaire par morceaux [14] - on construit un polynôme d'interpolation de degré 1 sur chaque sous-intervalle $[x_i, x_{i+1}]$,



Interpolation linéaire par morceaux, 8 points d'interpolation

[14] voir `script_InterpolationLinParMorceaux.m` pour exemples numériques.

Avantage de l'interpolation linéaire par morceaux:

- Évite l'utilisation de polynômes de degrés élevés (et donc le phénomène de Runge, même avec des abscisses équi-distantes). Au lieu d'avoir une erreur d'interpolation portant sur $n+1$ points, on a une erreur d'interpolation sur chaque sous-intervalle $[x_i, x_{i+1}]$.

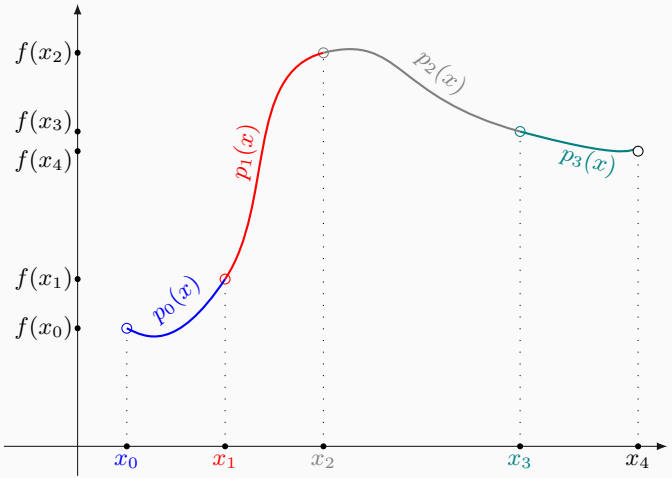
Inconvénient majeur:

- Le polynôme d'interpolation est **continu mais n'est pas dérivable aux abscisses d'interpolation x_i**.

On va garder l'idée d'interpolation par morceaux, mais en utilisant sur chaque sous-intervalle un polynôme d'un degré nous garantissant un polynôme d'interpolation suffisamment «lisse».

Polynôme de degré 2 $\rightarrow$ on peut imposer la continuité des dérivées premières aux x_i.
Polynôme de degré 3 $\rightarrow$ on peut imposer la continuité des dérivées premières et secondes aux x_i.

C'est le principe de base des **splines cubiques**: utiliser des polynômes de degré 3 sur chaque sous-intervalle, et les «recoller» adéquatement.



Recollement de 4 polynômes de degré 3

Soit $(x_i, y_i = f(x_i))$, $0 \leq i \leq n$, les $n+1$ points d'interpolation.

Idée générale des splines cubiques:
Réaliser une interpolation cubique sur les n sous-intervalles $[x_i, x_{i+1}]$ en imposant **la continuité des dérivées premières et secondes.** Soit

$$S(x) = \begin{cases} p_0(x) & x \in [x_0, x_1] \\ p_1(x) & x \in [x_1, x_2] \\ \vdots & \vdots \\ p_{n-1}(x) & x \in [x_{n-1}, x_n] \end{cases}$$

la spline recherchée. On cherche à définir sur chaque sous intervalle un polynôme de degré 3 sous la forme

$$p_i(x) = f_i + f_i'(x - x_i) + \frac{f_i''}{2}(x - x_i)^2 + \frac{f_i'''}{6}(x - x_i)^3, \quad 0 \leq i \leq n-1$$

où f_i, f_i', f_i'' et f_i''' sont des constantes (et **ne sont pas les dérivées de f !**)

✓ L'écriture des p_i sous la forme d'un développement de Taylor permet d'interpréter plus facilement les coefficients f_i, f_i', f_i'' et f_i'''. On a en effet

$$f_i = p_i(x_i), \quad f_i' = p_i'(x_i), \quad f_i'' = p_i''(x_i), \quad f_i''' = p_i'''(x_i)$$

On veut :

- $S(x_i) = y_i$, pour $0 \leq i \leq n \rightarrow\ n+1$ conditions,
- $S(x)$ continue aux $n-1$ nœuds internes, c'est-à-dire $p_i(x_{i+1}) = p_{i+1}(x_{i+1})$, $1 \leq i \leq n-1$ $\rightarrow\ n-1$ conditions,
- $S'(x)$ et $S''(x)$ continues aux $n-1$ nœuds internes, c'est-à-dire

$$p_i'(x_{i+1}) = p_{i+1}'(x_{i+1}),\ 1 \leq i \leq n-1 \rightarrow\ n-1 \text{ conditions}$$
$$p_i''(x_{i+1}) = p_{i+1}''(x_{i+1}),\ 1 \leq i \leq n-1 \rightarrow\ n-1 \text{ conditions}$$

Bilan : On a $4n-2$ équations pour $4n$ inconnues (n polynômes, chaque polynôme a 4 inconnues) $\rightarrow$ Système sous-déterminé.

Il va exister plusieurs splines cubiques pour les mêmes points d'interpolation.

Objectif: On va exprimer toutes les inconnues du problème en fonction des coefficients f_i''.

$\rightarrow$ On va ramener la construction de la spline à la résolution d'un système matriciel $n+1 \times n+1$, où les inconnues seront les $f_i'', i = 0, \cdots, n$.

Remarque(s)

On introduit une variable supplémentaire : $\boldsymbol{f_n''}$, la dérivée seconde de p_{n-1} au noeud x_n (en pratique, cette valeur sera imposée).

$$f_n'' = p_{n-1}''(x_n)$$

Commençons par exprimer les conditions en fonctions des f_i, f_i', f_i'' et f_i''', et voyons ensuite comment tout exprimer en fonction de f_i''

Expressions des conditions en fonction des f_i, f_i', f_i'' et f_i''' : On note $h_i = x_{i+1} - x_i$, pour $0 \leq i \leq n-1$.

- À la première extrémité x_i de chaque intervalle $[x_i, x_{i+1}]$, le polynôme $p_i(x)$ passe au point $(x_i, f(x_i))$. On a
$$p_i(x_i) = f_i = f(x_i), \text{ pour } i = 0, 1, \cdots, n-1. \tag{9}$$
- L'équation de continuité en x_{i+1} nous donne que $p_i(x_{i+1}) = p_{i+1}(x_{i+1}) = f(x_{i+1})$. On en déduit une condition sur f_i'
$$f_i' = f[x_i, x_{i+1}] - \frac{h_i f_i''}{3} - \frac{h_i f_{i+1}''}{6}, \text{ pour } i = 0, 1, \cdots, n-1 \tag{10}$$
- La continuité de la dérivée première aux $(n-1)$ nœuds intérieurs ($p_i'(x_{i+1}) = p_{i+1}'(x_{i+1})$) nous donne
$$f_{i+1}' = f_i' + f_i'' h_i + \frac{f_i'''}{2} h_i^2, \text{ pour } i = 0, 1, \cdots, n-2. \tag{11}$$
- La continuité de la dérivée seconde aux $(n-1)$ nœuds intérieurs ($p_i''(x_{i+1}) = p_{i+1}''(x_{i+1})$) nous donne
$$f_i''' = \frac{f_{i+1}'' - f_i''}{h_i} \text{ pour } i = 0, 1, \cdots, n-2. \tag{12}$$

En exprimant la condition (11) à l'aide des conditions (10) et (12), nous obtenons un système en f_i'', soit pour $1 \leq i \leq n-1$:

$$\frac{h_{i-1}}{h_{i-1}+h_i} f''_{i-1} + 2f''_i + \frac{h_i}{h_{i-1}+h_i} f''_{i+1} = 6f\left[x_{i-1}, x_i, x_{i+1}\right] \tag{13}$$

$$\begin{pmatrix} ? & ? & ? & \dots & ? & ? \\ \frac{h_0}{h_1+h_0} & 2 & \frac{h_1}{h_1+h_0} & 0 & \dots & 0 \\ 0 & \frac{h_1}{h_1+h_2} & 2 & \frac{h_2}{h_1+h_2} & \dots & 0 \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \frac{h_{n-2}}{h_{n-2}+h_{n-1}} & 2 & \frac{h_{n-1}}{h_{n-2}+h_{n-1}} \\ ? & ? & ? & \dots & ? & ? \end{pmatrix} \begin{pmatrix} f''_0 \\ f''_1 \\ \vdots \\ \vdots \\ f''_{n-1} \\ f''_n \end{pmatrix} = \begin{pmatrix} ? \\ 6f[x_0, x_1, x_2] \\ \vdots \\ \vdots \\ 6f[x_{n-2}, x_{n-1}, x_n] \\ ? \end{pmatrix}$$

C'est un système **tridiagonal** dont **la première ligne et la dernière ligne ne sont pas définies**. On a besoin de deux conditions supplémentaires pour remplir ces lignes, faisant intervenir f_0'' et f_n''.

Plusieurs façons de faire. On peut prendre :

- $f_0'' = a$ et $f_n'' = b \rightarrow$ On impose la valeur des dérivées secondes,
- $f_0'' = 0$ et $f_n'' = 0 \rightarrow$ Spline «naturelle»,
- $p_0'(x_0) = a$ et $p_{n-1}'(x_n) = b \rightarrow$ dérivée première imposée, cela donne en terme de f_i''

$$\begin{cases} 2f_0'' + f_1'' = \dfrac{6}{h_0}(f[x_0, x_1] - a) \\ f_{n-1}'' + 2f_n'' = \dfrac{6}{h_{n-1}}(b - f[x_{n-1}, x_n]) \end{cases}$$

- $p_0'''(x_1) = p_1'''(x_1)$ et $p_{n-2}'''(x_{n-1}) = p_{n-1}'''(x_{n-1}) \rightarrow$ Spline «not a knot »: on impose la continuité des dérivées troisièmes, cela donne en terme f_i''

$$\begin{cases} h_1 f_0'' - (h_0 + h_1) f_1'' + h_0 f_2'' = 0 \\ h_{n-1} f_{n-2}'' - (h_{n-2} + h_{n-1}) f_{n-1}'' + h_{n-2} f_n'' = 0 \end{cases}$$

- **D'autres choix sont possibles !**

Dans le cas où $h_i = h \ \forall \ i$, (points équidistants) le système (13) est simplifié

$$\frac{1}{2} f''_{i-1} + 2f''_i + \frac{1}{2} f''_{i+1} = 6f\left[x_{i-1}, x_i, x_{i+1}\right]$$

Une fois les f''_i déterminés, on exprime f'_i et f''_i en fonction des f''_i, cela nous donne

$$f'_i = f[x_i, x_{i+1}] - h_i \frac{f''_i}{3} - h_i \frac{f''_{i+1}}{6}$$
$$f'''_i = \frac{f''_{i+1} - f''_i}{h_i}$$

Pour la spline not a knot, les conditions imposées reviennent à retirer virtuellement les points x_1 et x_{n-1}. En effet, en x_1, les polynômes p_0 et p_1 coïncident ainsi que leurs trois dérivées premières. Comme p_0 et p_1 sont de degrés 3, ils s'agit en fait d'un seul et même polynôme sur les deux premiers intervalles. Idem en x_{n-1}.

Méthode pour calculer une spline cubique [15] :

- Calcul des $h_i = x_{i+1} - x_i$, $0 \leq i \leq n-1$.
- Calcul des deuxièmes différences divisées $f[x_{i-1}, x_i, x_{i+1}]$, $1 \leq i \leq n-1$.
- Résolution du système tridiagonal (13) avec la première et la dernière ligne de la matrice et du second membre adaptées aux deux conditions supplémentaires choisies. On obtient f_i'' pour $0 \leq i \leq n$.
- On calcule les autres coefficients :

$$f_i = f(x_i)$$

$$f_i' = f[x_i, x_{i+1}] - \frac{h_i f_i''}{3} - \frac{h_i f_{i+1}''}{6}$$

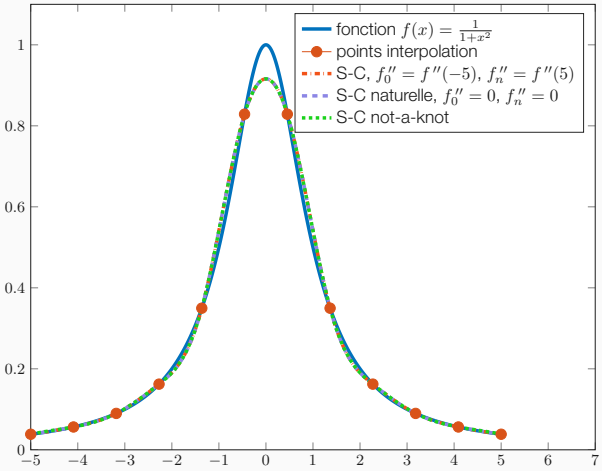
$$f_i''' = \frac{f_{i+1}'' - f_i''}{h_i}$$

- On construit les polynômes composant la spline, pour $i = 0, \cdots, n-1$:

$$p_i(x) = f_i + f_i'(x - x_i) + \frac{f_i''}{2}(x - x_i)^2 + \frac{f_i'''}{6}(x - x_i)^3$$

[15] voir `script_Spline_cubique.m` pour exemples numériques. Pour exécuter le script, la fonction `resolution_fi_pp.m` est nécessaire.

Retour sur la fonction $f(x) = \frac{1}{1+x^2}$



Splines cubiques, 12 points d'interpolation

→ Les différents types de splines permettent d'éviter le phénomène de Runge.

Je dois être capable de :

- Construire un polynôme d'interpolation par les méthodes de Vandermonde, de Lagrange et de Newton,
- Connaître le terme d'erreur, comprendre son caractère oscillatoire,
- Savoir comment choisir des points pour construire une interpolation,
- Comprendre le concept de spline, connaître les particularités et savoir vérifier toutes ses propriétés (distinguer une spline d'une interpolation cubique par morceaux),
- Construire une spline cubique naturelle.

Chapitre 6: Différentiation & Intégration numérique

Objectifs du chapitre:

1. À partir de la connaissance d'une fonction f en certains points, être capable d'obtenir une approximation de $f'(x)$ (ou $f''(x)$) en ces mêmes points.

   Exemple d'application : obtenir une approximation de la vitesse à partir d'une position calculée ponctuellement.

   $\rightarrow$ **Différentiation numérique**

2. Être capable d'obtenir une approximation de

   $$\int_a^b f(x)\,dx$$

   Exemple d'application : obtenir une approximation d'une intégrale quand il n'existe pas de primitive simple, comme par exemple

   $$\int_0^1 e^{-x^2}\,dx$$

   $\rightarrow$ **Intégration numérique**

› Différentiation numérique

But : Obtenir une approximation de f', f'' à partir de la connaissance de f en certains points.

Première approche: développements de Taylor

Retour sur le chapitre 1: Soit f une fonction régulière, alors en effectuant le développement de Taylor d'ordre 2 de f en un point x , on a

$$f(x+h) = f(x) + hf'(x) + \frac{f''(\xi_h)}{2}h^2,\ \xi_h \in [x, x+h]$$

Soit encore

$$\frac{f(x+h)-f(x)}{h} = f'(x) + \underbrace{\frac{O(h^2)}{h}}_{O(h)}$$

et donc

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

Autrement dit, à partir de la connaissance de la fonction f en deux points (x et $x+h$), on est capable d'en déduire une approximation de la dérivée en x.

Seconde approche: polynôme d'interpolation de degré n

On a

$$f(x) = p_n(x) + E_n(x), \quad x \in [x_0, x_n]$$

avec

$$E_n(x) = \frac{f^{(n+1)}(\xi_x)}{(n+1)!} \prod_{j=0}^{n} (x - x_j), \quad \xi_x \in [x_0, x_n]$$

On a donc

$$f'(x) = p'_n(x) + E'_n(x)$$

et de manière générale, pour $p \leq n$

$$f^{(p)}(x) = p_n^{(p)}(x) + E_n^{(p)}(x)$$

Remarque(s)

- Comme l'erreur peut osciller fortement avec n grand, on le gardera petit,
- Ne peut s'utiliser que pour $x \in [x_0, x_n]$ et $p \leq n$.

Regardons ce qu'il se passe pour $n = 1$, avec deux points d'interpolations $(x_0, f(x_0))$ et $(x_1, f(x_1))$. On a

$$f(x) = \underbrace{f(x_0) + f[x_0, x_1](x - x_0)}_{p_1(x)} + \underbrace{\frac{f''(\xi_x)}{2}(x - x_0)(x - x_1)}_{E_1(x)}, \ \xi_x \in [x_0, x_1]$$

Ainsi, pour tous $x \in [x_0, x_1]$

$$\begin{aligned} f'(x) &= f[x_0, x_1] + E_1'(x) \\ &= \frac{f(x_1) - f(x_0)}{x_1 - x_0} + E_1'(x) \end{aligned} \tag{14}$$

Posons $h = x_1 - x_0 > 0 \Rightarrow x_1 = x_0 + h$, l'équation (14) devient alors

$$f'(x) = \frac{f(x_0 + h) - f(x_0)}{h} + E_1'(x)$$

avec

$$E_1'(x) = \frac{f'''(\xi_x)}{2}\xi_x'(x - x_0)(x - x_1) + \frac{f''(\xi_x)}{2}(x - x_0) + \frac{f''(\xi_x)}{2}(x - x_1).$$

En x_0, on a ainsi

$$f'(x_0) = \frac{f(x_0+h) - f(x_0)}{h} + E_1'(x_0)$$

avec

$$\begin{aligned} E_1'(x_0) &= \frac{f'''(\xi_{x_0})}{2}\xi_x'(x_0 - x_0)(x_0 - x_1) + \frac{f''(\xi_{x_0})}{2}(x_0 - x_0) + \frac{f''(\xi_{x_0})}{2}(x_0 - x_1). \\ &= \frac{f''(\xi_{x_0})}{2}(x_0 - x_1) = \underbrace{-h\frac{f''(\xi_{x_0})}{2}}_{O(h)},\ \xi_{x_0} \in [x_0, x_1] \end{aligned}$$

On retrouve ainsi une expression semblable à celle obtenue par développement de Taylor en x_0

$$\begin{aligned} \boldsymbol{f'(x_0) = \frac{f(x_0+h) - f(x_0)}{h} - h\frac{f''(\xi_{x_0})}{2}} \\ = \frac{f(x_0+h) - f(x_0)}{h} + O(h) \end{aligned} \tag{15}$$

L'équation (15) s'appelle **différence finie avant d'ordre 1**.

En faisant la même chose en x_1, on obtient

$$f'(x_1) = \frac{f(x_0+h) - f(x_0)}{h} + E_1'(x_1)$$

Avec

$$E_1'(x_1) = h\frac{f''(\xi_{x_1})}{2},\ \xi_{x_1} \in [x_0, x_1]$$

et donc

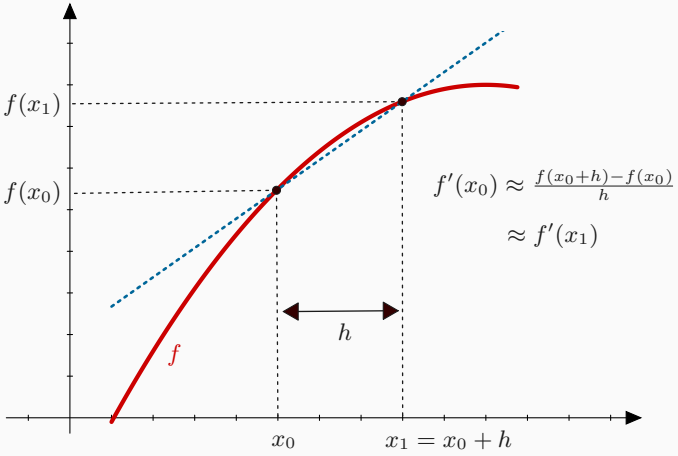
$$\begin{aligned} \boldsymbol{f'(x_1)} &= \boldsymbol{\frac{f(x_0+h) - f(x_0)}{h} + h\frac{f''(\xi_{x_1})}{2}} \\ &= \frac{f(x_1) - f(x_1 - h)}{h} + O(h) \end{aligned} \tag{16}$$

L'équation (16) s'appelle **différence finie arrière d'ordre 1**.

☢ Attention !

Bien que la quantité $\frac{f(x_0+h)-f(x_0)}{h}$ approxime à la fois la dérivée en x_0 et x_1, il faut bien garder à l'esprit que le terme d'erreur est différent aux deux endroits.

Interprétation géométrique :



Cela revient à approcher la dérivée (en x_0 ou x_1) par la pente de la droite passant par $(x_0, f(x_0))$ et $(x_1, f(x_1))$.

Est-il possible de construire une approximation de f' d'ordre 2 ?

$\rightarrow$ Par développement de Taylor : en considérant des développements faisant intervenir x, $x+h$, $x-h$, $x+2h$, $x-2h$

$\rightarrow$ Par interpolation : en considérant trois points $(x_0, f(x_0))$, $(x_1, f(x_1))$, $(x_2, f(x_2))$, et donc le polynôme d'interpolation de degré 2.

$$\begin{aligned} f'(x) &= p_2'(x) + E_2'(x) \\ &= f[x_0, x_1] + f[x_0, x_1, x_2](2x - (x_0 + x_1)) + E_2'(x) \end{aligned}$$

Et en évaluant cette expression successivement en x_0, x_1, x_2.

Dans les deux cas, on obtient les trois **approximations d'ordre 2** suivantes.

Formules de différences finies d'ordre 2 pour $f'(x)$

$x_0 = x$
$x_1 = x + h$
$x_2 = x + 2h$

$$f'(x) = \frac{-f(x+2h) + 4f(x+h) - 3f(x)}{2h} + \underbrace{\frac{h^2 f'''(\xi)}{3}}_{O(h^2)}$$

Différence avant d'ordre 2

$x_0 = x - h$
$x_1 = x$
$x_2 = x + h$

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \underbrace{\frac{h^2 f'''(\xi)}{3}}_{O(h^2)}$$

Différence centrée d'ordre 2

$x_0 = x - 2h$
$x_1 = x - h$
$x_2 = x$

$$f'(x) = \frac{3f(x) - 4f(x-h) + f(x-2h)}{2h} + \underbrace{\frac{h^2 f'''(\xi)}{3}}_{O(h^2)}$$

Différence arrière d'ordre 2

Toutes ces formules aux différences sont d'ordre 2. Les mentions avant, centrée et arrière renvoient au point où l'on calcule la dérivée et aux points utilisés pour la calculer.

Toutes les formules vues jusqu'à présent concernent f', mais **il est également possible d'approximer f'' via les mêmes approches.**

$$\begin{aligned} f''(x) &= p_2''(x) + E_2''(x) \\ &= 2f[x_0, x_1, x_2] + E_2''(x) = \frac{f(x_2) - 2f(x_1) + f(x_0)}{h^2} + E_2''(x) \end{aligned}$$

et donc

$$\begin{aligned} f''(x_0) &\approx \frac{f(x_2) - 2f(x_1) + f(x_0)}{h^2} = \frac{f(x_0 + 2h) - 2f(x_0 + h) + f(x_0)}{h^2} \\ f''(x_1) &\approx \frac{f(x_2) - 2f(x_1) + f(x_0)}{h^2} = \frac{f(x_1 + h) - 2f(x_1) + f(x_1 - h)}{h^2} \\ f''(x_2) &\approx \frac{f(x_2) - 2f(x_1) + f(x_0)}{h^2} = \frac{f(x_2) - 2f(x_2 - h) + f(x_2 - 2h)}{h^2} \end{aligned}$$

Chacune de ces expressions nous donne une formule de différences finies pour f''.

Pour analyser l'ordre d'approximation, il est plus simple de repasser par les développements de Taylor.

Considérons la formule

$$\frac{f(x+2h) - 2f(x+h) + f(x)}{h^2}$$

Et effectuons deux développements de Taylor

$$f(x+2h) = f(x) + 2hf'(x) + 4h^2 \frac{f''(x)}{2} + O(h^3) \tag{17}$$

$$f(x+h) = f(x) + hf'(x) + h^2 \frac{f''(x)}{2} + O(h^3) \tag{18}$$

Ainsi, en combinant (17) et (18), $f(x+2h) - 2f(x-h) + f(x) = h^2 f''(x) + O(h^3)$

$$\Rightarrow f''(x) = \frac{f(x+2h) - 2f(x+h) + f(x)}{h^2} + O(h)$$

$\rightarrow$ **Différence finie avant d'ordre 1.**

Considérons à présent

$$\frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

Et comme précédemment, effectuons deux développements de Taylor,

$$f(x+h) = f(x) + hf'(x) + h^2\frac{f''(x)}{2} + h^3\frac{f'''(x)}{6} + O(h^4) \tag{19}$$

$$f(x-h) = f(x) - hf'(x) + h^2\frac{f''(x)}{2} - h^3\frac{f'''(x)}{6} + O(h^4) \tag{20}$$

En additionnant (19) et (20), on a

$$f(x+h) + f(x-h) = 2f(x) + h^2 f''(x) + O(h^4)$$

Ainsi,

$$f''(x) = \frac{f(x+h) + f(x-h) - 2f(x)}{h^2} + O(h^2)$$

→ **Différence finie centrée d'ordre 2.**

☢ Attention ! (Ordre de la différence centrée)

Si on avait arrêté les développements de $f(x+h)$ et $f(x-h)$ à l'ordre 3, on aurait conclu que l'approximation était d'ordre 1. Cependant, en poussant les développements à l'ordre 4, on se rend compte que **les dérivées troisièmes s'annulent** ! **Morale:** quand les formules sont symétriques, il est prudent de pousser les développements à un ordre supérieur.

Le tableau suivant résume les approximations par différences finies possibles pour f'' ainsi que leur ordre.

Formules de différences finies pour $f''(x)$

$$f''(x) = \frac{f(x-2h) - 2f(x-h) + f(x)}{h^2} + O(h)$$

Différence arrière d'ordre 1

$$f''(x) = \frac{f(x+2h) - 2f(x+h) + f(x)}{h^2} + O(h)$$

Différence avant d'ordre 1

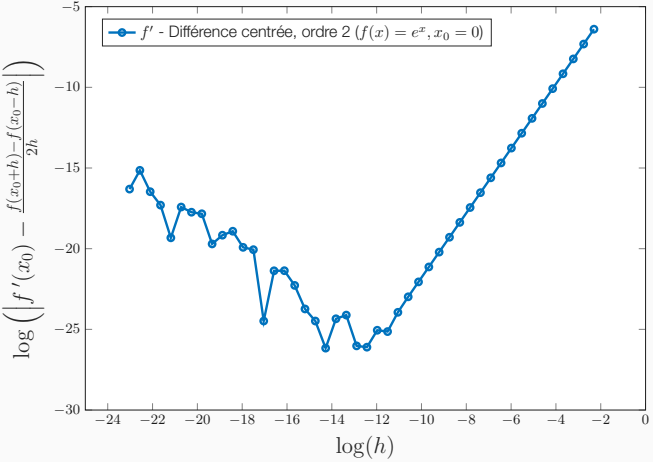
$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + O(h^2)$$

Différence centrée d'ordre 2

$$f''(x) = \frac{-f(x+2h) + 16f(x+h) - 30f(x) + 16f(x-h) - f(x-2h)}{12h^2} + O(h^4)$$

Différence centrée d'ordre 4

- Toutes ces formules de différentiation pour f' et f'' sont numériquement «dangereuses» quand $h \to 0$ (soustractions de nombres proches)[16].



Instabilité de la formule aux différences finies centrée pour f'.

[16] voir `script_instabilites_diff_finies.m` pour exemples avec différences finies avant et centrée.

BONUS ! Pas à l'examen

Il est possible d'obtenir une formule beaucoup plus stable numériquement en passant par les nombres complexes. L'idée est de considérer un «pas» complexe ih dans le développement de Taylor.

Par exemple, pour obtenir une approximation de $f'(x_0)$, considérons le développement

$$f(x_0 + ih) = f(x_0) + ihf'(x_0) - h^2 \frac{f''(x_0)}{2} - ih^3 \frac{f'''(x_0)}{6} + \cdots$$

et prenons la partie imaginaire de chaque bord,

$$\begin{aligned} \mathrm{Im}(f(x_0 + ih)) &= \mathrm{Im}\left(f(x_0) + ihf'(x_0) - h^2 \frac{f''(x_0)}{2} - ih^3 \frac{f'''(x_0)}{6} + \cdots \right) \\ &= hf'(x_0) + O(h^3) \end{aligned}$$

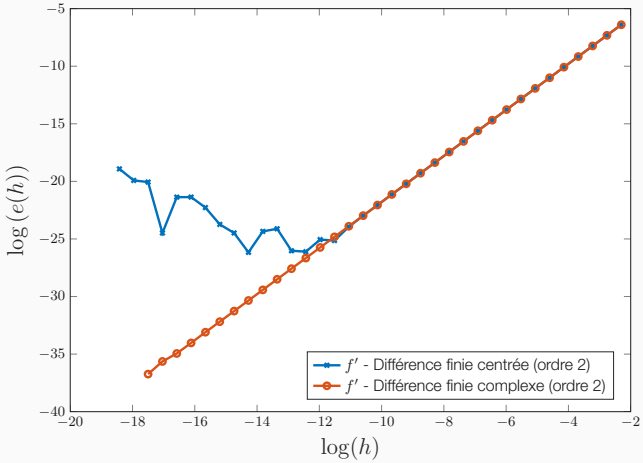
et donc

$$f'(x_0) = \frac{\mathrm{Im}(f(x_0 + ih))}{h} + O(h^2)$$

→ On obtient une formule d'ordre 2 sans soustraction dangereuse au numérateur.

BONUS ! Pas à l'examen

La formule est beaucoup plus stable numériquement comme le montre le graphique ci-dessous ($f(x) = e^x$, $x_0 = 0$).



Stabilité de la différentiation pour f' en utilisant les nombres complexes.

Il est possible d'augmenter la précision sans diminuer de manière excessive le h

$\rightarrow$ **Extrapolation de Richardson**

Soit Q une quantité que l'on cherche à approcher (quantité exacte). Supposons que l'on dispose déjà d'une approximation Q_h d'ordre n de cette quantité, *i.e*

$$Q = Q_h + O(h^n) = Q_h + \alpha h^n + \beta h^{n+1} + O(h^{n+2}) \tag{21}$$

L'extrapolation de Richardson consiste à obtenir, à partir d'une autre approximation d'ordre n, une approximation d'ordre **au moins $n+1$**. Pour cela, prenons $\frac{h}{2}$ dans (21), alors on a

$$Q = Q_{\frac{h}{2}} + \alpha \frac{h^n}{2^n} + \underbrace{\beta \frac{h^{n+1}}{2^{n+1}} + O(h^{n+2})}_{O(h^{n+1})} \tag{22}$$

En combinant (21) et (22), on a

$$Q = \frac{2^n Q_{\frac{h}{2}} - Q_h}{2^n - 1} + O(h^{n+1})$$

La quantité

$$Q_r = \frac{2^n Q_{\frac{h}{2}} - Q_h}{2^n - 1} \tag{23}$$

est donc une approximation d'ordre $n+1$ de Q. En pratique:

- On calcule donc deux approximations d'ordre n, en h et $\frac{h}{2}$,
- On calcule la quantité (23) afin d'avoir une approximation plus précise [17].

L'extrapolation de Richardson n'est pas magique et **n'empêche pas les instabilités !**

✓ L'extrapolation de Richardson ne s'applique pas qu'aux formules de différences finies, mais à n'importe quelle approximation (mais implique de connaître l'ordre de la méthode fournissant l'approximation).

Toutes les formules aux différences finies concernent la différentiation numérique. Voyons à présent **l'intégration numérique**.

[17] voir `script_richardson_diff_finies.m` pour application numérique aux différences finies.

› Intégration numérique

But : obtenir une approximation de

$$\int_a^b f(x)\,dx$$

On appelle une **formule de quadrature** ou formule d'intégration une formule permettant d'approximer l'intégrale d'une fonction. Deux familles:

- Formules de Newton-Cotes,
- Formules de Gauss-Legendre.

De manière générale, l'intégration numérique est basée sur la relation suivante

$$\int_a^b f(x)\,dx = \int_a^b p_n(x) + \int_a^b E_n(x)\,dx$$

Autrement dit, on «remplace» f par son polynôme d'interpolation de degré n.

Définition (Degré de précision)

Une formule de quadrature intégrant exactement tous les polynômes de degré inférieur ou égal à p sera dite **formule de quadrature de degré de précision p**.

Première formule: formule du trapèze.

Considérons un intervalle $[a, b]$ et prenons $n = 1$,

$$\begin{aligned}\int_a^b f(x)\,dx &= \int_a^b p_1(x)\,dx + \int_a^b E_1(x)\,dx \\ &= \int_a^b f(a) + f\,[a,b](x-a)\,dx + \int_a^b E_1(x)\,dx \\ &= (b-a)\left(\frac{f(a)+f(b)}{2}\right) + \int_a^b E_1(x)\,dx\end{aligned}$$

Théorème

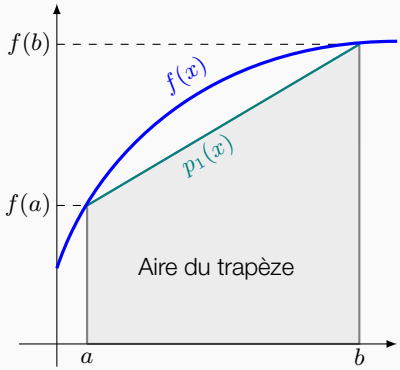
Il existe $\eta \in [a, b]$ tel que

$$\int_a^b E_1(x)\,dx = \int_a^b \frac{f''(\xi)}{2}(x-a)(x-b)\,dx = -\frac{f''(\eta)}{12}(b-a)^3$$

et donc, il existe $\eta \in [a, b]$ tel que

$$\int_a^b f(x)\,dx = (b-a)\left(\frac{f(a)+f(b)}{2}\right) - \frac{f''(\eta)}{12}(b-a)^3$$

$$\int_a^b f(x)\,dx \approx (b-a)\left(\frac{f(a)+f(b)}{2}\right)$$



Géométriquement, cela revient à approcher l'aire sous la courbe f par l'aire du trapèze formé des sommets $(a,0)$, $(b,0)$, $(a,f(a))$, $(b,f(b))$.

Si f est un polynôme de degré 0 ou 1, alors l'erreur sera nulle, car la dérivée dans le terme d'erreur s'annule $\rightarrow$ **le degré de précision de la formule des trapèzes est 1.**

Le terme d'erreur

$$-\frac{f''(\eta)}{12}(b-a)^3$$

nous indique que **l'ordre de l'approximation est 3** et que l'erreur est proportionnelle au cube de la longueur de l'intervalle.

$\rightarrow$ plus l'intervalle est grand, plus l'erreur est grande !

✓ Bien que l'approximation soit d'ordre 3, la méthode est relativement peu précise si l'intervalle $[a, b]$ est «grand».

En général, pour être plus précis, on «compose» la formule du trapèze.

Trapèzes composée: diviser l'intervalle $[a, b]$ en n sous-intervalles et appliquer la formule du trapèze sur chaque sous-intervalle.

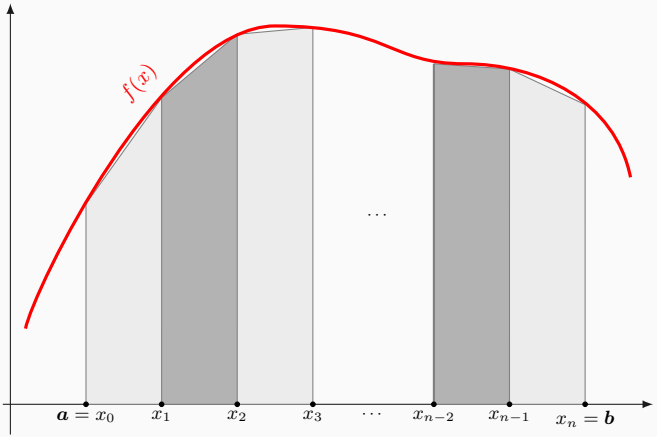


Illustration de la formule des trapèzes composée

Soit $x_0 = a < x_1 < \cdots < x_{n-1} < x_n = b$ une discrétisation de l'intervalle $[a, b]$ en n sous-intervalles $[x_i, x_{i+1}]$, sur lesquels on va appliquer la formule du trapèze.

Pour simplifier, on prendra les intervalles $[x_i, x_{i+1}]$ de même longueur, *i.e* $h = h_i = x_{i+1} - x_i = \frac{b-a}{n}$.
On a alors

$$\begin{aligned}
\int_a^b f(x)\,dx &= \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} f(x)\,dx \\
&\approx \sum_{i=0}^{n-1} (x_{i+1} - x_i) \left(\frac{f(x_{i+1}) + f(x_i)}{2} \right) \\
&= \frac{h}{2} \sum_{i=0}^{n-1} f(x_{i+1}) + f(x_i) \\
&= \frac{h}{2} (f(x_0) + 2f(x_1) + 2f(x_2) + \cdots + 2f(x_{n-1}) + f(x_n))
\end{aligned}$$

Qu'en est-il du terme d'erreur ?

On a une erreur sur chaque sous intervalle $[x_i, x_{i+1}]$, de la forme

$$-\frac{f''(\eta_i)}{12}h^3$$

Il est possible de montrer que l'erreur globale sur l'intervalle $[a, b]$ est donnée par

$$-\frac{(b-a)}{12}f''(\gamma)h^2 = -\frac{(x_n - x_0)}{12}f''(\gamma)h^2,\ \gamma \in [a, b]$$

Remarque(s)

- La formule du trapèze composée revient à intégrer une interpolation linéaire par morceaux de la fonction f.
- La formule composée est d'ordre 2, mais le degré de précision est toujours 1.

Si l'on souhaite augmenter la précision, il faut augmenter le nombre de points, cela revient à considérer un polynôme d'interpolation d'un degré plus élevé.

Deuxième formule: formule de Simpson 1/3.
Considérons toujours un intervalle $[a, b]$ et considérons cette fois $n = 2$, et donc $f(x) = p_2(x) + E_2(x)$. En intégrant sur $[a, b]$, on obtient

$$\int_a^b f(x)\,dx = \int_a^b p_2(x)\,dx + \int_a^b E_2(x)\,dx$$
$$= \int_a^b f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1)\,dx + \int_a^b E_2(x)\,dx$$

Avec $a = x_0$ et $b = x_2$. En prenant les points d'interpolations x_0, x_1, x_2 équidistants, c-à-d $\boldsymbol{x_2 - x_1 = x_1 - x_0 = h}$, on obtient

$$\int_a^b f(x)\,dx = \frac{h}{3}\left(f(x_0) + 4f(x_1) + f(x_2)\right) + \int_a^b E_2(x)\,dx$$

et donc

$$\int_{x_0}^{x_2} f(x)\,dx \approx \frac{h}{3}\left(f(x_0) + 4f(x_1) + f(x_2)\right)$$

Terme d'erreur: on montre que

$$\int_a^b E_2(x)\,dx = -\frac{h^5}{90} f^{(4)}(\eta),\ \eta \in [a,b]$$

☢ Attention !

h exprime toujours la distance entre les abscisses d'interpolation x_i et équivaut donc à la longueur de l'intervalle $[a,b]$ divisée par deux (les points sont équidistants).

- Comme la dérivée quatrième s'annule si f est un polynôme de degré inférieur ou égal à 3, **le degré de précision est 3**,
- Le terme d'erreur est proportionnel à la longueur de l'intervalle divisée par deux à la puissance 5,
- Comme pour la formule du trapèze, on a tout intérêt à diviser l'intervalle $[a,b]$ en sous-intervalles $\rightarrow$ **formule de Simpson 1/3 composée**

On divise l'intervalle $[a, b]$ en $2n$ sous intervalles (n intervalles sur lesquels on applique Simpson 1/3).
On aura au total $2n + 1$ points, $a = x_0 < x_1 < \cdots < x_{2n-1} < x_{2n} = b$ et

$$h = \frac{b-a}{2n}$$

On a alors

$$\begin{aligned}
\int_a^b f(x)\,dx &= \sum_{i=0}^{n-1} \int_{x_{2i}}^{x_{2i+2}} f(x)\,dx \\
&\approx \sum_{i=0}^{n-1} \frac{h}{3}\left(f(x_{2i}) + 4f(x_{2i+1}) + f(x_{2i+2})\right) \\
&= \frac{h}{3}(f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \dots \\
&\quad + 2f(x_{2n-2}) + 4f(x_{2n-1}) + f(x_{2n}))
\end{aligned}$$

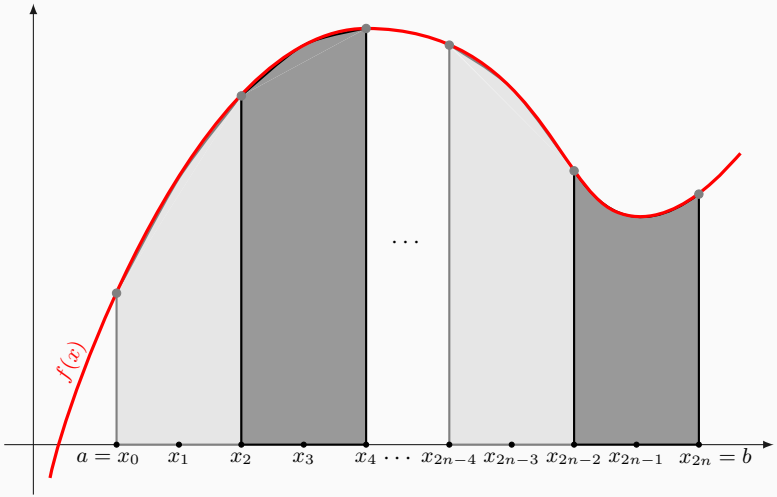


Illustration de la formule de Simpson 1/3 composée

Le terme d'erreur de la méthode de Simpson 1/3 composée est

$$-\frac{(b-a)}{180} f^{(4)}(\eta) h^4, \ \eta \in [a,b]$$

Remarque(s)

- Simpson 1/3 composée revient à intégrer une interpolation quadratique par morceaux de la fonction f,
- La formule est d'ordre 4 et le degré de précision est toujours 3.

On peut continuer à augmenter le degré du polynôme d'interpolation, en considérant $n = 3$ et donc un polynôme de degré 3. Cela nous donne la formule de Simpson 3/8 (sur un intervalle)

$$\int_{x_0}^{x_3} f(x)\,dx = \frac{3h}{8}(f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)) - 3h^5 \frac{f^{(4)}(\eta)}{80}$$

$\rightarrow$ même ordre et degré de précision que Simpson 1/3.

On pourrait continuer à augmenter le degré du polynôme d'interpolation. Mais l'oscillation du terme d'erreur de l'interpolation aura un effet sur la quadrature.

→ On se limite en général au trapèze et à Simpson 1/3.

Conclusions Newton-Cotes:

- Pour les méthodes de Newton-Cotes on a intérêt à utiliser les méthodes composées.
- Les méthodes composées **permettent un contrôle de l'erreur mais perdent un ordre.**
- On peut dresser le tableau suivant pour les méthodes **non composées**:

Méthode	nombre de points	Ordre	Précision
Trapèze	2	3	1
Simpson 1/3	3	5	3
Simpson 3/8	4	5	3

On va plutôt considérer une autre approche, cherchant à obtenir, **pour un nombre de points déterminé, un degré de précision maximal.**

On a vu que la formule du trapèze a un degré de précision de 1, et requiert l'évaluation de la fonction f aux extrémités de l'intervalle

$$\int_a^b f(x)\,dx \approx (b-a)\left(\frac{f(a)+f(b)}{2}\right)$$

On ne positionnant pas les points aux extrémités de l'intervalle, mais à l'intérieur, est-il possible de gagner en précision ?

$\rightarrow$ C'est le point de départ des **quadratures de Gauss-Legendre**.

De manière générale, une **formule de quadrature de Gauss-Legendre** consiste à trouver n paires de coefficients (w_i, t_i) tels que

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n} w_i f(t_i)$$

Les w_i sont les **poids d'intégration** et les t_i sont les **points d'intégration**.

Le but est donc de déterminer les n poids w_i et les n points t_i.

On a $2n$ inconnues, on a donc besoin de $2n$ équations...

On souhaite un degré de précision p maximal, *i.e* on veut intégrer exactement tous les polynômes de degré $\leq p$. En particulier on doit donc intégrer exactement les monômes

$$1, x, x^2, x^3, \ldots, x^p$$

En prenant $p = 2n - 1$, cela revient à dire que l'on doit intégrer exactement les monômes

$$x^k, \ k = 0, \ldots, 2n - 1$$

On impose donc à notre formule de quadrature d'intégrer exactement les monômes x^k, $k = 0, \ldots, 2n - 1 \rightarrow 2n$ équations.

La résolution des $2n$ équations non linéaires

$$\int_a^b x^k \, dx = \sum_{i=1}^{n} w_i f(t_i) = \sum_{i=1}^{n} w_i t_i^k, \quad k = 0, \ldots, 2n - 1 \tag{24}$$

nous donne les poids et points d'intégration.

Inconvénient : Les poids w_i et les points t_i dépendent de l'intervalle $[a, b]$, **si on change d'intervalle, on doit recalculer les poids et les points !**

Solution : Effectuer un changement de variable pour passer de l'intervalle $[-1, 1]$ à un intervalle $[a, b]$ quelconque.

$$x(t) = \frac{(b-a)t + a + b}{2}, \quad dx = \frac{(b-a)}{2} dt, \ t \in [-1, 1]$$

On a alors

$$\int_a^b f(x)\, dx = \frac{(b-a)}{2} \int_{-1}^{1} \underbrace{f\left(\frac{(b-a)t + a + b}{2}\right)}_{g(t)} dt \tag{25}$$

Conclusion : On calcule les poids et les points **une fois pour toute sur l'intervalle $[-1, 1]$**, et l'équation (25) nous permet de calculer l'intégrale sur l'intervalle $[a, b]$.

✓ Pour Gauss-Legendre, les poids/points sont calculés sur $[-1, 1]$, mais pour d'autres formules de quadrature (ex: Gauss-Radau) les poids/points sont calculés «une fois pour toute» sur un autre intervalle.

Voyons donc la résolution des équations (24) sur l'intervalle $[-1, 1]$.

• $\boldsymbol{n = 1}$, Gauss-Legendre à 1 point :

On cherche une expression de la forme

$$\int_{-1}^{1} g(t)\, dt \approx w_1 g(t_1)$$

et l'on impose qu'elle soit exacte pour les polynômes de degré $\leq 2n - 1 = 1$

→ Polynôme de degré 0 : prenons $g(t) = 1$

$$\int_{-1}^{1} 1\, dt = \mathbf{2 = w_1}$$

→ Polynôme de degré 1 : prenons $g(t) = t$

$$\int_{-1}^{1} t\, dt = 0 = w_1 t_1 = 2t_1 \Rightarrow \boldsymbol{t_1 = 0}$$

et donc

$$\int_{-1}^{1} g(t)\, dt = 2g(t_1) = 2g(0)$$

• $\boldsymbol{n = 2}$, Gauss-Legendre à 2 points :
On cherche une expression de la forme

$$\int_{-1}^{1} g(t)\,dt \approx w_1 g(t_1) + w_2 g(t_2)$$

et l'on impose qu'elle soit exacte pour les polynômes de degré $\leq 2n - 1 = 3$
Prenons successivement $g(t) = 1, g(t) = t, g(t) = t^2, g(t) = t^3$, on obtient le système

$$\begin{cases} \displaystyle\int_{-1}^{1} 1\,dt = 2 = w_1 + w_2 \\ \displaystyle\int_{-1}^{1} t\,dt = 0 = w_1 t_1 + w_2 t_2 \\ \displaystyle\int_{-1}^{1} t^2\,dt = \frac{2}{3} = w_1 t_1^2 + w_2 t_2^2 \\ \displaystyle\int_{-1}^{1} t^3\,dt = 0 = w_1 t_1^3 + w_2 t_2^3 \end{cases}$$

La résolution de ce système nous donne $\boldsymbol{w_1 = w_2 = 1}$, $\boldsymbol{t_1 = -\frac{1}{\sqrt{3}}}$, $\boldsymbol{t_2 = \frac{1}{\sqrt{3}}}$, et donc

$$\int_{-1}^{1} g(t)\,dt \approx g\left(-\frac{1}{\sqrt{3}}\right) + g\left(\frac{1}{\sqrt{3}}\right)$$

☢ Attention !

On n'oubliera pas de se ramener à l'intervalle $[a, b]$. Par exemple cela nous donne pour $n = 1$,

$$\begin{aligned}\int_a^b f(x)\,dx &= \frac{(b-a)}{2}\int_{-1}^{1} f\left(\frac{(b-a)t + a + b}{2}\right)dt \\ &\approx \frac{(b-a)}{2}\left(w_1 f\left(\frac{(b-a)t_1 + a + b}{2}\right)\right) \\ &\approx \frac{(b-a)}{2}\left(2f\left(\frac{a+b}{2}\right)\right)\end{aligned}$$

Conclusions/Remarques Gauss-Legendre

- La formule de Gauss-Legendre à 1 point a un degré de précision de 1, comme la formule du trapèze, qui nécessite elle, 2 points. En utilisant 2 points, la formule de Gauss-Legendre a un degré de précision de 3.
- La grande précision de ces quadratures fait en sorte qu'on ne les compose que très rarement.
- Il est possible d'établir des formules de quadratures à n points, voir p. 334 de votre manuel pour une table avec les poids et les points jusqu'à $n = 5$. Pour une formule à n points, il est possible d'obtenir la formule d'erreur suivante

$$\frac{2^{2n+1}(n!)^4}{(2n+1)((2n)!)^3} g^{(2n)}(\xi), \quad \xi \in [-1, 1]$$

Je dois être capable de :

- Comprendre comment construire une formule aux différences finies,
- Appliquer les formules aux différences finies pour approximer les dérivées d'une fonction,
- Comprendre les limitations/instabilités numériques qui peuvent apparaître dans les formules,
- Connaître la différence entre les formules avant, arrière et centrée,
- Utiliser l'extrapolation de Richardson pour augmenter l'ordre d'une formule aux différences,
- Comprendre et savoir construire une quadrature de type Newton-Cotes,
- Exploiter le terme d'erreur pour déterminer approximativement le nombre de sous intervalles d'une formule composée,
- Comprendre le concept de degré de précision,
- Utiliser les formules de Newton-Cotes et de Gauss-Legendre,
- Connaître la précision des formules de Gauss-Legendre,
- Construire une formule de quadrature, basé sur le même principe que Gauss-Legendre.

Chapitre 7: Équations différentielles

Objectif du chapitre: Résolution numérique des équations différentielles.

De manière générale, on s'intéresse à trouver une fonction $y : \mathbb{R} \to \mathbb{R}$ telle que

$$y^{(m)}(t) = f\left(t, y(t), y'(t), \ldots, y^{(m-1)}(t)\right), \quad m \geq 1, t \in [t_0, t_f]$$

Définition (EDO)

Une Équation Différentielle Ordinaire (EDO) est une relation faisant intervenir une fonction inconnue (ici y) d'une seule variable indépendante (ici t) et ses dérivées. **L'ordre** d'une EDO est déterminé par la dérivée la plus élevée de la fonction inconnue apparaissant dans l'équation.

L'unicité de la solution à ce type de problème est garantie par l'ajout de conditions supplémentaires.

En général il s'agit de conditions portant sur la valeur de l'inconnue en certains points. Le nombre de conditions nécessaires étant déterminées par l'ordre de l'EDO.

La variable indépendante t représente souvent (mais pas toujours !) le temps.

Quelques exemples d'EDO:

$$mg - k(v(t))^2 - v'(t) = 0, \text{ EDO d'ordre 1 (non linéaire)}$$
$$y^{(4)}(t) - ty(t) = 9, \text{ EDO d'ordre 4 (linéaire)}$$
$$(y''(x))^3 - y(x) = x^5, \text{ EDO d'ordre 2 (non linéaire)}$$

Dans un premier temps, nous allons nous intéresser aux EDO d'ordre 1.

Faisant toujours référence au temps, la condition que l'on ajoute pour pouvoir résoudre est souvent prise en t_0 et est dite **condition initiale**. On parlera de condition finale si la condition est prise en t_f.

$$\begin{cases} y'(t) = f(t, y(t)) \\ y(t_0) = y_0 \end{cases} \tag{26}$$

Le système (26) est dit problème de Cauchy.

Exemple:

$$\begin{cases} ty'(t) = 2y(t) + t^3 ln(t),\ t \geq 1 \\ y(1) = 0 \end{cases} \tag{27}$$

Vous avez vu (MAT-1900) un procédé de résolution «à la main»:

- Résolution de l'équation homogène (sans second membre)

$$ty'(t) - 2y(t) = 0 \Rightarrow y_h(t) = Ct^2$$

- Recherche d'une solution particulière (variation de la constante)

$$y_p(t) = \frac{t^2}{4}\left(ln(t^2) - 1\right)t^2$$

- On en déduit la forme générale des solutions

$$y(t) = y_h(t) + y_p(t) = t^2\left(C + \frac{t^2}{4}(ln(t^2) - 1)\right)$$

La condition initiale de (27) permet de fixer la constante C est d'obtenir une unique solution.

Comme toujours, on souhaite une approche de résolution systématique du problème, viable numériquement.

Naturellement au niveau numérique il n'est pas possible de construire une solution pour toutes les valeurs possibles de t.

On ne décrira $y(t)$ que pour un nombre fini de points $\rightarrow$ discrétisation de l'EDO
Plus généralement, on se donnera

- N, un nombre de pas de temps,
- t_i, $i = 0, \ldots, N-1$, les valeurs du temps où la solution sera approchée,
- y_i sera l'approximation de la solution au temps t_i, *i.e*

$$\boldsymbol{y_i \approx y(t_i)},$$

  L'erreur commise au temps t_i sera alors $|y_i - y(t_i)|$. Il n'y a pas d'erreur au temps t_0, c'est la condition initiale ! $\rightarrow$ $y(t_0) = y_0$
- $h_i = t_{i+1} - t_i$, les pas de temps. **On aura fréquemment un pas de temps constant noté $\boldsymbol{h}$.**

Première méthode : **Euler explicite**. Commençons par l'approche géométrique

On connaît la solution au temps t_0, on souhaite une approximation au temps $t_1 = t_0 + h$.

On connaît également la pente à y en t_0

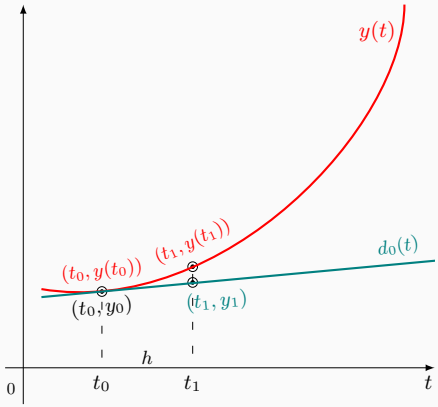
$$y'(t_0) = f(t_0, y(t_0)) = f(t_0, y_0)$$

On suit la droite passant par (t_0, y_0) de pente $f(t_0, y_0)$,

$$d_0(t) = y_0 + f(t_0, y_0)(t - t_0)$$

On prend $y_1 \approx y(t_1)$ comme étant

$$y_1 = d_0(t_1) = y_0 + hf(t_0, y_0)$$



On recommence à partir de t_1

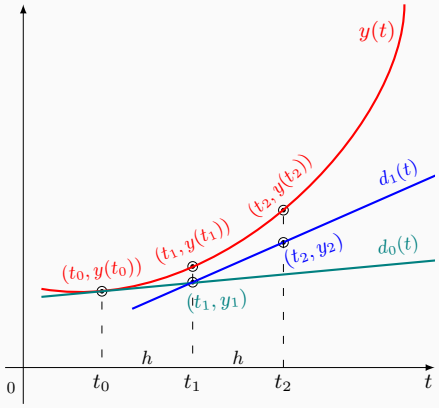
$$y'(t_1) = f(t_1, y(t_1)) \approx f(t_1, y_1)$$

on obtient

$$d_1(t) = y_1 + f(t_1, y_1)(t - t_1)$$

et l'on prend

$$y_2 = d_1(t_2) = y_1 + hf(t_1, y_1) \approx y(t_2)$$



L'erreur introduite lors du calcul de y_1 est réintroduite dans le calcul de $y_2 \rightarrow$ les erreurs vont se propager d'une itération à l'autre.

De manière générale, avec Euler explicite on obtient y_{n+1} à l'aide de la formule

$$y_{n+1} = y_n + hf(t_n, y_n)$$

Définition (Schéma explicite à un pas)

Un schéma numérique explicite à un pas est une relation de la forme

$$y_{n+1} = y_n + h\phi(t_n, y_n)$$

Autrement dit, pour calculer la solution au pas de temps suivant, on se sert uniquement de la solution au pas de temps précédent (connue).

Le schéma d'Euler explicite est un schéma à un pas avec $\phi(t, y) = f(t, y)$. Par opposition aux schémas explicites, il y a les schémas implicites

Définition (Schéma implicite)

Un schéma numérique implicite est une relation de la forme

$$y_{n+1} = y_n + h\phi(t_{n+1}, y_{n+1})$$

Cela implique de résoudre une équation à chaque pas de temps.

En résumé, pour la méthode d'Euler explicite, on a l'algorithme suivant

Algorithme d'Euler explicite, `script_euler_explicite.m`

Données :

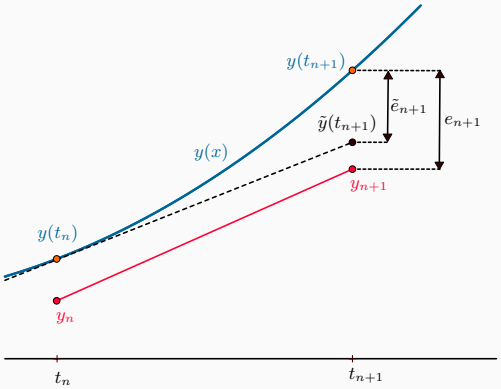
- Un pas de temps h, un nombre maximal de pas de temps N,
- Une condition initiale (t_0, y_0),

Résultat : Une variable (vecteur) contenant la solution à chaque pas de temps

1 **tant que** $0 \leq n \leq N$ **faire**
2 $\quad y_{n+1} = y_n + hf(t_n, y_n)$;
3 $\quad t_{n+1} = t_n + h$;
4 **fin**

Quid de l'erreur ?

Quand l'algorithme converge t-il vers la solution du problème ?



$$\tilde{y}(t_{n+1}) = y(t_n) + hf(t_n, y(t_n))$$

On a deux sources d'erreurs,

- Une erreur locale, $\tilde{e}_{n+1}$

$$\tilde{e}_{n+1} = y(t_n + 1) - \tilde{y}(t_{n+1})$$

- Une erreur tenant compte de la propagation

$$\tilde{y}(t_{n+1}) - y_{n+1}$$

Au final, l'erreur $e_{n+1} = y(t_{n+1}) - y_{n+1}$ est donnée par

$$e_{n+1} = (y(t_n + 1) - \tilde{y}(t_{n+1})) + (\tilde{y}(t_{n+1}) - y_{n+1})$$

On sait quantifier la première erreur. Pour Euler explicite, le développement de Taylor nous donne

$$\begin{aligned} & y(t_{n+1}) = y(t_n + h) = y(t_n) + hy'(t_n) + O(h^2) \\ \iff & y(t_{n+1}) = \underbrace{y(t_n) + hf(t_n, y(t_n))}_{\tilde{y}(t_{n+1})} + O(h^2) \\ \iff & y(t_{n+1}) - \tilde{y}(t_{n+1}) = O(h^2) \end{aligned}$$

À partir de ce développement de Taylor, on définit également l'**erreur de troncature** $\tau_{n+1}(h)$

$$\begin{aligned} \tau_{n+1}(h) = & \frac{y(t_{n+1}) - y(t_n)}{h} - f(t_n, y(t_n)) = O(h) \\ \iff \tau_{n+1}(h) = & \frac{y(t_{n+1}) - y(t_n)}{h} - y'(t_n) = O(h) \end{aligned}$$

☢ Attention !

On utilise bien $y(t_n)$ et non y_n. On cherche à mesurer l'erreur introduite par l'utilisation d'une différence finie.

L'erreur de troncature est l'erreur «locale» en t_n.

On dit également que la méthode d'Euler explicite est **consistante à l'ordre 1**, car $\tau_{n+1}(h) = O(h)$.

Cette erreur ne tient pas compte des erreurs introduites auparavant.

Le contrôle de cette erreur n'est pas suffisant pour assurer la convergence de la solution numérique vers la solution exacte. Il faut en plus que la méthode soit **stable**.

Définissons d'abord ce que l'on entend par convergence d'un schéma numérique pour une EDO.

Définition (Convergence ordre p)

On dit qu'un schéma converge à l'ordre p si

$$\max_{1 \leq n \leq N} |y(t_n) - y_n| = O(h^p)$$

La convergence à l'ordre p est le fruit de deux éléments : **la consistance** à l'ordre p **et la stabilité.**

- La consistance mesure l'erreur locale à un pas de temps t_n, c'est l'erreur commise en remplaçant y' par une différence finie à un pas donné.
  $\rightarrow$ Assure que la solution exacte des équations discrétisées tende vers la solution exacte des équations continues (quand $h \rightarrow 0$).
- La stabilité d'un schéma assure que l'erreur de propagation d'un pas de temps à l'autre, est bornée. L'étude de la stabilité n'est pas au programme du cours.

Théorème (de Lax)

Si un schéma numérique est **stable et consistant** à l'ordre p alors il est convergent à l'ordre p.

Quelques remarques supplémentaires sur la stabilité. La stabilité dépend:

- De l'EDO considérée (de la fonction f),
- Du schéma numérique. Un schéma numérique peut être :
  - conditionnellement stable, *i.e* stable sous une condition faisant intervenir le pas de discrétisation h,
  - inconditionnellement stable, *i.e* stable peu importe le pas de discrétisation h,
  - instable, *i.e* la propagation d'erreur dégradera (beaucoup) la solution à chaque pas de temps.

Le développement de Taylor ouvre la voie à des schémas d'ordre plus élevé. Voyons la méthode de Taylor du second ordre.

Reprenons le développement de Taylor, et poussons d'un ordre de plus

$$\begin{aligned} y(t_{n+1}) &= y(t_n) + hy'(t_n) + \frac{y''(t_n)h^2}{2} + O(h^3) \\ &= y(t_n) + hf(t_n, y(t_n)) + \frac{f'(t_n, y(t_n))h^2}{2} + O(h^3) \end{aligned} \tag{28}$$

Or,

$$\begin{aligned} f'(t, y(t)) &= \frac{\partial f(t, y(t))}{\partial t} + \frac{\partial f(t, y(t))}{\partial y} y'(t) \\ &= \frac{\partial f(t, y(t))}{\partial t} + \frac{\partial f(t, y(t))}{\partial y} f(t, y(t)) \end{aligned}$$

et donc

$$y(t_{n+1}) \approx y(t_n) + hf(t_n, y(t_n)) + \frac{h^2}{2}\left(\frac{\partial f(t_n, y(t_n))}{\partial t} + \frac{\partial f(t_n, y(t_n))}{\partial y} f(t_n, y(t_n))\right)$$

En remplaçant $y(t_n)$ par y_n, on obtient l'algorithme de Taylor d'ordre 2.

Algorithme de Taylor d'ordre 2, pseudo-code

Données :

- Un pas de temps h, un nombre maximal de pas de temps N,
- Une condition initiale (t_0, y_0),

Résultat : Variable (vecteur) contenant la solution à chaque pas de temps.

1 **tant que** $0 \leq n \leq N$ **faire**

2 $\quad y_{n+1} = y_n + hf(t_n, y_n) + \frac{h^2}{2}\left(\frac{\partial f(t_n, y_n)}{\partial t} + \frac{\partial f(t_n, y_n)}{\partial y} f(t_n, y_n)\right)$;

3 $\quad t_{n+1} = t_n + h$;

4 **fin**

- Comme Euler explicite, c'est un schéma à un pas, avec

$$\phi(t_n, y_n) = f(t_n, y_n) + \frac{h}{2}\left(\frac{\partial f(t_n, y_n)}{\partial t} + \frac{\partial f(t_n, y_n)}{\partial y} f(t_n, y_n)\right)$$

- On montre que $\tau_{n+1}(h) = O(h^2)$, le schéma est consistant à l'ordre 2.

Inconvénient: nécessite le calcul des dérivées de la fonction f.

On souhaite avoir le même ordre (voir plus) sans avoir à calculer les dérivées de f. Reprenons (encore) notre développement de Taylor,

$$\begin{aligned} y(t_{n+1}) =& y(t_n) + hf(t_n, y(t_n)) \\ &+ \frac{h^2}{2}\left(\frac{\partial f(t_n, y(t_n))}{\partial t} + \frac{\partial f(t_n, y(t_n))}{\partial y} f(t_n, y(t_n))\right) + O(h^3) \end{aligned}$$

L'idée est de chercher à remplacer les dérivées de f par des évaluations de f en certains points bien choisis.

On cherche des poids α_1 et α_2 et des points β_1 et β_2 tels que l'expression

$$y(t_{n+1}) = y(t_n) + \alpha_1 hf(t_n, y(t_n)) + \alpha_2 hf(t_n + \beta_1 h, y(t_n) + \beta_2 h) \tag{29}$$

soit une **approximation d'ordre 3**. Un développement de Taylor en deux variables nous donne

$$f(t_n + \beta_1 h, y(t_n) + \beta_2 h) = f(t_n, y(t_n)) + \beta_1 h \frac{\partial f(t_n, y(t_n))}{\partial t} + \beta_2 h \frac{\partial f(t_n, y(t_n))}{\partial y} + O(h^2)$$

et donc

$$\begin{aligned} y(t_{n+1}) =& y(t_n) + (\alpha_1 + \alpha_2) hf(t_n, y(t_n)) \\ &+ \alpha_2 \beta_1 h^2 \frac{\partial f(t_n, y(t_n))}{\partial t} + \alpha_2 \beta_2 h^2 \frac{\partial f(t_n, y(t_n))}{\partial y} + O(h^3) \end{aligned}$$

On a deux expressions d'ordre 3:

$$y(t_{n+1}) = y(t_n) + hf(t_n, y(t_n)) + \frac{h^2}{2}\left(\frac{\partial f(t_n, y(t_n))}{\partial t} + \frac{\partial f(t_n, y(t_n))}{\partial y} f(t_n, y(t_n))\right) + O(h^3)$$

$$y(t_{n+1}) = y(t_n) + (\alpha_1 + \alpha_2)hf(t_n, y(t_n)) + \alpha_2\beta_1 h^2 \frac{\partial f(t_n, y(t_n))}{\partial t} + \alpha_2\beta_2 h^2 \frac{\partial f(t_n, y(t_n))}{\partial y} + O(h^3)$$

Par identifications des coefficients respectifs, on obtient le système

$$\begin{cases} 1 = \alpha_1 + \alpha_2 \\ \frac{1}{2} = \alpha_2\beta_1 \\ \frac{f(t_n, y(t_n))}{2} = \alpha_2\beta_2 \end{cases}$$

On obtient 3 équations pour 4 inconnues → pas de solution unique, il y a donc plusieurs combinaisons possibles ! On va voir deux solutions populaires.

Méthode d'Euler modifiée: On prend $\alpha_1 = \alpha_2 = \frac{1}{2}$, $\beta_1 = 1$ et $\beta_2 = f(t_n, y(t_n))$. Cela nous donne dans (29)

$$y(t_{n+1}) = y(t_n) + \frac{h}{2}\Big(f(t_n, y(t_n)) + f\big(t_n + h, y(t_n) + hf(t_n, y(t_n))\big)\Big) + O(h^3)$$

Ne nécessite plus l'évaluation des dérivées ! Au niveau discret, en remplaçant $y(t_n)$ par son approximation y_n, on obtient

$$y_{n+1} = y_n + \frac{h}{2}\Big(f(t_n, y_n) + f\big(t_n + h, \underbrace{y_n + hf(t_n, y_n)}_{\text{Euler explicite}}\big)\Big)$$

À chaque itération (pas de temps) on décompose souvent le calcul en deux étapes:

1. Calcul de $\hat{y} = y_n + hf(t_n, y_n) \rightarrow$ prédiction,
2. Calcul de $y_{n+1} = y_n + \frac{h}{2}\Big(f(t_n, y_n) + f\big(t_n + h, \hat{y}\big)\Big) \rightarrow$ correction.

Au final, on a l'algorithme suivant,

Algorithme d'Euler modifié, pseudo-code

Données :

- Un pas de temps h, un nombre maximal de pas de temps N,
- Une condition initiale (t_0, y_0),

Résultat : Une variable (vecteur) contenant la solution à chaque pas de temps

1 **tant que** $0 \leq n \leq N$ **faire**

2 $\quad \hat{y} = y_n + hf(t_n, y_n)$;

3 $\quad y_{n+1} = y_n + \dfrac{h}{2}\Big(f(t_n, y_n) + f\big(t_n + h, \hat{y}\big)\Big)$;

4 $\quad t_{n+1} = t_n + h$;

5 **fin**

Méthode du point milieu: On prend $\alpha_1 = 0$, $\alpha_2 = 1$, $\beta_1 = \frac{1}{2}$ et $\beta_2 = \frac{f(t_n, y(t_n))}{2}$. Cela nous donne dans (29)

$$y(t_{n+1}) = y(t_n) + hf\left(t_n + \frac{h}{2}, y_n + \frac{hf(t_n, y(t_n))}{2}\right) + O(h^3)$$

Au niveau discret, en remplaçant $y(t_n)$ par son approximation y_n, on obtient

$$y_{n+1} = y_n + hf\left(t_n + \frac{h}{2}, y_n + \frac{hf(t_n, y_n)}{2}\right)$$

À chaque itération (pas de temps) on décompose souvent le calcul en deux étapes:

1. Calcul de $k_1 = hf(t_n, y_n)$,
2. Calcul de $y_{n+1} = y_n + hf\left(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right)$.

Au final, on a l'algorithme dit du point milieu,

Algorithme du point milieu, pseudo-code

Données :

- Un pas de temps h, un nombre maximal de pas de temps N,
- Une condition initiale (t_0, y_0),

Résultat : Une variable (vecteur) contenant la solution à chaque pas de temps

1 **tant que** $0 \leq n \leq N$ **faire**

2 $\quad k_1 = hf(t_n, y_n)$;

3 $\quad y_{n+1} = y_n + hf\left(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right)$;

4 $\quad t_{n+1} = t_n + h$;

5 **fin**

D'autres combinaisons de coefficients α_1, α_2, β_1 et β_2 sont possibles. Ces deux schémas numériques (Euler modifié et point milieu) sont dites méthodes de **Runge-Kutta d'ordre 2**.

Nous allons continuer à augmenter l'ordre du développement de Taylor (28), afin d'obtenir des schémas convergeant à l'ordre 4.

On pousse le développement de Taylor à l'ordre 5.

On introduit 4 poids et 6 points, un raisonnement similaire à celui des méthodes de Runge-Kutta d'ordre 2 nous donne un système de 8 équations pour 10 inconnues.

Parmi toute les possibilités pour le choix des coefficients, un choix est particulièrement populaire. Voici le schéma «RK 4». À chaque itération (pas de temps), on effectue

1. Calcul de $k_1 = hf(t_n, y_n)$,
2. Calcul de $k_2 = hf\left(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right)$,
3. Calcul de $k_3 = hf\left(t_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right)$,
4. Calcul de $k_4 = hf(t_n + h, y_n + k_3)$,
5. Calcul de $y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$.

Cela nous donne l'algorithme de Runge Kutta d'ordre 4,

Algorithme de Runge-Kutta ordre 4, pseudo-code `script_RK4.m`

Données :

- Un pas de temps h, un nombre maximal de pas de temps N,
- Une condition initiale (t_0, y_0),

Résultat : Une variable (vecteur) contenant la solution à chaque pas de temps

tant que $0 \leq n \leq N$ **faire**

$k_1 = hf(t_n, y_n)$;

$k_2 = hf\left(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right)$;

$k_3 = hf\left(t_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right)$;

$k_4 = hf(t_n + h, y_n + k_3)$;

$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$;

$t_{n+1} = t_n + h$;

fin

Le nombre et la complexité des calculs augmentent avec l'ordre:

- Euler explicite : 1 évaluation de la fonction f à chaque itération,
- RK 2 : 2 évaluations de la fonction f à chaque itération,
- RK 4 : 4 évaluations de la fonction f à chaque itération.

Cependant plus l'ordre est grand plus l'erreur diminue vite quand h diminue [18].

Rappel

- Si la convergence est d'ordre 2, alors l'erreur diminue par un facteur 2^2 quand h est divisé par deux,
- Si la convergence est d'ordre 4, alors l'erreur diminue par un facteur $2^4 = 16$ quand h est divisé par deux,

De plus, pour h fixé, l'erreur sera plus petite avec une méthode d'ordre 2 qu'avec une méthode d'ordre 1, et ainsi de suite.

Y'a-t-il un équilibre à respecter entre effort de calcul et précision ? Pour éclaircir ce point, effectuons une comparaison.

[18] voir `script_comparaison_ordre_EDO.m` pour exemples numériques.

À «coût» égal (nombre d'évaluation de la fonction f), comparons la précision pour Euler explicite, Euler modifié et Runge-Kutta ordre 4 sur un exemple.

L'EDO est la suivante

$$\begin{cases} y'(t) = y + t + 1, \ t \in [0, 1] \\ y(t_0) = 1 \end{cases}$$

40 évaluations de la fonction f représente:

- 40 pas de temps avec Euler explicite $\Rightarrow h = \frac{1}{40}$
- 20 pas de temps avec Euler modifié $\Rightarrow h = \frac{1}{20}$
- 10 pas de temps avec Runge-Kutta ordre 4 $\Rightarrow h = \frac{1}{10}$

Méthode	h	N	$\max_{1\leq n\leq N} \lvert y(t_n) - y_n \rvert$
Euler explicte (ordre 1)	$1/40$	40	9.9654×10^{-2}
Euler modifiée (ordre 2)	$1/20$	20	3.2723×10^{-3}
RK 4 (ordre 4)	$1/10$	10	6.25297×10^{-6}

Moralité: Pour un coût de calcul à peu prés équivalent, il est préférable de prendre un pas de temps «grand» avec un schéma d'ordre élevé plutôt qu'un pas de temps petit avec un schéma d'ordre moins élevé !

Pour une précision comparable à RK4 avec Euler explicite il faudrait $10\,000$ fois plus de pas de temps !

La méthode de Runge-Kutta d'ordre 4 est la méthode la plus utilisée en raison de sa grande précision.

Les méthodes vues jusqu'à présent concernent les EDO d'ordre 1.

Considérons à présent une équation d'ordre m, avec m conditions initiales,

$$\begin{cases} y^{(m)}(t) = f\left(t, y(t), y'(t), \ldots, y^{(m-1)}(t)\right), \\ y(t_0) = y_0 \\ y'(t_0) = y_1 \\ y''(t_0) = y_2 \\ \vdots \\ y^{(m-1)}(t_0) = y_{m-1} \end{cases} \tag{30}$$

Plan de bataille:

1. Résolution d'un système d'EDO d'ordre 1 (en utilisant les méthodes déjà vues).
2. Ré-écriture de (30) en m EDO d'ordre 1 $\rightarrow$ système d'EDO d'ordre 1,

• **Résolution d'un système d'EDO d'ordre 1**:

Plaçons nous dans le cas général, consistant à trouver $y_1, y_2, \ldots, y_m$ tels que

$$\begin{cases} y_1'(t) = f_1(t, y_1(t), y_2(t), \ldots, y_m(t)) & y_1(t_0) = y_{1,0} \\ y_2'(t) = f_2(t, y_1(t), y_2(t), \ldots, y_m(t)) & y_2(t_0) = y_{2,0} \\ \vdots & \vdots \\ y_{m-1}'(t) = f_{m-1}(t, y_1(t), y_2(t), \ldots, y_m(t)) & y_{m-1}(t_0) = y_{m-1,0} \\ y_m'(t) = f_m(t, y_1(t), y_2(t), \ldots, y_m(t)) & y_m(t_0) = y_{m,0} \end{cases}$$

Que l'on ré-écrit sous forme vectorielle

$$\begin{cases} \vec{Y}'(t) = \vec{F}(t, \vec{Y}(t)) \\ \vec{Y}(t_0) = \vec{Y_0} \end{cases}$$

Où

$$\begin{aligned} &\overrightarrow{Y(t)} = (y_1(t), y_2(t), \ldots, y_m(t)), \\ &\vec{F}(t, \vec{Y}(t)) = (f_1(t, \vec{Y}(t)), f_2(t, \vec{Y}(t)), \ldots, f_m(t, \vec{Y}(t))) \\ &\vec{Y_0} = (y_{1,0}, y_{2,0}, \ldots, y_{m,0}). \end{aligned}$$

Les méthodes vues dans ce chapitre se généralisent bien aux systèmes.

On note $\vec{Y}_i = (y_{1,i}, y_{2,i}, \dots, y_{m,i}) \approx \vec{Y}(t_i) = (y_1(t_i), y_2(t_i), \dots, y_m(t_i))$

• **Euler Explicite:**

$$\vec{Y}_{n+1} = \vec{Y}_n + h\vec{F}(t_n, \vec{Y_n})$$

Soit encore

$$\begin{cases} y_{1,n+1} = y_{1,n} + hf_1(t_n, y_{1,n}, y_{2,n}, \dots, y_{m,n}) \\ y_{2,n+1} = y_{2,n} + hf_2(t_n, y_{1,n}, y_{2,n}, \dots, y_{m,n}) \\ \qquad \vdots \\ y_{m,n+1} = y_{m,n} + hf_m(t_n, y_{1,n}, y_{2,n}, \dots, y_{m,n}) \end{cases}$$

À chaque pas de temps, on fait m Euler explicite «standard» avant de passer au pas de temps suivant.

On en déduit l'algorithme d'Euler explicite pour un système,

Algorithme d'Euler explicite pour les systèmes d'EDO, pseudo-code

Données :

- Un pas de temps h, un nombre maximal de pas de temps N,
- Une condition initiale $\overrightarrow{Y_0} = (y_{1,0}, y_{2,0}, \dots, y_{m,0})$.

Résultat : Une variable (matrice) contenant la solution à chaque pas de temps, pour chaque y_i.

```
tant que 0 ≤ n ≤ N faire
    tant que 1 ≤ i ≤ m faire
        y_{i,n+1} = y_{i,n} + h f_i(t_n, y_{1,n}, y_{2,n}, ..., y_{m,n}) ;
    fin
    t_{n+1} = t_n + h ;
fin
```

On peut faire la même chose pour Euler modifié, ou la méthode du Point milieu. Voyons directement la généralisation de la méthode de Runge-Kutta d'ordre 4.

• **Runge-Kutta ordre 4 pour les systèmes:** À chaque pas de temps, on va calculer $4m$ coefficients $k_{i,1}$, $k_{i,2}$, $k_{i,3}$, $k_{i,4}$, $i = 1, \ldots, m$. On part de $(t_0, \vec{Y_0})$:

- On détermine $\vec{k}_1 = (k_{i,1})_{1 \leq i \leq m}$,

$$k_{i,1} = hf_i(t_n, y_{1,n}, y_{2,n}, \cdots, y_{m,n})$$

- On détermine $\vec{k}_2 = (k_{i,2})_{1 \leq i \leq m}$,

$$k_{i,2} = hf_i(t_n + \frac{h}{2}, y_{1,n} + \frac{k_{1,1}}{2}, y_{2,n} + \frac{k_{2,1}}{2}, \cdots, y_{m,n} + \frac{k_{m,1}}{2})$$

- On détermine $\vec{k}_3 = (k_{i,3})_{1 \leq i \leq m}$,

$$k_{i,3} = hf_i(t_n + \frac{h}{2}, y_{1,n} + \frac{k_{1,2}}{2}, y_{2,n} + \frac{k_{2,2}}{2}, \cdots, y_{m,n} + \frac{k_{m,2}}{2})$$

- On détermine $\vec{k}_4 = (k_{i,4})_{1 \leq i \leq m}$,

$$k_{i,4} = hf_i(t_n + h, y_{1,n} + k_{1,3}, y_{2,n} + k_{2,3}, \cdots, y_{m,n} + k_{m,3})$$

- Et finalement, on calcule

$$y_{i,n+1} = y_{i,n} + \frac{1}{6}(k_{i,1} + 2k_{i,2} + 2k_{i,3} + k_{i,4})$$

Il faut calculer les m constantes $k_{i,1}$ avant de passer au calcul des constantes $k_{i,2}$ et ainsi de suite.

- **Revenons à présent aux équations d'ordre m :**

$$\begin{cases} y^{(m)}(t) = f\left(t, y(t), y'(t), \ldots, y^{(m-1)}(t)\right), \\ y(t_0) = y_0 \\ y'(t_0) = y_1 \\ y''(t_0) = y_2 \\ \vdots \\ y^{(m-1)}(t_0) = y_{m-1} \end{cases}$$

On va transformer cette EDO d'ordre m en un système de m EDO d'ordre 1, pour pouvoir utiliser les méthodes vues précédemment.

Pour écrire l'EDO d'ordre m comme un système de m EDO d'ordre 1, posons

$$
\begin{aligned}
u_0(t) &= y(t) \\
u_1(t) &= y'(t) \\
u_2(t) &= y''(t) \\
&\vdots \\
u_{m-1}(t) &= y^{(m-1)}(t) \\
(u_m(t) &= y^{(m)}(t))
\end{aligned}
$$

On a alors

$$
\begin{aligned}
u_0'(t) &= y'(t) = u_1(t) \\
u_1'(t) &= y''(t) = u_2(t) \\
&\vdots \\
u_{m-2}'(t) &= y^{(m-1)}(t) = u_{m-1}(t) \\
u_{m-1}'(t) &= y^{(m)}(t) = f\left(t, u_0(t), u_1(t), \ldots, u_{m-1}(t)\right)
\end{aligned}
$$

Avec ces notations, l'EDO d'ordre m (30) s'écrit comme un système de m d'EDO d'ordre 1

$$\begin{cases} u_0'(t) = u_1(t) \ (= y'(t)) & u_0(t_0) = y_0 \\ u_1'(t) = u_2(t) \ (= y''(t)) & u_1(t_0) = y_1 \\ u_2'(t) = u_3(t) \ (= y'''(t)) & u_2(t_0) = y_2 \\ \vdots & \vdots \\ u_{m-2}'(t) = u_{m-1}(t) \ (= y^{(m-1)}(t)) & u_{m-2}(t_0) = y_{m-2} \\ u_{m-1}'(t) = f\left(t, u_0(t), \ldots, u_{m-1}(t)\right) \ (= y^{(m)}(t)) & u_{m-1}(t_0) = y_{m-1} \end{cases} \tag{31}$$

✓ Ne pas oublier que c'est surtout $u_0(t) = y(t)$ qui nous intéresse, c'est la solution de l'EDO !

La résolution d'une EDO d'ordre m est donc un cas particulier de la résolution d'un système d'EDO d'ordre 1 avec

$$\begin{aligned}\vec{Y}(t) &= (u_0(t), u_1(t), \dots, u_{m-1}(t)),\\ \vec{F}(t, \vec{Y}(t)) &= (u_1(t), u_2(t), \dots, u_{m-1}(t), \underbrace{f\left(t, u_0, u_1, \dots, u_{m-1}\right)}_{y^{(m)}(t)})\\ \vec{Y}_0 &= (y_0, y_1, y_2, \dots, y_{m-1})\end{aligned}$$

En conclusion, la stratégie de résolution d'une EDO d'ordre m est la suivante[19]:

- Ré-écriture de l'EDO d'ordre m (30) en un système d'EDO d'ordre 1 de la forme (31),
- Résolution du système à l'aide de l'une des méthodes vue au début du chapitre, adaptée à la résolution d'un système d'équations.

[19]Pour un exemple numérique, voir le problème de chute libre `script_parachutiste.m` (nécessite `affiche_bonhomme.m`). Pour visualiser le résultat, cliquer **ici**.

Je dois être capable de :

- Comprendre le concept de schéma en temps,
- Comprendre la construction d'un schéma basé sur le développement de Taylor,
- Comprendre l'origine du concept de famille de schéma pour Runge-Kutta,
- Appliquer les différents schémas pour une EDO d'ordre 1,
- Comprendre le concept d'erreur locale ainsi que l'ordre de convergence,
- Transformer une EDO d'ordre supérieur en un système d'EDO,
- Appliquer les schémas sur un système d'EDO d'ordre 1,
- Comprendre qu'avec les méthodes vues, pour avoir stabilité du schéma, le pas de temps est important et ne doit pas être choisit n'importe comment.

Autre approche de résolution: on intègre l'EDO sur l'intervalle $[t_n, t_{n+1}]$

$$\int_{t_n}^{t_{n+1}} y'(t)dt = \int_{t_n}^{t_{n+1}} f(t, y(t))\, dt$$

$$\iff y(t_{n+1}) - y(t_n) = \int_{t_n}^{t_{n+1}} f(t, y(t))\, dt$$

La question est comment intégrer le terme de droite $\rightarrow$ on utilise l'interpolation de Newton de la fonction f.

Dans la suite, on note $f_n = f(t_n, y_n) \approx f(t_n, y(t_n))$.

L'idée pour construire le polynôme d'interpolation est d'utiliser des quantités connues provenant des pas de temps précédents.

Par exemple, à partir des 3 pas de temps précédents, la table de différences divisées s'écrit

Table de différences divisées

t_n	f_n			
		$f[t_n, t_{n-1}]$	$f[t_n, t_{n-1}, t_{n-2}]$	
t_{n-1}	f_{n-1}			
		$f[t_{n-1}, t_{n-2}]$		$f[t_n, t_{n-1}, t_{n-2}, t_{n-3}]$
t_{n-2}	f_{n-2}		$f[t_{n-1}, t_{n-2}, t_{n-3}]$	
		$f[t_{n-2}, t_{n-3}]$		
t_{n-3}	f_{n-3}			

Le polynôme d'interpolation de degré 3 est:

$$p_3(t) = f_n + f[t_n, t_{n-1}](t - t_n) + f[t_n, t_{n-1}, t_{n-2}](t - t_n)(t - t_{n-1}) + f[t_n, t_{n-1}, t_{n-3}](t - t_n)(t - t_{n-1})(t - t_{n-2})$$

En utilisant des polynômes de différents degrés, on obtient différents schémas.

- Degré 0, $p_0(t) = f_n = f(t_n, y_n) \approx f(t, y(t))$ et on obtient

$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} f_n dt = y_n + hf(t_n, y_n)$$

  On retrouve Euler explicite.

- Degré 1, $p_1(t) = f_n + f[t_n, t_{n-1}](t - t_n) \approx f(t, y(t))$ et on obtient

$$\begin{aligned} y_{n+1} &= y_n + \int_{t_n}^{t_{n+1}} f_n + f[t_n, t_{n-1}](t - t_n)\, dt \\ &= y_n + \frac{h}{2}(3f(t_n, y_n) - f(t_{n-1}, y_{n-1})) \end{aligned}$$

  C'est une méthode à deux pas, on se sert de y_n et y_{n-1} pour calculer y_{n+1}.

On peut continuer ainsi de suite, chaque fois que l'on augmente de 1 le degré du polynôme d'interpolation, on augmente de 1 l'ordre de convergence.

BONUS ! Pas à l'examen

Toutes ces méthodes sont toujours explicites, seuls les pas de temps précédents sont nécessaires.

En utilisant d'autres points pour construire la table de différence divisées, en particulier en ajoutant (t_{n+1}, y_{n+1}) pour construire le polynôme d'interpolation, on obtient des méthodes dites **implicites**.

- Degré 0 (Euler implicite [20]), $p_0(t) = f_{n+1} = f(t_{n+1}, y_{n+1}) \approx f(t, y(t))$

$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} f_{n+1}\, dt = y_n + hf(t_{n+1}, y_{n+1})$$

- Degré 1, $p_1(t) = f_{n+1} + f[t_{n+1}, t_n](t - t_{n+1}) \approx f(t, y(t))$

$$\begin{aligned} y_{n+1} &= y_n + \int_{t_n}^{t_{n+1}} f_{n+1} + f[t_{n+1}, t_n](t - t_{n+1})\, dt \\ &= y_n + \frac{h}{2}(f(t_{n+1}, y_{n+1}) + f(t_n, y_n)) \end{aligned}$$

Pour les méthodes implicites, cela implique de résoudre une équation à chaque pas de temps.

[20] voir `script_euler_implicite.m`