Chapitre 12

Calcul approché de dérivées, intégrales, primitives

12.1 Calcul approché de la valeur d'une dérivée

12.1.1 Comparaison de deux méthodes

• 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...

12.1.2 Explication

• Il ne s'agit pas d'un hasard ! Explication :

• *Développement de Taylor-Young à l'ordre 2 :* Soit f de classe C^2 :

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

$$\Longrightarrow \boxed{\frac{f(x_0+h)-f(x_0)}{h}} = f'(x_0) + \frac{h}{2}.f''(x_0) + o(h) = \boxed{f'(x_0)+o(1)}$$

Tandis que :

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

$$\Longrightarrow \boxed{\frac{f(x_0+h)-f(x_0-h)}{2h} = f'(x_0)+o(h)}$$

12.2 Calcul approché d'intégrales/primitives

12.2.1 Subdivision régulière d'un segment

• Calculer la valeur d'une intégrale est une opération en général difficile.

• 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.

• Subdivision régulière du segment $[a,b]$:

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 de terme général $a_{k+1}-a_k$ est stationnaire.

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

• Sous `python` :

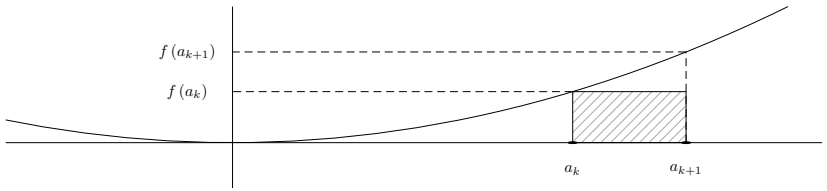
```
>>> import numpy as np
>>> X = np.linspace(a,b,n+1)      # ou SANS numpy :
>>> X = [a + k*(b-a)/n for k in range(n+1)]
```

12.2.2 Méthode des rectangles

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 :

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



f de classe $C^1\implies$ ERREUR $\leqslant\dfrac{(b-a)^2}{2n}\sup_{[a,b]}|f'|=O\left(\dfrac{1}{n}\right)$

12.2.3 Code pour la méthode des rectangles

Code :

```
import numpy as np                  # pour la fonction linspace()
def rectangle(f,a,b,n):             # Methode des rectangles
    X = np.linspace(a,b,n+1)        # Subdivision régulière
    Y = f(X)                  # Image par f du premier au dernier
    return (b-a)/n * (sum(Y) - Y[-1])
```

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

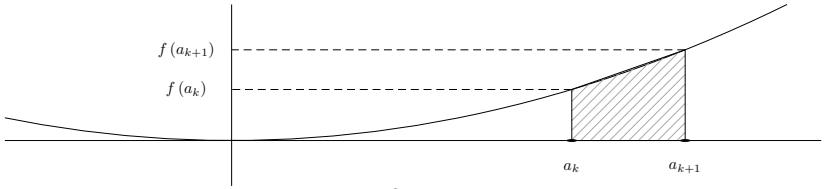
```
import numpy as np
def rectangle(f,a,b,n):
    X = np.linspace(a,b,n+1)
    somme = 0
    for k in range(n):
        somme += f(X[k])
    return somme * (b-a) / n
```

12.2.4 Méthode des Trapèzes

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.

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 :

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



f de classe $C^2\implies$ erreur $\leqslant\dfrac{(b-a)^3}{12n^2}\sup_{[a,b]}|f''|=O\left(\dfrac{1}{n^2}\right)$

12.2.5

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

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

($k \leftarrow k+1$ dans la 2ème $\sum$)

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

12.2.6 Simplification de l'expression

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 :

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

12.2.7 Code pour la méthode des Trapèzes

```
def trapeze(f,a,b,n):    # sans numpy
    x = a
    pas = (b-a)/n
    Somme = (f(a) + f(b))/2
    for _ in range(1,n):
        x = x + pas
        somme += f(x)
    return pas * somme
```

```
def trapeze(f,a,b,n):    # avec numpy.linspace()
    X = np.linspace(a,b,n+1)            # Subdivision régulière
    somme = 0
    for k in range(n):
        somme += (f(X[k]) + f(X[k+1])) / 2
    return (b-a)/n * somme
```

```
def trapeze(f,a,b,n):    # avec numpy.linspace() et sum()
    X = np.linspace(a,b,n+1)
    return (b-a)/n * (sum(f(X)) - f(a)/2 - f(b)/2)
```

12.2.8 Tracé de primitives

12.2.9 Tracé de primitive par la méthode des rectangles

• Pour tracer une primitive : on applique une méthode des rectangles/trapèzes, mais on mémorise dans un tableau toutes les valeurs intermédiaires :

```
import numpy as np
import matplotlib.pyplot as plt

def primitiveR(f,a,b,y0,n): # Avec la méthode des rectangles
    plt.figure(1)                # Nouvelle figure
    X = np.linspace(a,b,n+1)    # Subdivision régulière
    Y = np.empty(n+1)            # Tableau vide
    Y[0] = y0            # Remplissage tableau primitive
    pas = (b-a)/n
    for k in range(n):
        Y[k+1] = Y[k] + pas * f(X[k])
    plt.plot(X,f(X),X,Y)            # Tracé
    plt.legend(('$y=f(x)$', '$y=\int f(x)dx$'), 'upper left')
    plt.show()
```

Le tableau `Y` contient les valeurs prises par une primitive au dessus des points de la subdivision régulière.

12.2.10 Tracé de primitive par la méthode des trapèzes

• Pour tracer une primitive : on applique une méthode des rectangles/trapèzes, mais on mémorise dans un tableau toutes les valeurs intermédiaires :

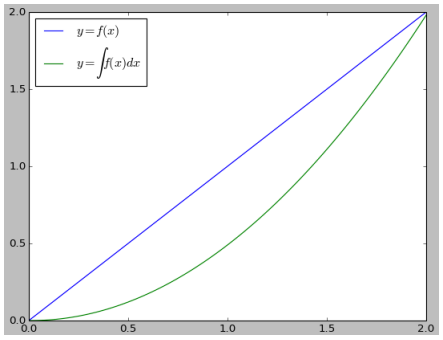
```
import numpy as np
import matplotlib.pyplot as plt

def primitiveT(f,a,b,y0,n): # Avec la méthode des trapezes
    plt.figure(1)
    X = np.linspace(a,b,n+1)
    Y = np.empty(n+1)
    Y[0] = y0
    pas = (b-a)/n
    for k in range(n):
        Y[k+1] = Y[k] + pas * (f(X[k]) + f(X[k+1]))/2
    plt.plot(X,f(X),X,Y)
    plt.legend(('$y=f(x)$', '$y=\int f(x)dx$'), 'upper left')
    plt.show()
```

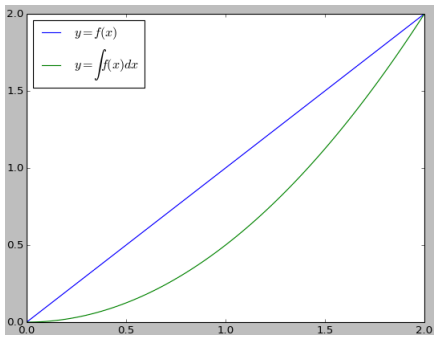
Le tableau `Y` contient les valeurs prises par une primitive au dessus des points de la subdivision régulière.

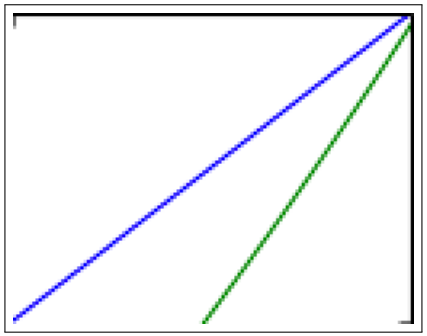
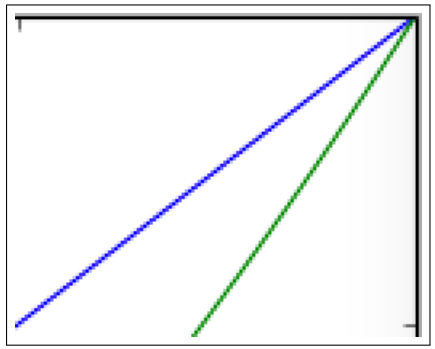
12.2.11 Comparaison pour $\int x\,dx$ sur $[0,2]$

Exemple :

```
In [1] : primitiveR(lambda x:x,0,2,0,100)
In [2] : primitiveT(lambda x:x,0,2,0,100)
```





On vérifie, comme attendu, que la méthode des trapèzes donne un meilleur résultat.

12.3 Intégration avec `scipy.integrate`

12.3.1 Donnés une fonction et un intervalle : avec `quad`

• `scipy` dispose de méthodes d'intégration de fonctions dans son sous-module `integrate` :

```
>>> from scipy import integrate
```

La méthode `quad(f,a,b)` pour intégrer f sur l'intervalle $[a,b]$:

```
>>> f = lambda x: x**2
>>> integrate.quad(f,0,1)    # Integration de f sur [0,1]
(0.33333333333333337, 3.700743415417189e-15)
```

Elle retourne un couple constitué de la valeur approchée de l'intégrale (1er élément) et d'une estimation de l'erreur commise. Pour accéder indépendamment à ces 2 valeurs :

```
>>> res = integrate.quad(f,0,1)
>>> res[0]    # Résultat obtenu
0.33333333333333337
>>> res[1]    # Estimation de l'erreur
3.700743415417189e-15
```

12.3.2 Donnés le pas et les valeurs de la fonction

• Les méthodes suivantes ne s'appliquent pas à une fonction et à un intervalle, mais seulement à un échantillonnage (régulier) de la fonction sur l'intervalle d'intégration :
`trapz()` applique la méthode des trapèzes à un échantillonnage :

```
>>> import numpy as np
>>> x = np.linspace(0,1,100)
>>> y = x ** 2
>>> integrate.trapz(y, dx=0.01)
0.33001683501683515
```

Attention à bien passer en argument optionnel le pas `dx` ($= h = \frac{b-a}{n}$) car par défaut `dx = 1` ce qui aboutirait à un résultat erroné.

• On peut facilement programmer une fonction approchante :

```
def trapz(Y, dx):
    return dx * (sum(Y) - (Y[0]+Y[-1])/2)
```