IN1-Cours

# Calcul de la dérivée d'une fonction

On apprend dans ce cours à **calculer numériquement la dérivée d'une fonction à valeur dans** $\mathbb{R}$.

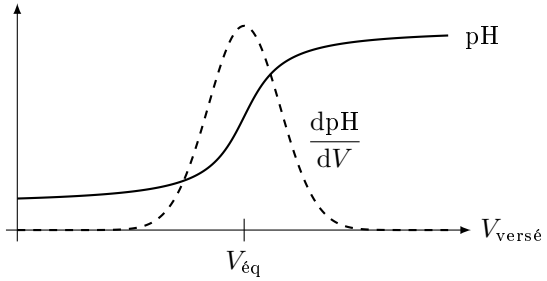
## Table des matières



## 1 Exemples

Mentionnons quelques exemples de situations dans lesquelles le calcul de la dérivée d'une fonction numérique intervient, en physique et en chimie.

- En TP de physique, vous étudiez un circuit $RC$ série, et vous faites l'acquisition avec Latis Pro de l'évolution temporelle de la tension $u_C$ aux bornes du condensateur pendant la charge de celui-ci. On vous demande ensuite de tracer l'intensité qui traverse le circuit. Vous savez qu'elle vaut
$$i = C\,\frac{\mathrm{d}u_C}{\mathrm{d}t}$$
Il vous suffit donc de dériver la tension précédemment acquise (et de la multiplier par $C$).
- En TP de chimie, vous réalisez un titrage pH-métrique d'un acide par une base, et vous tracez point par point le pH de la solution pour des valeurs successives de volume de solution titrante versé. À l'équivalence, cette courbe passe par un point d'inflexion (la dérivée seconde y est nulle), donc la dérivée est extrémale, en fait maximale. Vous pouvez donc chercher à obtenir la dérivée de votre courbe puis regarder où celle-ci passe par un maximum : c'est le point d'équivalence.




## 2 Représentation numérique d'une fonction

En python, on peut définir des fonctions (pour rappel, soit avec l'instruction `def`, soit avec le mot clé `lambda` pour les fonctions « courtes »).

Néanmoins, lorsqu'on travaille dans un contexte mathématique, physique ou chimique, ou autre... on cherche souvent à **tracer les fonctions**. Et pour cela, on les représente comme des listes de nombres : pour un jeu discret d'abscisses $x_i$, on dresse une liste des $f(x_i)$ et on trace les couples $(x_i, f(x_i))$ point par point (qu'on peut éventuellement relier entre eux pour donner une impression de continuité).

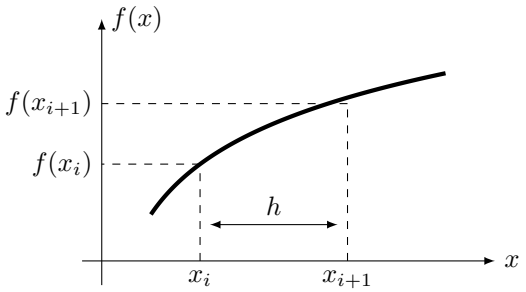
C'est sous cette forme « point par point » que l'on va chercher à déterminer la dérivée d'une fonction.

# 3 Obtention de la dérivée d'une fonction par différence finie

Vous savez que la dérivée en mathématiques est définie comme la limite du taux de variation

$$f'(x) = \lim_{h\to 0} \frac{f(x+h) - f(x)}{h}$$

La notion de limite est évidemment totalement absente en python (ce n'est pas un langage de mathématiques formelles...). L'idée est donc simplement de ne pas prendre la limite ! Mais de seulement prendre $h$ « petit ».




Dans ce cas, pour un jeu d'abscisses $x_i$ réparties uniformément par **pas** de $h$ (c'est-à-dire que $x_{i+1} - x_i = h$ pour tout $i$), la dérivée de la fonction $f$ est donnée par

$$f'(x_i) \approx \frac{f(x_{i+1}) - f(x_i)}{h} \qquad \text{avec} \qquad x_{i+1} = x_i + h$$

Cette approche pour le calcul de la dérivée est appelée **méthode de différence finie** (« finie » signifie en fait qu'on ne prend pas la limite : « $h$ fini » doit être compris comme « $h$ non nul »).

**Proposition d'implémentation.** On peut implémenter le calcul d'une dérivée par différence finie avec le programme suivant. `f` est la fonction qu'on souhaite dériver, `a` et `b` les bornes entre lesquelles on souhaite obtenir la dérivée, et `N` la taille du jeu d'abscisse.

```
def derivee(f, a, b, N) :
    h = (b - a) / N                                  # le pas des abscisses
    x = [ a + h * i for i in range(N) ]              # la liste des abscisses
    deri = []                                        # on initialise la liste des valeurs
                                                       de la derivee

    for i in range(N-1) :
        deri.append( ( f(x[i+1]) - f(x[i]) ) / h )   # on arrete a N-1 pour ne pas appeler
                                                       f(x[N]) qui n'existe pas

    deri.append( ( f(x[N-1]) - f(x[N-2]) ) / h )     # on choisit comme derniere valeur
                                                       celle identique a
                                                       la precedente
    return deri
```

La dérivée est renvoyé sous forme d'un tableau de `N` valeurs, donnant une approximation de la dérivée à chaque point $x_i$ (en donnant aux bords droit la même valeur qu'à son point voisin... Pas terrible mais on voit comment faire mieux dans l'exercice 1 !)

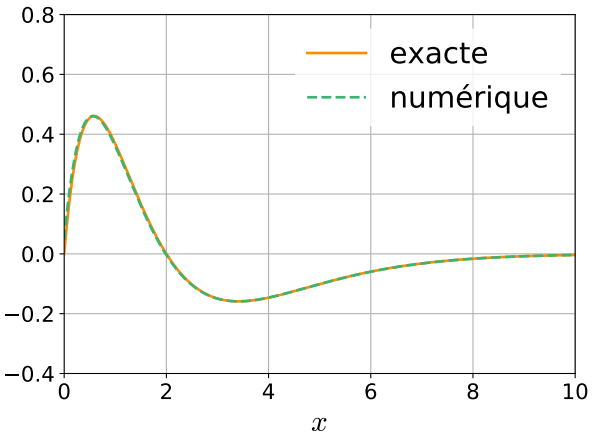
**Exemple.** Considérons la fonction

$$f(x) = x^2\,\mathrm{e}^{-x} \qquad \text{dont la dérivée est} \qquad f'(x) = (2-x)\,x\,\mathrm{e}^{-x}$$

Superposons alors cette dérivée exacte à celle obtenue numériquement par la méthode de différence finie. On obtient leurs valeurs respectives par

```
import numpy as np
f = lambda x : np.exp(-x) * x**2
fderExacte = lambda x : np.exp(-x) * x *(2 - x)
fderAppro = derivee(f, 0, 10, 200)
```

Les deux courbes sont quasiment identiques à l'œil : la méthode de différence finie fonctionne bien.

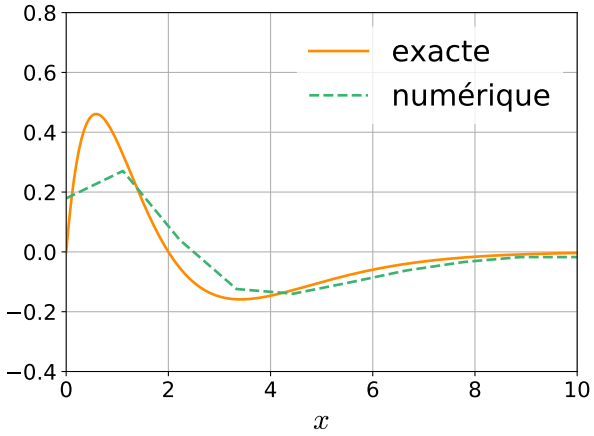


# 4 Précision de la méthode par différence finie

## 4.1 Aspect qualitatif

Évidemment, on se doute que plus le pas $h$ est petit, donc plus le nombre de points $N$ est élevé, plus la méthode est précise (car en fait, on se rapproche de la limite théorique $h \to 0$). Dans le cas contraire, que se passe-t-il si on ne considère que $N = 10$ points par exemple ?

```
import numpy as np
f = lambda x : np.exp(-x) * x**2
fderExacte = lambda x : np.exp(-x) * x *(2 - x)
fderAppro = derivee(f, 0, 10, 10)
```




Sans surprise, cette fois la méthode de différence finie est très mauvaise pour approcher la dérivée exacte, en particulier dans les zones où la fonction varie rapidement : retenez qu'il faut un pas suffisament petit pour que la méthode de différence finie fonctionne suffisament bien.

Le fait d'obtenir la dérivée par différence finie est néanmoins toujours sujet à une erreur (puisqu'on ne prend jamais la limite $h \to 0$ nécessaire pour obtenir la dérivée exacte). Peut-on évaluer l'ordre de grandeur de cette erreur ?

## 4.2 Ordre de grandeur de la précision

**Rappel. Développement de Taylor.** Le développement de Taylor à l'ordre $n$ en $h$ autour de $x$ est

$$f(x+h) = \sum_{k=0}^{n} \frac{h^n}{n!} f^{(n)}(x) + \mathcal{O}(h^{n+1})$$

En fait, la méthode de différence finie repose sur le développement de Taylor de la fonction $f$ considérée en $x$ à l'ordre 1 en $h$ :

$$f(x+h) = f(x) + h\,f'(x) + \mathcal{O}(h^2)$$

d'où on déduit

$$\boxed{f'(x) = \frac{f(x+h) - f(x)}{h} + \mathcal{O}(h)}$$

On se rend compte qu'approximer la dérivée par le premier terme revient à tolérer une erreur en $\mathcal{O}(h)$.

**Définition.** On appelle **ordre** de l'erreur la puissance de $h$ dans le terme d'erreur en $\mathcal{O}$.

L'approximation précédente est donc d'ordre 1. Peut-on faire mieux ?

## 4.3 Amélioration de la précision : formule centrée

On peut ! Considérons les deux développements limités suivants, en $h$ et en $-h$, poussés à l'ordre 2 :

$$\begin{aligned} f(x+h) &= f(x) + h\,f'(x) + \frac{h^2}{2} f''(x) + \mathcal{O}(h^3) \\ f(x-h) &= f(x) - h\,f'(x) + \frac{h^2}{2} f''(x) + \mathcal{O}(h^3) \end{aligned}$$

En soustrayant les deux, on se rend compte que

$$\boxed{f'(x) = \frac{f(x+h) - f(x-h)}{2\,h} + \mathcal{O}(h^2)}$$

Donc, en approximant la dérivée par

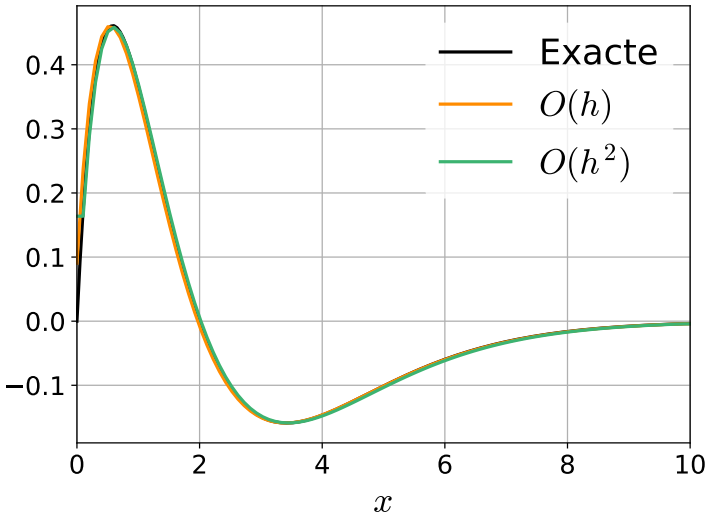
$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2\,h}$$

on fait cette fois une erreur en $\mathcal{O}(h^2)$, plutôt que le $\mathcal{O}(h)$ de la différence finie précédente. Si $h$ est plus petit que 1 (ce qui est le cas puisque $h$ a vocation à être petit), $h^2$ est plus petit que $h$ : l'erreur est donc plus faible (elle est d'ordre 2) avec cette deuxième méthode de différence finie.

**Proposition d'implémentation.** Voici une implémentation de cette autre méthode de différence finie.

```
def derivee2(f, a, b, N) :
    h = (b - a) / N
    x = [ a + h * i for i in range(N) ]
    deri = []

    deri.append( ( f(x[2]) - f(x[0]) ) / (2*h) )

    for i in range(1, N-1) :
        deri.append( ( f(x[i+1]) - f(x[i-1]) ) / (2*h) )

    deri.append( ( f(x[N-1]) - f(x[N-3]) ) / (2*h) )

    return deri
```

et traçons le résultat




La différence n'est pas flagrante ici car la première méthode en $\mathcal{O}(h)$ était déjà très bonne.

# 5 Dérivées d'ordre supérieur

## 5.1 Formule centrée pour la dérivée seconde

On peut aussi utiliser la méthode des différences finies pour évaluer des dérivées d'ordre supérieur. Par exemple, pour obtenir la dérivée seconde d'une fonction $f$, on peut de nouveau regarder les développements limités suivants

$$\begin{aligned} f(x+h) &= f(x) + h\,f'(x) + \frac{h^2}{2}\,f''(x) + \mathcal{O}(h^3) \\ f(x-h) &= f(x) - h\,f'(x) + \frac{h^2}{2}\,f''(x) + \mathcal{O}(h^3) \end{aligned}$$

Alors, en additionnant les deux, on obtient

$$f(x+h) + f(x-h) = 2\,f(x) + h^2\,f''(x) + \mathcal{O}(h^3)$$

soit

$$\boxed{f''(x) = \frac{f(x+h) + f(x-h) - 2\,f(x)}{h^2} + \mathcal{O}(h)}$$

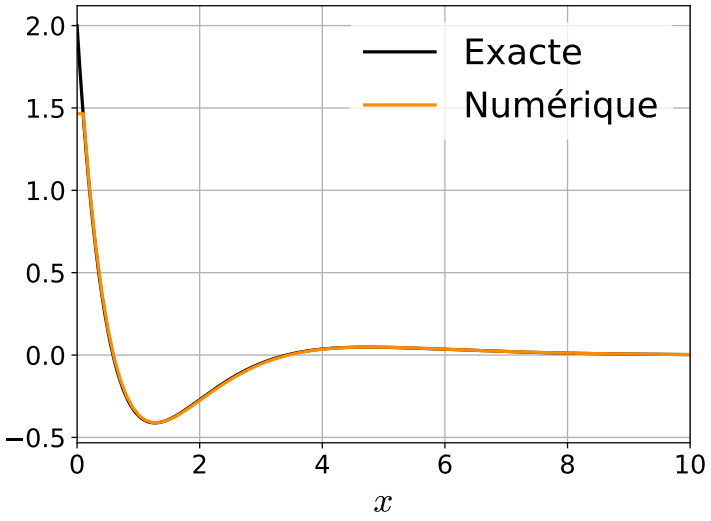
La dérivée seconde peut donc être approchée à l'ordre 1 en $h$ par

$$f''(x) \approx \frac{f(x+h) + f(x-h) - 2\,f(x)}{h^2}$$

ce que l'on peut implémenter comme suit (en donnant aux bords la même valeur qu'en leur point voisin... Pas terrible mais on voit comment faire mieux après !)

```
def seconde(f, a, b, N) :
    h = (b - a) / N
    x = [ a + h * i for i in range(N) ]
    seco = []

    seco.append( ( f(x[2]) + f(x[0]) - 2*f(x[1]) ) / h**2 )

    for i in range(N-1) :
        seco.append( ( f(x[i+1]) + f(x[i-1]) - 2*f(x[i]) ) / h**2 )

    seco.append( ( f(x[N-1]) + f(x[N-3]) - 2*f(x[N-2]) ) / h**2 )

    return seco
```

On trace la dérivée seconde exacte et celle obtenue par le programme précédent pour la fonction $f(x) = x^2\,\mathrm{e}^{-x}$. Le résultat ci-dessous est très convaincant.




## 5.2 Formule asymétrique pour la dérivée seconde

On peut aussi imaginer une approximation de la dérivée seconde en $x$ qui n'utilise pas $f(x+h)$ et $f(x-h)$ mais plutôt les valeurs de $f(x+h)$ et $f(x+2\,h)$. Cette expression n'est pas centrée mais utilise au contraire uniquement des valeurs de $f$ « à droite de $x$ » (c'est-à-dire plus grandes que $x$). On parle de formule **asymétrique**. Essayons de déterminer une telle expression. On a déjà les développements limités

$$\begin{aligned} f(x+h) &= f(x) + h\,f'(x) + \frac{h^2}{2}\,f''(x) + \mathcal{O}(h^3) \\ f(x+2\,h) &= f(x) + 2\,h\,f'(x) + 2\,h^2\,f''(x) + \mathcal{O}(h^3) \end{aligned}$$

donc

$$f(x+2\,h) - 2\,f(x+h) = -f(x) + h^2\,f''(x) + +\mathcal{O}(h^3)$$

d'où on construit une formule d'approximation de la dérivée seconde « asymétrique »

$$\boxed{f''(x) = \frac{f(x+2\,h) - 2\,f(x+h) + f(x)}{h^2} + \mathcal{O}(h)}$$

où l'approximation est d'ordre 1 en $h$ : elle n'est donc pas plus précise que la formule centrée. Par contre, cette formule asymétrique peut être utilisée pour évaluer la dérivée seconde sur le bord inférieur des abscisses, lorsque $f(x-h)$ ne peut pas être évaluée !

$\Delta\ \Delta\ \Delta$ Fin du cours.

**Méthode générale à retenir.** Pour chercher à estimer la derivée $k$-ième d'une fonction avec une erreur à l'ordre $n$, on se base sur des développements de Taylor de

$$f(x \pm h), \quad f(x \pm 2\,h), \quad \dots \quad , \quad f(x \pm k\,h), \quad f(x \pm (k+1)\,h), \quad \dots$$

et on essaie d'isoler la dérivée qui nous intéresse par combinaison linéaire de ces développements. On obtient alors, si on trouve les bonnes combinaisons à calculer, une expression du type

$$f^{(k)}(x) = \underbrace{\text{un truc avec } f(x \pm h),\ f(x \pm 2\,h) \text{ etc...}}_{\text{la formule pour l'estimation !}} + \underbrace{\mathcal{O}(h^n)}_{\text{et son erreur}}$$

L'erreur de l'approximation est exprimée en puissance de $h$, et cette puissance est appelée l'**ordre** de l'approximation. Plus l'ordre est grand plus l'approximation est précise.

**Exercice 1.** Proposez un algorithme qui calcule la dérivée première à l'aide de $f(x)$ et de $f(x-h)$.

**Exercice 2.** Proposez un algorithme qui calcule la dérivée première avec une erreur en $\mathcal{O}(h^3)$.

**Exercice 3.** Proposez un algorithme qui calcule la dérivée troisième et donner l'ordre de son erreur.

## Correction exercice 1.

On utilise le développement limité

$$f(x-h) = f(x) - h\,f'(x) + \mathcal{O}(h^2)$$

d'où on déduit que

$$f'(x) = \frac{f(x) - f(x-h)}{h} + \mathcal{O}(h)$$

Une approximation de la dérivée première à l'ordre un en $h$ est donc

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

Par rapport à celle vue en début de cours, elle est du même ordre (donc même précision). La différence est qu'on estime la dérivée en $x$ avec $f(x-h)$, donc elle est pratique si on n'a pas accès à $f(x+h)$ (au niveau d'un bord par exemple).

## Correction exercice 2.

On utilise les quatre développements limités suivants

$$\begin{aligned}
f(x+h) &= f(x) + h\,f'(x) + \frac{h^2}{2}\,f''(x) + \frac{h^3}{6}\,f'''(x) + \mathcal{O}(h^4) \\
f(x-h) &= f(x) - h\,f'(x) + \frac{h^2}{2}\,f''(x) - \frac{h^3}{6}\,f'''(x) + \mathcal{O}(h^4) \\
f(x+2\,h) &= f(x) + 2\,h\,f'(x) + 2\,h^2\,f''(x) + 4\,\frac{h^3}{3}\,f'''(x) + \mathcal{O}(h^4) \\
f(x-2\,h) &= f(x) - 2\,h\,f'(x) + 2\,h^2\,f''(x) - 4\,\frac{h^3}{3}\,f'''(x) + \mathcal{O}(h^4)
\end{aligned}$$

On calcule alors que

$$\begin{cases}
f(x+h) - f(x-h) &= 2\,h\,f'(x) + \dfrac{h^3}{3}\,f'''(x) + \mathcal{O}(h^4) \\
f(x+2\,h) - f(x-2\,h) &= 4\,h\,f'(x) + 8\,\dfrac{h^3}{3}\,f'''(x) + \mathcal{O}(h^4)
\end{cases}$$

et donc

$$\boxed{f'(x) = \frac{8\,\big(f(x+h) - f(x-h)\big) - \big(f(x+2\,h) - f(x-2\,h)\big)}{12\,h} + \mathcal{O}(h^3)}$$