Intégration et dérivation

Ce chapitre est le premier d'une série s'intéressant à la question du calcul numérique. Nous verrons dans ces chapitres comment Python peut être utilisé comme un outil aidant à résoudre des problèmes de mathématiques, de physique, de sciences de l'ingénieur ou bien encore de chimie. Comme une grande partie de ces chapitres sont liés au calcul numérique sur des flottants, il sera bon de garder en tête les limitations issue de la représentation machine de ces flottants.

1 Intégration numérique

1.1 Utilité de l'intégration numérique

Considérons une fonction f, définie sur un sous-ensemble de R et à valeur dans R :

$$f \quad : \quad x \longmapsto f(x)$$

On suppose cette fonction f définie et intégrable sur un intervalle $[a,b]$ (où a et b sont deux réels). On souhaiterait calculer la quantité

$$\mathrm{I}_{[a,b]}(f) = \int_a^b f(\xi)\ \mathrm{d}\xi$$

La méthode usuellement employée en mathématiques consiste à déterminer une primitive F de la fonction f, puis à déterminer $\mathrm{F}(b) - \mathrm{F}(a)$. Il n'est cependant pas toujours possible de trouver par le calcul une telle primitive F, aussi s'en remet-on parfois à un calcul *numérique* de l'intégrale.

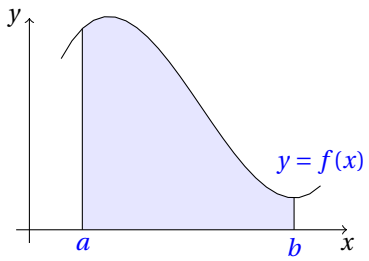
La plupart des méthodes d'intégration numérique sont des méthodes dites de *quadrature*. Elles s'efforcent d'approcher la quantité recherchée à partir des valeurs que prend la fonction en un certain nombre de points u_i pris dans l'intervalle $[a,b]$.

Dans ce chapitre, nous présenterons et étudierons quelques-unes de ces méthodes d'intégration par quadrature.

1.2 Différentes approches du problème de l'intégration

Les méthodes que nous allons présenter dans la suite s'efforcent d'approximer la fonction f dont on recherche une intégrale par une fonction $\overline{f}$ aussi proche de f que possible et dont on connaît une primitive, et par conséquent pour laquelle il est aisé de calculer l'intégrale.

On peut également envisager les choses sous un autre angle, équivalent. Rappelons que pour une fonction f positive sur $[a,b]$, lorsque $a < b$, la quantité $\int_a^b f(\xi)\ \mathrm{d}\xi$ correspond à l'aire géométrique comprise entre la courbe associée à la fonction f et l'axe des abscisses.



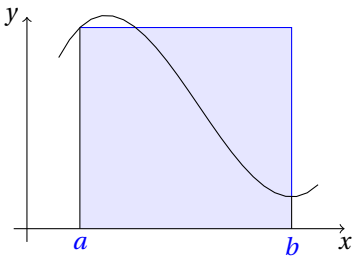
Déterminer l'intégrale consiste donc à estimer cette aire, aussi l'approximation de f par $\overline{f}$ peut être vue comme l'approximation de la forme géométrique sous la courbe de f par une forme géométrique dont l'aire est connue (rectangle, trapèze, etc.).

Il n'est en fait pas nécessaire que f soit positive, ni que $a < b$, sous réserve que l'on travaille avec des aires algébriques, et les formules que l'on établira s'étendront naturellement à ces situations.

1.3 Méthode du rectangle

La forme géométrique la plus simple que l'on puisse considérer pour notre problème est un rectangle, dont la base serait l'axe des abscisses, les côtés se situant au niveau des abscisses $x = a$ et $x = b$ et le sommet « au niveau de f ».

On peut envisager plusieurs solutions pour le choix de la hauteur du rectangle. La première, appelée « rectangle gauche », illustrée ci-dessous, consiste à choisir pour hauteur $f(a)$:



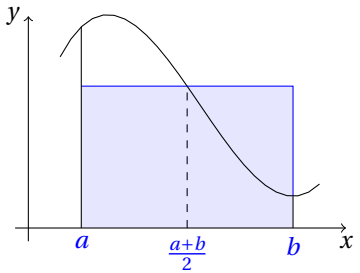
L'estimation $\hat{\mathrm{I}}_{[a,b]}(f)$ de l'intégrale $\mathrm{I}_{[a,b]}(f)$ correspond alors à l'aire du rectangle, soit

$$\hat{\mathrm{I}}_{[a,b]}(f) = (b-a) \times f(a)$$

De façon équivalente, cela revient à faire l'hypothèse que, pour $\xi \in [a,b]$, $f(\xi) \simeq f(a)$, ce qui conduit donc à écrire

$$\mathrm{I}_{[a,b]}(f) = \int_a^b f(\xi)\,\mathrm{d}\xi \simeq \int_a^b f(a)\,\mathrm{d}\xi = (b-a)\times f(a) = \hat{\mathrm{I}}_{[a,b]}(f)$$

Bien que la méthode du rectangle gauche puisse parfois avoir un intérêt pratique, on peut lui trouver une amélioration toute simple : plutôt que de prendre la valeur de f à l'abscisse $x = a$, on lui préférera la valeur de f au point situé au milieu de l'intervalle $[a,b]$. C'est la méthode dite du *point milieu*.



Dans le cas de la méthode du point milieu, l'estimation $\hat{\mathrm{I}}_{[a,b]}(f)$ de l'intégrale $\mathrm{I}_{[a,b]}(f)$ s'écrit

$$\hat{\mathrm{I}}_{[a,b]}(f) = (b-a)\times f\left(\frac{a+b}{2}\right)$$

On peut espérer que, dans un cas comme celui illustré précédemment, l'erreur commise dans un sens sur la première moitié de l'intervalle est partiellement compensée par une erreur commise dans l'autre sens sur la seconde moitié de l'intervalle. Nous reviendrons plus tard sur l'estimation de l'erreur commise lors de l'utilisation des différentes méthodes d'intégration numérique.

Implémenter cette méthode en Python ne présente aucune difficulté. On écrira par exemple :

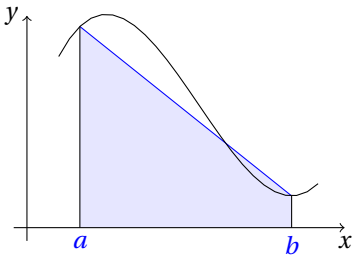
```
def IntegrationPointMilieu(f, a, b) :
    return (b-a) * f((a+b)/2)
```

Bien évidemment, cette méthode est encore très grossière. Par exemple, l'intégration par la méthode du point milieur de la fonction sin sur $[0,\pi]$ donnerait le résultat π, bien loin de la valeur théorique (égale à 2). Mais la méthode du point milieu (et les méthodes qui suivent) serviront de base pour construire des solutions beaucoup plus précises.

1.4 Méthode du trapèze

Pour s'approcher convenablement à la fois du point $(a, f(a))$ et du point $(b, f(b))$, on peut envisager de considérer un trapèze, dont les bases s'appuient sur les abscisses $x = a$ et $x = b$, dont un côté est l'axe des abscisses, et l'autre le segment entre les points $(a, f(a))$ et $(b, f(b))$.

Autrement dit, on approche l'intégrale par l'aire du trapèze représenté ci-dessous :



Dans cette situation, l'estimation $\hat{\mathrm{I}}_{[a,b]}(f)$ de l'intégrale $\mathrm{I}_{[a,b]}(f)$ s'écrira

$$\hat{\mathrm{I}}_{[a,b]}(f) = (b-a)\times\frac{f(a)+f(b)}{2}$$

Ce résultat peut se retrouver simplement en constatant que le calcul de l'aire du trapèze revient à intégrer sur l'intervalle $[a,b]$ une fonction affine $\overline{f}$ passant par les points $(a, f(a))$ et $(b, f(b))$, ce qui donc correspond à l'expression suivante :

$$\overline{f} \;:\; x \longmapsto \overline{f}(x) = f(a) + \frac{f(b)-f(a)}{b-a}\times(x-a)$$

puis en calculant l'intégrale de cette fonction $\overline{f}$ sur l'intervalle $[a,b]$, soit

$$\hat{\mathrm{I}}_{[a,b]}(f) = \int_a^b \overline{f}(\xi)\,\mathrm{d}\xi = f(a)\times(b-a) + \frac{f(b)-f(a)}{b-a}\times\frac{(b-a)^2}{2} = (b-a)\times\frac{f(a)+f(b)}{2}$$

L'implémentation Python est immédiate :

```
def IntegrationTrapeze(f, a, b) :
    return (b-a) * (f(a)+f(b)) / 2
```

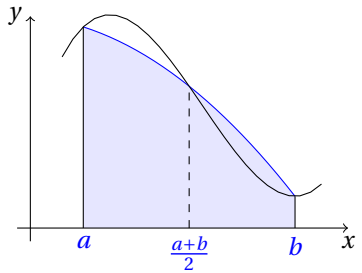
1.5 Méthode de Simpson

Plutôt que d'approximer la courbe par un segment, comme dans les approches précédentes, on peut vouloir utiliser un arc de parabole, c'est-à-dire par une fonction polynomiale d'ordre deux. Ainsi, la méthode de Simpson [1] propose d'approcher la fonction f sur l'intervalle $[a,b]$ par une fonction $\overline{f}$ polynomiale d'ordre deux, qui prend non seulement les mêmes valeurs que f aux abscisses a et b, comme la méthode du trapèze, mais également à l'abcisse m, correspondant au milieu de l'intervalle $[a,b]$:

$$\overline{f}(a) = f(a) \qquad \overline{f}(b) = f(b) \qquad \text{et} \qquad \overline{f}(m) = f(m) \qquad \text{avec} \qquad m = \frac{a+b}{2}$$

1. laquelle n'est pas explicitement mentionnée dans le programme, mais pourra se révéler utile.

Le graphe ci-dessous illustre un exemple de fonction $\overline{f}$ polynomiale d'ordre 2 approchant, au sens des critères précédents, une fonction f dont on cherche l'intégrale :



Il sera étudié en détail dans le cours de mathématiques le problème de l'interpolation polynomiale, c'est-à-dire la détermination du (ou des) polynôme(s) satisfaisant des contraintes telles que celles présentées ici. Il est possible de montrer (on l'admettra pour le moment) qu'il existe un unique polynôme du second degré qui réponde à nos présents besoins (sous réserve que $a \neq b$). On se contentera ici de construire une telle fonction polynomiale.

Les polynômes du second degré s'annulant en m et en b sont de la forme :

$$\mathscr{P} : x \longmapsto \mathrm{K}(x-m)(x-b)$$

où K est une constante réelle. Un seul de ces polynômes, s'annulant en m et en b, prend la valeur $f(a)$ pour $x = a$:

$$\mathscr{P}_a : x \longmapsto \frac{f(a)}{(a-m)(a-b)}(x-m)(x-b) = \frac{2f(a)}{(b-a)^2}(x-m)(x-b)$$

De même, il est possible de construire une fonction polynômiale s'annulant aux abscisses a et b et prenant la valeur $f(m)$ pour $x = m$:

$$\mathscr{P}_m : x \longmapsto \frac{f(m)}{(m-a)(m-b)}(x-a)(x-b) = \frac{-4f(m)}{(b-a)^2}(x-a)(x-b)$$

Ainsi qu'une fonction polynômiale s'annulant aux abscisses u et w et prenant la valeur $f(v)$ pour $x = v$:

$$\mathscr{P}_b : x \longmapsto \frac{f(b)}{(b-a)(b-m)}(x-a)(x-m) = \frac{2f(b)}{(b-a)^2}(x-a)(x-m)$$

La somme de ces trois fonctions polynômiales donne donc une fonction polynômiale du second degré qui correspond à nos attentes :

$$\mathscr{P} : x \longmapsto \frac{1}{(b-a)^2}\left[2f(a)(x-m)(x-b) - 4f(m)(x-a)(x-b) + 2f(b)(x-a)(x-m)\right]$$

L'intégrale de cette fonction polynomiale ne pose pas de difficulté particulière, et donne

$$\int_a^b \mathscr{P}(\xi)\ \mathrm{d}\xi = (b-a) \times \frac{f(a)+4f(m)+f(b)}{6}$$

Cette fois encore, l'implémentation en Python est immédiate :

```
def IntegrationSimpson(f, a, b) :
    return (b-a) * (f(a) + 4*f(m) + f(b)) / 6
```

1.6 Bilan

Comme on peut le voir, toutes les formules qui ont été proposées jusqu'à présent (et celles qui le seront dans la suite également d'ailleurs) apparaissent comme le produit de la largeur de l'intervalle, $b-a$, par une moyenne pondérée des valeurs que prend la fonction pour différentes valeurs de x. Pour l'instant, ces valeurs étaient $x = a$, $x = (a+b)/2$ et $x = b$.

Seuls les coefficients de pondération changent. Pour le rectangle gauche, on avait $\{1,0,0\}$; pour le point milieu, $\{0,1,0\}$; pour la méthode du trapèze, $\{1/2,0,1/2\}$ et enfin pour, la méthode de Simpson, $\{1/6,4/6,1/6\}$.

On peut considérer d'autres abscisses dans l'intervalle $[a,b]$, et d'autres coefficients, ce qui conduira à autant d'autres méthodes avec leurs forces et leurs faiblesses. Nous comparerons la précision obtenue avec les approches précédentes un peu plus tard. En attendant, il est assez facile d'imaginer des fonctions pour lesquels même une approximation par un arc de parabole ne donnera pas un bon résultat. Aussi nous faut-il améliorer quelque peu les choses.

2 Méthodes composites

2.1 Principe

On pourrait envisager d'approcher la fonction que l'on cherche à intégrer par des fonctions de plus en plus complexes (notamment des polynômes d'ordre de plus en plus élevés) que l'on sait intégrer. En pratique, toutefois, les choses deviennent rapidement complexes, et il existe une meilleure approche.

Pour déterminer une valeur approchée de cette aire, on peut découper l'intervalle $[a,b]$ en un ensemble de $n$intervalles $[u_i, u_{i+1}]$ vérifiant (en supposant $a < b$)

$$a = u_0 < u_1 < u_2 < \ldots < u_{n-1} < u_n = b$$

Les u_i étant connus, on peut décomposer notre intégrale :

$$\int_a^b f(\xi)\ \mathrm{d}\xi = \int_{u_0}^{u_1} f(\xi)\ \mathrm{d}\xi + \int_{u_1}^{u_2} f(\xi)\ \mathrm{d}\xi + \ldots + \int_{u_{n-1}}^{u_n} f(\xi)\ \mathrm{d}\xi$$

L'idée derrière cette décomposition est que sur un « petit » intervalle $[u_i, u_{i+1}]$, la fonction f sera peut-être suffisamment « régulière » pour que les méthodes précédentes soient de bonnes approximations. On utilisera donc les méthodes précédentes pour calculer les

intégrales sur chacun des n intervalles, et on en fera la somme. On parle de méthodes d'intégration *composites*.

Le découpage le plus simple de l'intervalle $[a,b]$ est un découpage régulier, en n intervalles $[u_i, u_{i+1}]$ de largeur $(b-a)/n$. On peut alors définir simplement les u_i :

$$u_i = a + \frac{b-a}{n} \times i$$

On retrouve bien $u_0 = a$ et $u_n = b$ avec la formule précédente.

Si l'on applique la méthode du rectangle gauche, on a donc

$$\hat{\mathrm{I}}_{[a,b],n}(f) = \sum_{i=0}^{n-1} \frac{b-a}{n} f\left(a + \frac{b-a}{n} \times i\right) = \frac{b-a}{n} \sum_{i=0}^{n-1} f\left(a + \frac{b-a}{n} \times i\right)$$

En Python, cela s'écrirait par exemple :

```
def IntegrationRectComposite(f, a, b, n) :
    pas = (b-a)/n

    # On crée une liste pour recueillir les termes de la somme
    liste = []

    # On évalue la fonction aux différentes abscisses u_i
    for i in range(n) :
        liste.append( f(a + i*pas) )

    # Et enfin on calcule la somme des différents termes
    return pas * math.fsum(liste)
```

On remarquera que l'on n'a pas utilisé un accumulateur pour calculer la somme, comme on l'a fait fréquemment précédemment. En effet, on travaille ici sur des flottants, avec les limitations que leur représentation machine impose. Si le nombre d'intervalles n est très grand, les derniers termes de la somme pourraient être partiellement absorbés, ce qui donnerait des imprécisions sur le résultat. Autant laisser à Python le soin d'effectuer la somme des différentes contributions le plus soigneusement possible !

L'inconvénient est qu'il nous faut mémoriser chacun des n termes de la somme, ce qui représentent pour de grandes valeurs de n un certain coût mémoire. Il existe des solutions pour éviter à la fois les soucis de précision et un trop grand coût en terme de mémoire, mais nous ne nous attarderons pas dessus pour le moment.

On peut procéder de la même façon avec la méthode du point milieu, pour laquelle on peut écrire

$$\hat{\mathrm{I}}_{[a,b],n}(f) = \sum_{i=0}^{n-1} \frac{b-a}{n} f\left(a + \frac{b-a}{n} \times \left(i + \frac{1}{2}\right)\right) = \frac{b-a}{n} \sum_{i=0}^{n-1} f\left(a + \frac{b-a}{n} \times \left(i + \frac{1}{2}\right)\right)$$

Et donc en Python

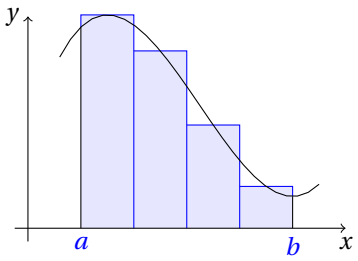
```
def IntegrationPointMilieuComposite(f, a, b, n) :
    pas = (b-a)/n

    liste = []

    for i in range(n) :
        liste.append( f(a + (i+0.5)*pas) )

    return pas * math.fsum(liste)
```

Graphiquement, cette méthode du point milieu composite consiste à approximer l'aire sous la courbe de f par la somme des aires d'un ensemble de rectangles, comme ci-dessous :



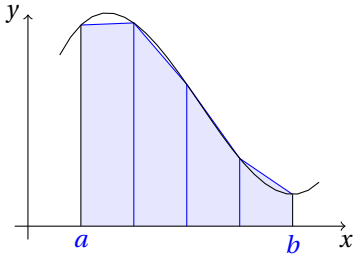
Il est assez logique de penser que si le nombre n d'intervalles, donc de rectangles, est assez grand, on s'approchera avec une assez bonne précision de l'aire recherchée !

Il en est de même pour la méthode des trapèzes :

$$\hat{\mathrm{I}}_{[a,b],n}(f) = \sum_{i=0}^{n-1} \frac{b-a}{2n} \times \left[f\left(a + \frac{b-a}{n} \times i\right) + f\left(a + \frac{b-a}{n} \times (i+1)\right)\right]$$

Mais notons que la plupart des termes apparaîssent deux fois dans la somme, ce qui permet de simplifier quelque peu notre formule :

$$\hat{\mathrm{I}}_{[a,b],n}(f) = \frac{b-a}{n} \times \left[\frac{f(a)}{2} + \frac{f(b)}{2} + \sum_{i=1}^{n-1} f\left(a + \frac{b-a}{n} \times i\right)\right]$$



Et donc en Python :

```
def IntegrationTrapComposite(f, a, b, n) :
    pas = (b-a)/n

    # Une liste pour recueillir les différentes contributions
    liste = [ f(a)/2, f(b)/2 ]

    # Dans laquelle on ajoute les termes de la somme
    x = a + pas

    for i in range(n-1) :
        liste.append( f(x) )
        x = x + pas

    return pas * math.fsum(liste)
```

Et enfin la méthode de Simpson composite donne :

$$\hat{I}_{[a,b],n}(f) = \sum_{i=0}^{n-1} \frac{b-a}{6n} \left[f\left(a + \frac{b-a}{n} \times i\right) + 4f\left(a + \frac{b-a}{n} \times \left(i + \frac{1}{2}\right)\right) + f\left(a + \frac{b-a}{n} \times (i+1)\right) \right]$$

Ici encore, certains des termes apparaissent plusieurs fois. Si l'on met le terme $\frac{b-a}{n}$ en facteur, il y aura un $\frac{1}{6}$ devant $f(a)$ et $f(b)$, un coefficient $\frac{1}{3}$ devant les $f(u_i)$ pour $0 < i < n$ et un coefficient $\frac{2}{3}$ au niveau des milieux des intervalles.

En Python, si l'on veut éviter de faire trop de calculs inutiles, on pourra écrire (il est intéressant d'essayer de comprendre pourquoi ce programme effectue bien le calcul de l'expression du dessus) :

```
def IntegrationSimpsonComposite(f, a, b, n) :
    pas = (b-a)/(2*n)

    liste = [ f(a), f(b) ]

    x = a + pas

    for i in range(2*n-1) :
        if i%2 == 1 :
            liste.append( 2 * f(x) )
        else :
            liste.append( 4 * f(x) )
        x = x + pas

    return pas * math.fsum(liste) / 3
```

2.2 Choix du nombre de subdivisions

Comme on peut s'y attendre, plus le nombre de subdivisions est élevé, plus on peut s'attendre à avoir une bonne estimation de l'intégrale recherchée. Supposons par exemple que l'on cherche à calculer

$$\int_0^\pi \sin(\xi)\, d\xi$$

Le résultat devrait être $\left[-\cos(\xi)\right]_0^\pi = 2$.

La méthode du point milieu pour $n = 1, 2, 8$ et 64 nous donne :

```
In []: IntegrationPointMilieuComposite(sin, 0, pi, n=1)
Out[]: 3.141592653589793

In []: IntegrationPointMilieuComposite(sin, 0, pi, n=2)
Out[]: 2.221441469079183

In []: IntegrationPointMilieuComposite(sin, 0, pi, n=8)
Out[]: 2.012909085599128

In []: IntegrationPointMilieuComposite(sin, 0, pi, n=64)
Out[]: 2.000200811728367
```

On voit que, progressivement, lorsque n augmente, on s'approche de la valeur correcte.

Il est possible de s'en approcher beaucoup en prenant de très grandes valeurs de n (sous réserve que l'on soit très prudent en effectuant la sommes des contributions) :

```
In []: IntegrationPointMilieuComposite(sin, 0, pi, n=2**20)
Out[]: 2.0000000000007483
```

Pour les autres méthodes, comme par exemple la méthode de Simpson, l'augmentation de n permet également de s'approcher davantage du résultat recherché :

```
In []: IntegrationSimpsonComposite(sin, 0, pi, n=1)
Out[]: 2.0943951023931953

In []: IntegrationSimpsonComposite(sin, 0, pi, n=2)
Out[]: 2.0045597549844207

In []: IntegrationSimpsonComposite(sin, 0, pi, n=8)
Out[]: 2.000016591047935

In []: IntegrationSimpsonComposite(sin, 0, pi, n=64)
Out[]: 2.000000004032262
```

On remarque cependant que la méthode de Simpson semble, sur cet exemple, converger plus rapidement vers le résultat attendu.

Sous réserve que la somme des contributions soit effectuée correctement, augmenter le n n'aura pas de conséquences néfaste sur le résultat obtenu. En revanche, cela augmentera, nécessairement, le temps de calcul. On souhaiterait donc connaître le n qui donnerait un résultat suffisamment précis sans calculs inutiles.

Pour cela, il est fréquent de procéder par raffinement successifs : on effectue une première estimation de l'intégrale avec un n relativement faible, puis on augmente progressivement ce n (en choisissant les valeurs successives de n permettant de ne pas faire trop de calculs). Lorsque les valeurs obtenues pour l'intégrale n'évoluent pratiquement plus (par exemple, lorsque deux valeurs consécutives diffèrent de moins qu'un seuil ϵ), on cesse d'augmenter la valeur de n.

Il existe cependant des cas où la suite constituée des estimations successives de l'intégrale est, pendant un temps, quasi-stationnaire autour d'une valeur qui n'est pas l'intégrale définitive (cf séance de travaux pratiques), et cela peut donner des estimations fortement erronées de l'intégrale, la subdivision n'étant pas poussée assez loin.

En fait, les algorithmes d'intégration n'effectuent pas des subdivisions successives sur la totalité de l'intervalle. Si les estimations de l'intégrale sur une partie de l'intervalle $[a,b]$ n'évoluent plus, on ne poursuivra pas les subdivisions dans cette partie de l'intervalle, et on ne prendra des points supplémentaires qu'aux endroits où c'est nécessaire.

3 Précision des méthodes d'intégration

3.1 Introduction

Il est pertinent de se poser la question de l'erreur que l'on va commettre en intégrant numériquement une fonction. Le problème est double :

- les méthodes d'intégration numérique présentées ici, sans même qu'il soit question d'implémentation, ne fournissent qu'une *estimation* de l'intégrale, et non sa valeur exacte ;
- lorsque l'on va implémenter ces méthodes d'intégration numérique, on travaillera avec des données qui ont une précision limitée (nombres flottants par exemple), ce qui occasionnera une *seconde* source d'erreur.

Pour un nombre de subdivisions limité, c'est généralement la première source d'erreur qui est la plus importante, et c'est cette erreur (liée à la méthode d'intégration) que nous allons étudier dans la suite de cette section.

Toutefois, si cette erreur décroît avec le nombre n de subdivisions, elle ne permettra pas d'obtenir une précision aussi grande qu'on le souhaite : en effet, au bout d'un certain moment, ce sont les erreurs liées à la précision des flottants qui limiteront la précision que l'on peut obtenir. C'est la raison pour laquelle on a utilisé la fonction `math.fsum`, pour limiter les problèmes d'absorption et de « cancellation ».

Mais quelles que soient les précautions que l'on prend, le travail avec des nombres flottants sur 64 bits par exemple ne saura jamais fournir un résultat avec plus d'une quinzaine de chiffres significatifs.

On gardera donc en tête dans la suite que l'on parle d'erreurs liées à la méthode d'intégration, en négligeant les problèmes d'imprécision liées aux calculs avec des nombres flottants.

3.2 Ordre d'une méthode de quadrature

Une méthode de quadrature est dite *d'ordre k* lorsque le résultat qu'elle fournit est exact pour un polynôme de degré inférieur ou égal à k.

La méthode des trapèzes est, de façon évidente, d'ordre 1 puisqu'elle donne un résultat exact pour des fonctions affines (dans cette situation, on a en effet $\overline{f} = f$). La méthode du point milieu se trouve être également d'ordre 1 ! En effet, si l'on a une fonction affine

$$f \quad : \quad x \longmapsto \lambda x + \mu$$

alors le calcul direct de l'intégrale donne

$$\int_a^b \lambda\xi + \mu \ \mathrm{d}\xi = \left[\lambda\frac{\xi^2}{2} + \mu\xi\right]_a^b = \lambda\frac{b^2-a^2}{2} + \mu\,(b-a) = (b-a)\times\left(\lambda\frac{a+b}{2} + \mu\right)$$

ce qui correspond précisément à l'expression obtenue par la méthode du point milieu[2].

De la même façon, on pourrait s'attendre à ce que la méthode de Simpson soit une méthode d'ordre 2, puisqu'elle approxime la fonction f par un polynôme de degré 2, mais elle est en fait d'ordre 3. Pour le vérifier, compte tenu de la linéarité de la méthode, il suffit de le vérifier pour un polynôme de degré 3 quelconque.

On prend par exemple $(x-a)^3$. Le calcul direct de l'intégrale donne :

$$\int_a^b (\xi-a)^3 \ \mathrm{d}\xi = \left[\frac{(\xi-a)^4}{4}\right]_a^b = \frac{(b-a)^4}{4}$$

Tandis que la méthode de Simpson conduit à :

$$\hat{\mathrm{I}}_{[a,b]}\left((\xi-a)^3\right) = \frac{b-a}{6}\times\left[0 + 4\left(\frac{b+a}{2}-a\right)^3 + (b-a)^3\right] = \frac{(b-a)^4}{4}$$

La méthode de Simpson donne donc bien le résultat exact pour des polynômes de degré inférieur ou égal à 3, ce qui en fait une méthode de quadrature d'ordre 3.

2. La méthode du rectangle gauche, en revanche, est simplement d'ordre 0.

3.3 Influence de la subdivision

Si l'on met à part le cas particulier de polynômes de degré suffisamment faible pour que les méthodes proposées donnent un résultat exact, il est probable que la méthode de quadrature que l'on utilise ne donne pas tout à fait la bonne valeur pour l'intégrale.

Si les intervalles choisis pour l'approche composite sont trop grands, on comprends aisément que le résultat risque d'être moins bon. Il est donc naturel de vouloir augmenter le nombre n d'intervalles afin d'avoir une estimation de l'intégrale aussi juste que possible. Nous allons voir que, lorsqu'il est possible de faire quelques hypothèses supplémentaires sur les fonctions, on peut quantifier la vitesse de convergence vers la solution en déterminant un majorant de l'erreur commise lors du calcul de l'intégrale de f sur l'intervalle $[a,b]$.

Cas de la méthode du rectangle gauche

On s'intéresse, dans un premier temps, à la méthode du rectangle gauche composite, et on suppose que la fonction f dont on cherche l'intégrale sur $[a,b]$ est de classe $\mathscr{C}^1$, et que M_1 et un majorant de $\left|f'\right|$ sur ce même intervalle $[a,b]$.

Cette hypothèse permet d'appliquer le théorème des accroissements finis entre deux points x_1 et x_2 de $[a,b]$, et d'écrire l'intégalité

$$\left|f(x_2)-f(x_1)\right| \leq \mathrm{M}_1\left|x_2-x_1\right|$$

Nous allons, dans un premier temps, trouver un majorant de l'erreur commise par la méthode du rectangle gauche sur un des intervalles $[u_i,u_{i+1}]$, puis en déduire un majorant de l'erreur commise par la méthode composite sur $[a,b]$.

La méthode du rectangle « gauche » appliquée à un intervalle $[u_i,u_{i+1}]$ conduit à une erreur ε commise lors de l'estimation de l'intégrale sur cet intervalle $[u_i,u_{i+1}]$

$$\varepsilon = \left|(u_{i+1}-u_i)\times f(u_i) - \int_{u_i}^{u_{i+1}} f(\xi)\,\mathrm{d}\xi\right| \quad \text{soit} \quad \varepsilon = \left|\int_{u_i}^{u_{i+1}} f(u_i)-f(\xi)\,\mathrm{d}\xi\right|$$

En utilisant l'inégalité des accroissements finis, on peut en déduire que

$$\varepsilon \leqslant \int_{u_i}^{u_{i+1}} \left|f(u_i)-f(\xi)\right|\mathrm{d}\xi \leqslant \int_{u_i}^{u_{i+1}} \mathrm{M}_1\left|u_i-\xi\right|\mathrm{d}\xi = \mathrm{M}_1\frac{(u_{i+1}-u_i)^2}{2}$$

Pour la méthode du rectangle gauche composite, on peut donc affirmer que l'erreur commise lors de l'estimation de l'intégrale sur $[a,b]$ vérifie

$$\left|\hat{\mathrm{I}}_{[a,b],n}(f)-\mathrm{I}_{[a,b]}(f)\right| \leqslant n\times\mathrm{M}_1\times\frac{(b-a)^2}{2n^2}$$

Et donc

$$\left|\hat{\mathrm{I}}_{[a,b],n}(f)-\mathrm{I}_{[a,b]}(f)\right| \leqslant \frac{\mathrm{M}_1(b-a)^2}{2n}$$

Cas de la méthode du point milieu

Pour la méthode du point milieu, on peut obtenir, avec un raisonnement similaire et les mêmes hypothèses sur la fonction f,

$$\left|\hat{\mathrm{I}}_{[a,b],n}(f)-\mathrm{I}_{[a,b]}(f)\right| \leqslant \frac{\mathrm{M}_1(b-a)^2}{4n}$$

Mais si la fonction f est de classe $\mathscr{C}^2$ sur l'intervalle $[a,b]$, et que M_2 est un majorant de $\left|f''\right|$ sur $[a,b]$, on peut proposer une autre majoration de l'erreur commise.

On commence, de la même façon, par majorer l'erreur commise sur un intervalle $[u_i,u_{i+1}]$, qui s'écrit

$$\varepsilon = \left|(u_{i+1}-u_i)\times f\left(\frac{u_i+u_{i+1}}{2}\right) - \int_{u_i}^{u_{i+1}} f(\xi)\,\mathrm{d}\xi\right| = \left|\int_{u_i}^{u_{i+1}} f\left(\frac{u_i+u_{i+1}}{2}\right)-f(\xi)\,\mathrm{d}\xi\right|$$

On peut remarquer que :

$$\int_{u_i}^{u_{i+1}}\left(\xi-\frac{u_i+u_{i+1}}{2}\right)\times f'\left(\frac{u_i+u_{i+1}}{2}\right)\mathrm{d}\xi = 0$$

Et donc,

$$\varepsilon = \left|\int_{u_i}^{u_{i+1}} f\left(\frac{u_i+u_{i+1}}{2}\right)+\left(\xi-\frac{u_i+u_{i+1}}{2}\right)\times f'\left(\frac{u_i+u_{i+1}}{2}\right)-f(\xi)\,\mathrm{d}\xi\right|$$

Pour un ξ donné, différent de $\frac{u_i+u_{i+1}}{2}$, considérons la fonction g_ξ définie sur l'intervalle considéré par

$$g_\xi : x \mapsto f\left(\frac{u_i+u_{i+1}}{2}\right)+\left(x-\frac{u_i+u_{i+1}}{2}\right)\times f'\left(\frac{u_i+u_{i+1}}{2}\right)-f(x)+\frac{\mathrm{K}}{2}\times\left(x-\frac{u_i+u_{i+1}}{2}\right)^2$$

dans laquelle on a choisi le coefficient K de sorte que $g_\xi(\xi)=0$.

On peut remarquer que $g_\xi(\xi)=0$ et $g_\xi\left(\frac{u_i+u_{i+1}}{2}\right)=0$. Par application du théorème de Rolle, il existe donc un réel c_1 compris strictement entre ξ et $\frac{u_i+u_{i+1}}{2}$ tel que $g'_\xi(c_1)=0$.

Par ailleurs,

$$g'_\xi(x) = f'\left(\frac{u_i+u_{i+1}}{2}\right)-f'(x)+\mathrm{K}\times\left(x-\frac{u_i+u_{i+1}}{2}\right)$$

Donc la fonction g' s'annule en c_1, mais également en $\frac{u_i+u_{i+1}}{2}$ (distinct de c_1). En appliquant une nouvelle fois le théorème de Rolle à g', on montre qu'il existe un c_2 compris entre c_1 et $\frac{u_i+u_{i+1}}{2}$ tel que $g''(c_2)=0$.

Or $g''(x)=\mathrm{K}-f''(x)$, donc $\mathrm{K}=f''(c_2)$, et donc

$$f\left(\frac{u_i+u_{i+1}}{2}\right)+\left(\xi-\frac{u_i+u_{i+1}}{2}\right)\times f'\left(\frac{u_i+u_{i+1}}{2}\right)-f(\xi) = -\frac{f''(c_2)}{2}\times\left(\xi-\frac{u_i+u_{i+1}}{2}\right)^2$$

Ce qui permet d'écrire

$$\varepsilon \leqslant \frac{M_2}{2}\int_{u_i}^{u_{i+1}}\left(\xi-\frac{u_i+u_{i+1}}{2}\right)^2 d\xi = M_2\frac{(u_{i+1}-u_i)^3}{24}$$

et donc, pour l'erreur commise sur l'ensemble de l'intervalle $[a,b]$,

$$\left|\hat{I}_{[a,b],n}(f)-I_{[a,b]}(f)\right| \leqslant \frac{M_2(b-a)^3}{24n^2}$$

Cas de la méthode du trapèze

Supposons f de classe $\mathscr{C}^2$, et M_2 un majorant de $|f''|$ sur l'intervalle $[a,b]$.

L'erreur commise sur l'intervalle $[u_i,u_{i+1}]$ s'écrit

$$\varepsilon = \left|\int_{u_i}^{u_{i+1}}\overline{f}(\xi)-f(\xi)\,d\xi\right|$$

où $\overline{f}$ est la fonction affine vérifiant $\overline{f}(u_i)=f(u_i)$ et $\overline{f}(u_{i+1})=f(u_{i+1})$.

Pour un ξ quelconque, on peut définir une fonction g_ξ définie sur $[u_i,u_{i+1}]$ par

$$g_\xi : x \mapsto \overline{f}(x)-f(x)+K\frac{(x-u_i)(x-u_{i+1})}{2}$$

où la constante K est choisie de sorte que $g_\xi(\xi)=0$.

On a $g_\xi(u_i)=g_\xi(\xi)=g_\xi(u_{i+1})=0$, donc d'après le théorème de Rolle, il existe $c_1\in]u_i,\xi[$ et $c_2\in]\xi,u_{i+1}[$ vérifiant $g_\xi'(c_1)=g_\xi'(c_2)=0$.

Toujours d'après le théorème de Rolle, il existe un $c_3\in]c_1,c_2[$ vérifiant $g''(c_3)=0$.

Mais comme $g_\xi''(x)=K-f''(x)$, on a $K=f''(c_3)$ et donc

$$\overline{f}(\xi)-f(\xi)=-f''(c_3)\frac{(\xi-u_i)(\xi-u_{i+1})}{2}$$

Ce qui conduit à l'inégalité

$$\varepsilon \leqslant \frac{M_3}{2}\int_{u_i}^{u_{i+1}}(\xi-u_i)\times(u_{i+1}-\xi)\,d\xi = M_2\frac{(u_{i+1}-u_i)^3}{12}$$

Et donc, finalement, à la majoration de l'erreur commise sur l'estimation de l'intégrale sur $[a,b]$ suivante :

$$\left|\hat{I}_{[a,b],n}(f)-I_{[a,b]}(f)\right| \leqslant \frac{M_2(b-a)^3}{12n^2}$$

Ce qui est une majoration un peu moins intéressante que celle obtenue pour la méthode du point milieu composite.

Cas de la méthode de Simpson

On suppose, pour ce dernier exemple, que la fonction f est de classe $\mathscr{C}^4$ sur l'intervalle $[a,b]$, et que M_4 est un majorant de $|f^4|$ sur ce même intervalle.

L'erreur commise lors de l'application de la méthode de Simpson sur un intervalle $[u_i,u_{i+1}]$ est

$$\varepsilon = \left|\int_{u_i}^{u_{i+1}}\mathscr{P}(\xi)-f(\xi)\,d\xi\right|$$

où $\mathscr{P}$ est la fonction polynomiale de degré 2 vérifiant

$$\mathscr{P}(u_i)=f(u_i),\quad \mathscr{P}(u_{i+1})=f(u_{i+1})\quad\text{et}\quad \mathscr{P}\left(\frac{u_i+u_{i+1}}{2}\right)=f\left(\frac{u_i+u_{i+1}}{2}\right).$$

Considérons, pour un ξ donné, la fonction g_ξ définie sur $[u_i,u_{i+1}]$ par

$$g_\xi : x \mapsto \mathscr{P}(x)-f(x)+\frac{K}{24}(x-u_i)(x-u_{i+1})\left(x-\frac{u_i+u_{i+1}}{2}\right)^2$$

la constante K étant choisie de sorte que $g_\xi(\xi)=0$.

On a ainsi $g_\xi(u_i)=g_\xi\left(\frac{u_i+u_{i+1}}{2}\right)=g_\xi(u_{i+1})=g_\xi(\xi)=0$.

Par application du théorème de Rolle, g_ξ' admet donc trois racines entre ces différentes valeurs.

Par ailleurs, $\frac{u_i+u_{i+1}}{2}$ est également racine de g_ξ', distincte des précédentes.

Toujours grâce au théorème de Rolle, on en déduit que g_ξ'' admet trois racines distinctes sur l'intervalle $[u_i,u_{i+1}]$, puis que $g_\xi^{(3)}$ en admet deux, et donc enfin qu'il existe un c dans $]u_i,u_{i+1}[$ vérifiant $g_\xi^{(4)}(c)=0$.

On a donc $K=-f^{(4)}(c)$, ce qui conduit à

$$\mathscr{P}(\xi)-f(\xi)=-\frac{f^{(4)}(c)}{24}(\xi-u_i)(\xi-u_{i+1})\left(\xi-\frac{u_i+u_{i+1}}{2}\right)^2$$

et donc à majorer l'erreur commise par

$$\varepsilon \leqslant \frac{M_4}{24}\int_{u_i}^{u_{i+1}}(\xi-u_i)(u_{i+1}-\xi)\left(\xi-\frac{u_i+u_{i+1}}{2}\right)^2 d\xi = M_4\frac{(u_{i+1}-u_i)^5}{2880}$$

Et finalement, on peut en déduire une majoration sur l'erreur commise lors de l'estimation de l'intégrale sur $[a,b]$:

$$\left|\hat{I}_{[a,b],n}(f)-I_{[a,b]}(f)\right| \leqslant \frac{M_4(b-a)^5}{2880n^4}$$

4 Utilisation du module scipy

Lorsque l'on a besoin d'intégrer une fonction, il n'est en général pas conseillé de programmer soi-même une méthode d'intégration numérique. Si l'on a présenté ici les différentes méthodes, c'est afin d'expliquer comment elles fonctionnent, et quelles difficultés on peut être amené à rencontrer.

Le module Python `scipy` contient une grande quantité de fonctions qui peuvent être utilisées dans le cadre de calcul « scientifique », et donc naturellement des fonctions permettant l'intégration numérique d'une fonction.

Comme le module contient un grand nombre de fonctions, elles ont été réparties en différents sous-modules. La fonction permettant de calculer l'intégrale d'une fonction sur un intervalle s'appelle `quad` et se trouve dans `scipy.integrate`.

Son utilisation est très simple, et il n'est pas nécessaire de choisir le niveau de subdivision, la fonction déterminera elle-même le niveau adéquat :

```
In []: from scipy.integrate import quad

In []: quad(sin, 0, pi)
Out[]: (2.0, 2.220446049250313e-14)

In []: quad(cos, 0, pi)
Out[]: (4.9225526349740854e-17, 2.2102239425853306e-14)
```

La fonction renvoie deux valeurs. La première est le résultat de l'intégration. La seconde est une estimation de l'erreur commise lors de l'intégration. Souvent, cette erreur est très petite, mais il convient quand même d'y faire attention.

En effet, de façon évidente,

$$\int_0^{384\pi} \cos(\xi)\,\mathrm{d}\xi = 0$$

Et pourtant, la fonction `quad` est bien loin de répondre 0 lorsque l'on tente de calculer cette intégrale :

```
In []: quad(cos, 0, 384*pi)
Out[]: (42.08833366982774, 451.16259740561156)
```

Le fait que la seconde valeur est très grande doit vous alerter sur le fait que l'intégration ne s'est pas bien passée, et que la fonction a rencontré des difficultés (sur lesquelles on reviendra en séance de travaux pratiques). Heureusement, cela arrive très, très rarement.

Signalons enfin que l'on peut calculer, grâce à cette fonction, des intégrales sur des intervalles allant jusqu'à l'infini. Pour ce faire, il faut préalablement importer la valeur `inf` qui se trouve dans le module `numpy`.

Cela permet ensuite de vérifier que

$$\int_{-\infty}^{0} e^{\xi}\,\mathrm{d}\xi = 1$$

```
In []: from numpy import inf

In []: quad(exp, -inf, 0.0)
Out[]: (1.0000000000000002, 5.842607038578007e-11)
```

5 Dérivation numérique

De la même façon que l'on a pu avoir besoin d'intégrer notre fonction f sur un intervalle $[a,b]$, on peut avoir besoin de connaître sa dérivée en un point x_0 (on supposera, cette fois encore, que la fonction est bien dérivable en ce point).

Une manière de calculer la dérivée $f'(x_0)$ de la fonction f en x_0, mathématiquement, peut être :

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

Le problème, d'un point de vue numérique, est que la notion de limite est complexe à mettre en oeuvre.

On peut, dans un premier temps, écrire une fonction calculant, pour un epsilon donné, la fraction ci-dessus :

```
def der(f, x0, eps) :
    return (f(x0+eps) - f(x0-eps)) / (2.0*eps)
```

Servons nous à présent de cette fonction pour essayer de déterminer la dérivée de la fonction $x \mapsto \sqrt{x}$ en $x_0 = 1$ (dont la valeur théorique est $1/(2\sqrt{1})$ soit 0.5).

```
In []: der(sqrt, x0=1.0, eps=1.0)
Out[]: 0.7071067811865476

In []: der(sqrt, x0=1.0, eps=0.1)
Out[]: 0.5006277505981893

In []: der(sqrt, x0=1.0, eps=0.01)
Out[]: 0.5000062502734492

In []: der(sqrt, x0=1.0, eps=1e-5)
Out[]: 0.5000000000032756
```

Au fur et à mesure que ϵ diminue, on se rapproche bien de la valeur correcte. Mais...

```
In []: der(sqrt, x0=1.0, eps=1e-6)
Out[]: 0.5000000000143778

In []: der(sqrt, x0=1.0, eps=1e-16)
Out[]: 0.5551115123125783

In []: der(sqrt, x0=1.0, eps=1e-17)
Out[]: 0.0
```

L'erreur, lorsque ϵ devient très faible, recommence à augmenter ! La raison en est simple, on cherche à calculer une différence de deux flottants de plus en plus proches. Le phénomène de « cancellation » joue ici à plein, et le résultat de la différence contient de moins en moins de chiffres significatifs. Jusqu'au moment où ϵ est tellement faible qu'il n'est plus possible de faire la différence entre $\sqrt{x_0+\epsilon}$ et $\sqrt{x_0-\epsilon}$, d'où un résultat nul !

Peut-on faire mieux ? En fait, pas vraiment. Si l'on regarde la fonction `derivative` du module `scipy.misc`, qui permet justement de calculer la dérivée, on retrouve ce paramètre ϵ parmi les arguments (facultatifs) de la fonction (sous le nom `dx`).

```
In []: from scipy.misc import derivative

In []: derivative(sqrt, 1.0)
Out[]: 0.70710678118654757

In []: derivative(sqrt, 1.0, dx=1e-6)
Out[]: 0.50000000001437783

In []: derivative(sqrt, 1.0, dx=1e-17)
Out[]: 0.0
```

On constate que les résultats sont pratiquement les mêmes que notre propre fonction, ce qui n'est pas étonnant car la méthode de calcul est en fait la même, faute d'une meilleure solution. Attention, la valeur par défaut pour ϵ est 1 ! C'est très fréquemment insuffisant pour obtenir même une vague approximation, donc si vous utilisez la fonction `derivative` (ce qui est plutôt recommandé plutôt que de programmer votre propre fonction), il est indispensable de choisir une valeur plus adaptée.

Malheureusement, il n'existe pas de règle universelle pour choisir un « bon » ϵ, cela dépend complètement de la fonction à dérivée. Aussi, il convient de garder en mémoire que la dérivation numérique pose beaucoup de problèmes, peut-être encore davantage que l'intégration numérique, et que s'il est possible de dériver formellement la fonction f pour obtenir f', cette solution sera toujours préférable, comme on le verra d'ailleurs ultérieurement.

6 Traitement de données

6.1 Une acquisition fournit des données discrètes !

Dans la pratique, on ne travaille pas toujours avec de véritables fonctions f, définies en tout point d'un intervalle $[a,b]$ ou au voisinage d'un point x_0. Par exemple, on peut avoir un capteur mesurant une grandeur physique dépendant du temps $f(t)$ (telle que la température en un point d'un matériau), et fournissant une grandeur mesurable $\hat{f}(t)$ (une tension proportionnelle à cette température).

Si l'on veut traiter ces données numériquement, et déterminer par exemple la température moyenne sur une période de temps $[t_a, t_b]$ (ce qui nécessite d'intégrer la température sur cet intervalle de temps), il n'est pas possible d'enregistrer $\hat{f}(t)$: cela correspond à une infinité de valeurs, ce qui nécessiterait à la fois une mémoire infinie et un temps de traitement nul pour chaque valeur.

En pratique, on n'enregistre que des mesures discrètres $\mathscr{F}_i$ associées à des instants t_i pris, en général régulièrement, dans l'intervalle $[t_a, t_b]$. Ces mesures discrètres $\mathscr{F}_i$ peuvent être liées à $\hat{f}(t_i)$, à une moyenne de $\hat{f}(t)$ sur l'intervalle $[t_i, t_{i+1}]$, ou encore à d'autres choses, selon la manière dont la conversion entre la grandeur analogique et la donnée numérique est implémentée.

Souvent, les choses se passent de la façon suivante :

- la grandeur physique $f(t)$ est convertie en une tension électrique $\hat{f}(t)$ via un capteur (thermistance, thermocouple, photodiode, phototransistor, jauge de contrainte, cellule de conductimètre...) associé à un circuit électrique ;
- un dispositif appelé *échantillonneur-bloqueur* « fige » la tension fournie par le capteur et son circuit à intervalles de temps réguliers pendant quelques instants, le temps de réaliser la mesure de la tension ;
- durant ces quelques instants où on dispose d'une tension constante, un dispositif de conversion analogique-digitale convertit cette tension en une valeur numérique, souvent un entier.

Finalement, on dispose à l'arrivée non pas d'une fonction $f(t)$ mais d'une liste de valeurs $\mathscr{F}_i$ associées à des instants t_i.

6.2 Intégration

Il est toujours possible d'approcher l'intégrale qui peut nous intéresser par les méthodes d'intégration présentées précédemment.

Cependant, il n'est évidemment plus possible de choisir librement la position des points pour les méthodes composites, ni le niveau de subdivision : la plus petite subdivision possible correspondra à la fréquence d'acquisition des données.

Ainsi, pour une acquisition régulière avec une période d'acquisition $\mathrm{d}t$, entre t_0 et $t_N = t_0 + N\,\mathrm{d}t$, l'intégrale entre t_0 et t_N sera associée, selon la méthode des trapèzes et en

choisissant la plus petite subdivision possible, à la quantité

$$\left(\frac{\mathscr{F}_0}{2}+\left(\sum_{i=1}^{N-1}\mathscr{F}_i\right)+\frac{\mathscr{F}_N}{2}\right)\times \mathrm{d}t$$

Ou en général, plus simplement, à

$$\left(\sum_{i=0}^{N-1}\mathscr{F}_i\right)\times \mathrm{d}t$$

(soit la méthode du rectangle gauche), les différences n'étant généralement pas significatives.

Il ne faudra pas oublier de déterminer le coefficient devant cette expression, si on veut interpréter le résultat, puisque l'on a effectué une double « conversion »

grandeur physique ⟼ grandeur électrique ⟼ valeur numérique

chacune de ces grandeurs ayant sa propre échelle et sa propre unité (par exemple, l'entier 129 désignera une tension de 4,2 V, qui elle-même fera référence à une température de 54 °C).

Là encore, on dispose de fonctions dans le module `scipy.integrate` pour calculer cette intégrale proprement.

Supposons par exemple que les données acquises se trouvent dans une liste `F` et que les instants où elles ont été obtenues se trouvent dans une liste `T`, comme ce serait par exemple le cas si l'on définissait les listes de la sorte :

```
In []: T = [ k*pi/100 for k in range(101) ]

In []: F = [ sin(ti) for ti in T ]
```

Si l'on souhaite déterminer une estimation de l'intégrale sur ces données en utilisant la méthode d'intégration numérique de Simpson[3], on dispose de la fonction `simps` du module `scipy.integrate`.

On l'utilise en lui fournissant en argument les deux listes `T` et `F` :

```
In []: from scipy.integrate import simps

In []: simps(F, T)
Out[]: 2.000000010824504
```

3. Attention, lorsque l'on travaille avec des données venant d'une acquisition, on n'a pas accès au « point milieu » nécessaire à la méthode de Simpson, aussi les mesures fournies sont alternativement considérées comme des valeurs aux bornes des intervalles et au milieu des intervalles. Autrement dit, le « pas » pour la méthode de Simpson ici serait $2\,\mathrm{d}t$ et non $\mathrm{d}t$. Cela suppose en principe qu'il y ait un nombre impair de mesures, la fonction propose différentes options pour gérer le cas d'un nombre pair de mesures.

Lorsque l'on ne dispose pas d'une liste `T` contenant les instants t_i, mais que la période d'acquisition $\mathrm{d}x$ est constante et connue (ici $\mathrm{d}x = \pi/100$), on peut également écrire :

```
In []: simps(F, dx = pi/100)
Out[]: 2.0000000108245044
```

Signalons enfin qu'il existe aussi une fonction `trapz` dans le même module qui réalise la même opération avec la méthode des trapèzes.

6.3 Dérivation

De la même façon, si l'on s'intéresse à la dérivée en un point, on ne pourra pas choisir un ϵ aussi petit que l'on veut. Une des meilleures estimations que l'on ait pour la dérivée à l'instant t_i sera donc associée à

$$\frac{\mathscr{F}_{i+1}-\mathscr{F}_{i-1}}{t_{i+1}-t_{i-1}}$$

ce qui, pour une acquisition régulière avec une période d'acquisition $\mathrm{d}t$, donne

$$\frac{\mathscr{F}_{i+1}-\mathscr{F}_{i-1}}{2\,\mathrm{d}t}$$

Parfois, on préférera à cette expression une expression légèrement différente,

$$\frac{\mathscr{F}_{i+1}-\mathscr{F}_{i}}{\mathrm{d}t}$$

Elle correspond simplement à une définition mathématique équivalente de la dérivée (pour une fonction f toujours dérivable en x_0) :

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

De même, on peut écrire pour la dérivée seconde

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

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

Ce qui amène à calculer la dérivée seconde numérique (en choisissant $\mathrm{d}t = 2\epsilon$) de la façon suivante :

$$\frac{\mathscr{F}_{i+1}-2\mathscr{F}_i+\mathscr{F}_{i-1}}{\mathrm{d}t^2}$$