Cours 11 : Calcul scientifique.
Calcul approché de dérivées et d'intégrales

MPSI - Lycée Thiers

2014/2015

Calcul approché d'une dérivée

• Soit f une application réelle dérivable sur un intervalle ouvert $]a;b[$ non-vide. La définition du nombre dérivée $f'(x_0)$ en $x_0 \in]a,b[$ fournit implicitement une méthode de calcul approché de $f'(x_0)$:

$$f'(x_0) = \lim_{h\to 0} \frac{f(x_0+h)-f(x_0)}{h} = \lim_{h\to 0} \frac{f(x_0+h)-f(x_0-h)}{2h}$$

• Premier algorithme :

```
def derive1(f,x,h):  # Calcul approchée d'un nombre dérivé
    return (f(x+h)-f(x)) / h
```

• Deuxième algorithme :

```
def derive2(f,x,h):  # Calcul approchée d'un nombre dérivé
    return (f(x+h)-f(x-h)) / (2*h)
```

• Remarque : en théorie il suffirait de prendre h suffisamment petit qu'on veut pour obtenir une approximation aussi fine que souhaité. Mais dans la pratique, la limitation de la représentation des nombres impose qu'il ne sert à rien de prendre une valeur de h trop petite : en effet les erreurs d'approximation dans le calcul du numérateur pourraient fausser le résultat : la division par h proche de 0 accroît grandement l'erreur.

• Utilisation :

```
>>> f=lambda x:x**2
>>> derive1(f,1,0.01)
2.0100000000000007
>>> derive2(f,1,0.01)
2.0000000000000018
```

Le résultat exact est ici 2 : on constate que l'approximation est bien meilleure dans le 2ème cas...

Calcul approché d'une dérivée

- Il ne s'agit pas d'un hasard ! Explication :
- *Développement de Taylor-Young à l'ordre 2 :*

Soit f de classe C^2 (c.à.d. deux fois dérivable et à dérivée seconde continue) sur un intervalle ouvert contenant x_0. Alors pour tout h suffisamment proche de 0 :

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

- Puisqu'au voisinage de 0 : $h^2 = O(h)$ (car $\lim_{h\to 0} \frac{h^2}{h}$ est finie) :

$$\boxed{\frac{f(x_0+h)-f(x_0)}{h}} = f'(x_0) + \frac{h}{2}.f''(x_0) + O(h^2) = \boxed{f'(x_0) + O(h)}$$

Tandis que :

$$\left.\begin{array}{l} f(x_0+h) = f(x_0) + h.f'(x_0) + \frac{h^2}{2}.f''(x_0) + O(h^3) \\ f(x_0-h) = f(x_0) - h.f'(x_0) + \frac{h^2}{2}.f''(x_0) + O(h^3) \end{array}\right. \Longrightarrow \boxed{\frac{f(x_0+h)-f(x_0-h)}{2h} = f'(x_0) + O(h^2)}$$

(attention $O(g) - O(g) = O(g) \neq 0$ en général !).

- Lorsque h tend vers 0, h^2 converge bien plus rapidement que h vers 0. C'est pourquoi, pour des valeurs identiques de h (proches de 0), la deuxième méthode donnera en général de bien meilleurs résultats que la première : l'erreur est un $O(h^2)$ plutôt qu'un $O(h)$.

Subdivision régulière d'un segment

• Calculer la valeur d'une intégrale est une opération en général difficile. Dans les applications scientifiques ou en ingénierie il est pourtant nécessaire de calculer rapidement des valeurs approchées d'intégrale (calcul d'aire, de volume, physique).

• En calculer une valeur approchée n'est pas très difficile. Les méthodes les plus simples consistent à subdiviser régulièrement l'intervalle d'intégration pour approcher le calcul intégral par un calcul d'aires de figures géométriques plus simples (par exemple dans la méthode des trapèzes).

• C'est pourquoi il est utile de définir la notion de subdivision régulière d'un segment :

Définition : *Soit* $[a,b]$ *un intervalle non vide et non réduit à un point (i.e.* $a<b$*). Soit* $n\in\mathbb{N}^*$ *; une subdivision régulière de* $[a,b]$ *en* $n+1$ *points est une suite finie de* $n+1$ *réels :* $(a_k)_{0\leqslant k\leqslant n}$ *vérifiant :* $a_0=a$*,* $a_n=b$ *et la suite* (u_n) *est arithmétique, autrement dit* $u_{k+1}-u_k$ *est constant.*

Autrement dit : pour tout $k\in[[0,n]]$, $\boxed{a_k=a+k.\dfrac{b-a}{n}}$.

• Sous `python` on crée facilement une subdivision régulière en $n+1$ points de l'intervalle $[a,b]$ à l'aide de la fonction `linspace(a,b,n+1)` du module `numpy` :

```
>>> import numpy as np
>>> X = np.linspace(a,b,n+1)
```

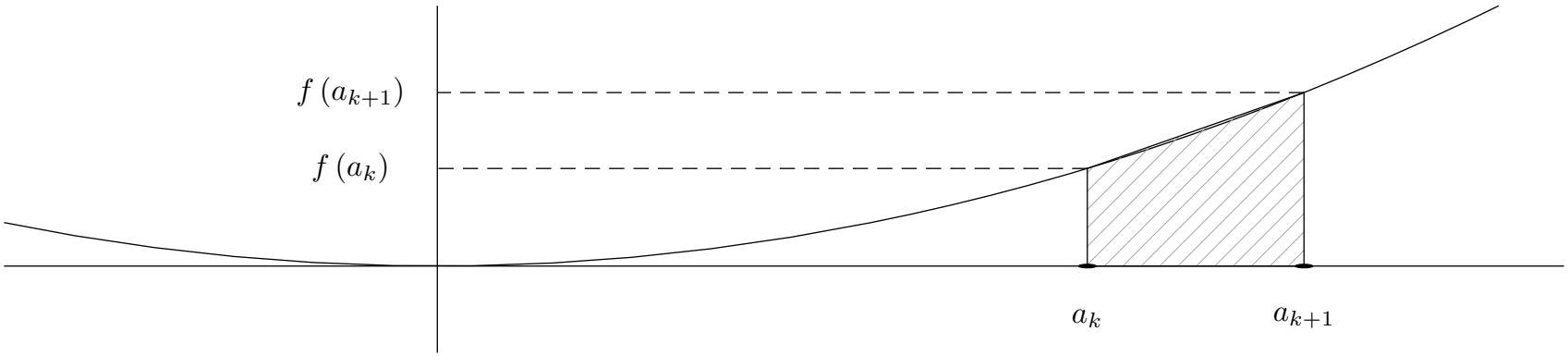
Méthode des Trapèzes

Soit une application f réelle : continue sur un intervalle $[a, b]$. Donné un entier n non nul (et suffisamment grand), la méthode des trapèzes calcule une valeur approchée de l'intégrale de f sur $[a, b]$ en approchant le domaine délimité par la courbe représentative de f (l'axe des abscisses et les droites $x = a$ et $x = b$) par des trapèzes ayant même hauteur.

Méthode des trapèzes : si $f : [a, b] \longrightarrow \mathbb{R}$ est continue :

Soit $a_k = a + k\dfrac{b-a}{n}$ pour $k \in [[0, n]]$ (ainsi $a_0 = a$, $a_n = b$). Si n est assez grand :

$$\int_a^b f(x)dx \approx \left(\frac{b-a}{n}\right) \sum_{k=0}^{n-1} \frac{f(a_k) + f(a_{k+1})}{2}$$



- $\left(\dfrac{b-a}{n}\right).\dfrac{f(a_k) + f(a_{k+1})}{2}$ est l'aire du trapèze hachuré (*hauteur* × *moyenne des bases*).

- Un calcul simple de somme fournit une formule plus concise :

$$\int_a^b f(x)dx \approx \left(\frac{b-a}{n}\right)\sum_{k=0}^{n-1}\frac{f(a_k)+f(a_{k+1})}{2}$$
$$= \left(\frac{b-a}{2n}\right)\left(\sum_{k=0}^{n-1} f(a_k) + \sum_{k=0}^{n-1} f(a_{k+1})\right)$$

En effectuant le changement d'indice $k \leftarrow k+1$ dans la deuxième somme :

$$= \left(\frac{b-a}{2n}\right)\left(\sum_{k=0}^{n-1} f(a_k) + \sum_{k=1}^{n} f(a_k)\right)$$
$$= \left(\frac{b-a}{2n}\right)\left(f(a_0) + \sum_{k=1}^{n-1} f(a_k) + \sum_{k=1}^{n-1} f(a_k) + f(a_n)\right)$$
$$= \left(\frac{b-a}{2n}\right)\left(f(a)+f(b)+2\sum_{k=1}^{n-1} f(a_k)\right)$$
$$= \left(\frac{b-a}{n}\right)\left(\frac{f(a)+f(b)}{2} + \sum_{k=1}^{n-1} f(a_k)\right)$$

$$\boxed{\int_a^b f(x)dx \approx \left(\frac{b-a}{n}\right)\left(\frac{f(a)+f(b)}{2} + \sum_{k=1}^{n-1} f(a_k)\right)} = \left(\frac{b-a}{n}\right)\left(\sum_{k=0}^{n} f(a_k) - \frac{f(a)+f(b)}{2}\right)$$

Méthode des Trapèzes

```
import numpy as np                                    # pour la fonction linspace()
def trapeze(f,a,b):
     if a > b:
        return -trapeze(f,b,a)
     N = 100                                          # Nombre de trapèzes (à choisir)
     X = np.linspace(a,b,N+1)                         # Subdivision régulière
     Y = f(X)                                         # Séquence des f(ak)
     return (b-a)/N * (sum(Y) - f(a)/2 - f(b)/2)
```

Le calcul de la somme peut s'effectuer à l'aide de la fonction `sum()` prenant en paramètre une séquence, ou par une boucle `for` sur la séquence Y.

- Lorsqu'appliqué à la fonction $f : x \longrightarrow x^2$ sur l'intervalle $[0, 1]$:

```
>>> trapeze(0,1)    # intégrale de 0 à 1
0.33335
>>> 1**3 /3         # valeur exacte
0.3333333333333333
>>> trapeze(0,100)  # intégrale de 0 à 100
333350.0
>>> 100**3 /3       # valeur exacte
333333.3333333333
```

- On obtiendrait une meilleure précision en fixant le nombre de point par unité de mesure :

```
N = 100                          # Nombre de trapèzes par unité de mesure (à choisir)
N = int(N * (b-a))               # Nombre total de trapèzes
```

Méthode des Trapèzes

- Version améliorée : passer le nombre de points par unité de mesure en argument :

```
import numpy as np                                  # pour la fonction linspace()
def trapeze(f,a,b,n):                               # avec 4 arguments
     if a > b:
        return -trapeze(f,b,a,n)
     N = int(n * (b-a))                             # Nombre de trapèzes
     X = np.linspace(a,b,N+1)                       # Subdivision régulière
     Y = f(X)                                       # Séquence des f(ak)
     return (b-a)/N * (sum(Y) - f(a)/2 - f(b)/2)
```

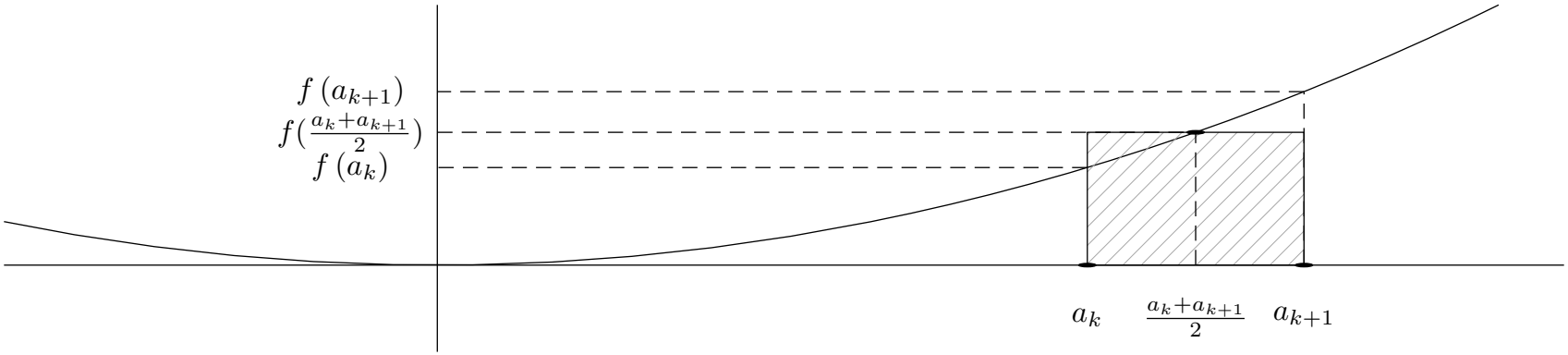
Méthode des rectangles

Soit une application f réelle : continue sur un intervalle $[a, b]$. Donné un entier n non nul (et suffisamment grand), la méthode des rectangles calcule une valeur approchée de l'intégrale de f sur $[a, b]$ en approchant le domaine délimité par la courbe représentative de f (l'axe des abscisses et les droites $x = a$ et $x = b$) par des rectangles.

Méthode des rectangles : si $f : [a, b] \longrightarrow \mathbb{R}$ est continue :

Soit $a_k = a + k\dfrac{b-a}{n}$ pour $k \in [[0, n]]$ (ainsi $a_0 = a$, $a_n = b$). Si n est assez grand :

$$\int_a^b f(x)dx \approx \left(\frac{b-a}{n}\right)\sum_{k=0}^{n-1} f\left(\frac{a_k + a_{k+1}}{2}\right)$$



- $\left(\dfrac{b-a}{n}\right)\sum_{k=0}^{n-1} f\left(\dfrac{a_k + a_{k+1}}{2}\right)$ est l'aire du rectangle hachuré (*hauteur* × *base*).

- Un calcul simple de somme fournit une formule plus concise :

$$\int_a^b f(x)dx \approx \left(\frac{b-a}{n}\right)\sum_{k=0}^{n-1} f\left(\frac{a_k + a_{k+1}}{2}\right)$$
$$= \left(\frac{b-a}{n}\right)\sum_{k=0}^{n-1} f\left(\frac{a + k(b-a)/n + a + (k+1)(b-a)/n}{2}\right)$$
$$= \boxed{\left(\frac{b-a}{n}\right)\sum_{k=0}^{n-1} f\left(a + k\frac{b-a}{n} + \frac{b-a}{2n}\right)}$$

Code :

```
import numpy as np                  # pour la fonction linspace()
def rectangle(f,a,b,n):             # Methode des rectangles
     if a > b:
        return -rectangle(f,b,a)
     N = int(n * (b-a))             # Nombre de rectangles
     X = np.linspace(a,b,N+1)       # Subdivision régulière
     X = X[:-1] + (b-a)/(2*N)       # Supprimer dernier élément et ajouter (b-a)/2N
     Y = f(X)                       # Image par f
     return (b-a)/N * sum(Y)
```

Méthode des rectangles

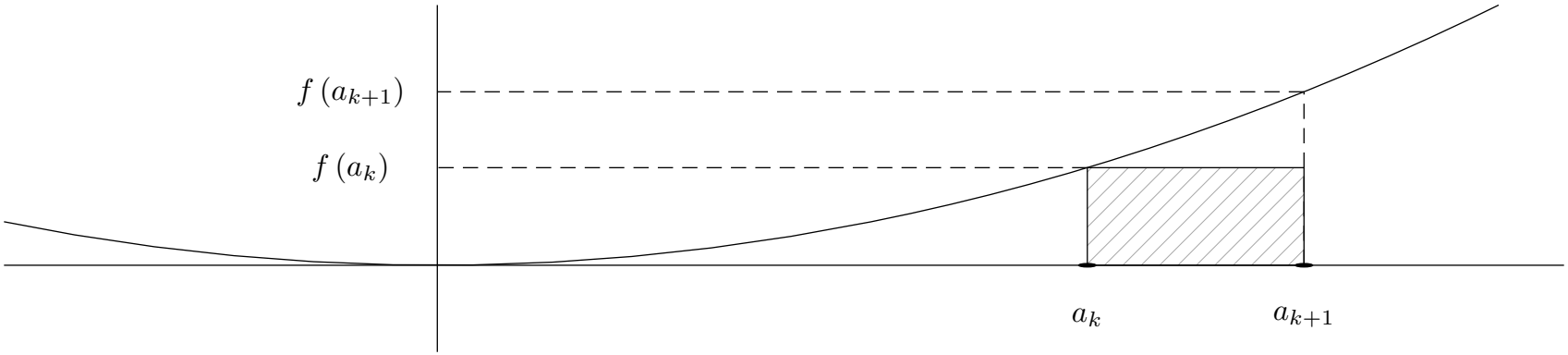
Il est plus simple de procéder ainsi :

Sous les mêmes hypothèses :

Méthode des rectangles : si $f : [a, b] \longrightarrow \mathbb{R}$ est continue :

Soit $a_k = a + k\dfrac{b-a}{n}$ pour $k \in [[0, n]]$ (ainsi $a_0 = a$, $a_n = b$). Si n est assez grand :

$$\int_a^b f(x)dx \approx \left(\frac{b-a}{n}\right)\sum_{k=0}^{n-1} f(a_k)$$



- $\left(\dfrac{b-a}{n}\right)\sum_{k=0}^{n-1} f(a_k)$ est l'aire du rectangle hachuré (*hauteur* × *base*).

Méthode des rectangles

Code :

```
import numpy as np                  # pour la fonction linspace()
def rectangle(f,a,b,n):             # Methode des rectangles
     if a > b:
        return -rectangle(f,b,a,n)
     N = int(n * (b-a))             # Nombre de rectangles
     X = np.linspace(a,b,N+1)       # Subdivision régulière
     Y = f(X[:-1])                  # Image par f du premier à l'avant-dernier
     return (b-a)/N * sum(Y)
```

- Ou encore, sans slicing ni `sum()` et à l'aide d'une boucle `for` :

```
import numpy as np
def rectangle(f,a,b,n):
    if a > b:
        return -rectangle(f,b,a,n)
    N = int(n * (b-a))
    X = np.linspace(a,b,N+1)
    somme = 0
    for k in range(N):
        somme += f(X[k])
    return somme * (b-a) / N
```