TD N°4 : Intégration numérique

Ahmed Ammar (ahmed.ammar@fst.utm.tn)

Institut Préparatoire aux Études Scientifiques et Techniques, Université de Carthage.

Jan 19, 2020

Contents

Exercise 1: Vitesse d'une fusée

On lance une fusée verticalement du sol et l'on mesure pendant les premières 80 secondes l'accélération γ:

t[s]	0	10	20	30	40	50	60	70	80
γ [m s^{-2}]	30	31.63	33.44	35.47	37.75	40.33	43.29	46.70	50.67

Calculer la vitesse V de la fusée à l'instant $t = 80\ s$, par la méthode des trapèzes.

Solution. On sait que l'accélération γ est la dérivée de la vitesse V, donc,

$$V(t) = \int_0^t \gamma(s)ds$$

$$I = V(80) = \int_0^{80} \gamma(s)ds$$

Calculons I par la méthode des trapèzes. Ici, d'après le tableau des valeurs, $h = 10$.

$$I \approx h\left[\frac{1}{2}f(x_0) + \sum_{i=1}^{n-1} f(x_i) + \frac{1}{2}f(x_n)\right]$$

$$\begin{aligned} I &\approx 10\left[\frac{1}{2}\times 30 + \frac{1}{2}\times 50.67 + 31.63 + 33.44 + \ldots + 46.70\right] \\ &\approx 3089.45 \quad ms^{-1} \end{aligned}$$

```
h = 10
I = 0.5 * (30 + 50.67) # 1/2 * [f(x0) + f(xn)]
fx = [31.63,33.44,35.47,37.75,40.33,43.29,46.70] # f(x1) ---> f(xn-1)
for i in range(len(fx)):
    I+= fx[i]
I*=h
print(I, "ms^-1")
```

Exercise 2: Valeur approchée de π

Étant donnée l'égalité:

$$\pi = 4\left(\int_0^{\infty} e^{-x^2}dx\right)^2 = 4\left(\int_0^{10} e^{-x^2}dx + \epsilon\right)^2 \tag{1}$$

avec $0 < \epsilon < 10^{-44}$, utiliser la méthode des trapèzes composite à 10 intervalles pour estimer la valeur de π.

Solution. La méthode des trapèzes composite à n intervalles pour calculer l'intégrale d'une fonction f sur l'intervalle $[a, b]$ s'écrit

$$\int_a^b f(x)\,dx \approx h\left[\frac{1}{2}f(x_0) + \sum_{i=1}^{n-1} f(x_i) + \frac{1}{2}f(x_n)\right]$$

avec $h = \frac{b-a}{n}$ et $x_i = a + ih, \quad i = 0, 1, \ldots, n$

Ici on a $f(x) = e^{-x}$, $a = 0$, $b = 10$, $n = 10$ d'où $h = 1$ et on obtient

$$I \approx \frac{1}{2} + \sum_{i=1}^{9} e^{-i} + \frac{1}{2e^{100}} = \frac{1}{2} + \frac{1}{e} + \frac{1}{e^4} + \frac{1}{e^9} + \frac{1}{e^{16}} + \frac{1}{e^{25}} + \frac{1}{e^{36}} + \frac{1}{e^{49}} + \frac{1}{e^{64}} + \frac{1}{e^{81}} + \frac{1}{2e^{100}}$$

ainsi en utilisant la fonction `trapeze(f,a,b,n)` du module `trapeze_integral.py` décrite dans le cours comme suit

```
from trapeze_integral import trapeze
from math import exp
f = lambda x: exp(-x**2)
I = trapeze(f, 0, 10, 10)
print(4*I**2)
```

on obtient $\pi \approx 4I^2 = 3.14224265994$.

Exercise 3: Intégration adaptative

Supposons que nous voulons utiliser la méthode des trapèzes ou du point milieu pour calculer une intégrale $\int_a^b f(x)dx$ avec une erreur inférieure à une tolérance prescrite ϵ. Quelle est la taille appropriée de n?

Pour répondre à cette question, nous pouvons entrer une procédure itérative où nous comparons les résultats produits par n et $2n$ intervalles, et si la différence est inférieure à ϵ, la valeur correspondant à $2n$ est retournée. Sinon, nous avons n et répétons la procédure.

Indication. Il peut être une bonne idée d'organiser votre code afin que la fonction `integration_adaptive` peut être utilisé facilement dans les programmes futurs que vous écrivez.

a) Écrire une fonction `integration_adaptive(f, a, b, eps, method=midpoint)` qui implémente l'idée ci-dessus (`eps` correspond à la tolérance ϵ, et la méthode peut être `midpoint` ou `trapeze`).

Solution. En facilitant les réponses à l'ensemble de cet exercice, ainsi qu'en préparant l'utilisation facile de l'intégration adaptative dans les futurs programmes, nous organisons le codage des tâches a, b et c en un seul fichier, ce qui en fait un module. Le code se lit alors:

```
from numpy import linspace, zeros, sqrt, log
from trapezoidal import trapezoidal
from midpoint import midpoint

def adaptive_integration(f, a, b, eps, method='midpoint'):
    n_limit = 1000000   # Just a choice (used to avoid inf loop)
    n = 2
    if method == 'trapezoidal':
        integral_n  = trapezoidal(f, a, b, n)
        integral_2n = trapezoidal(f, a, b, 2*n)
        diff = abs(integral_2n - integral_n)
        print 'trapezoidal diff: ', diff
        while (diff > eps) and (n < n_limit):
            integral_n  = trapezoidal(f, a, b, n)
            integral_2n = trapezoidal(f, a, b, 2*n)
            diff = abs(integral_2n - integral_n)
            print 'trapezoidal diff: ', diff
            n *= 2
    elif method == 'midpoint':
        integral_n  = midpoint(f, a, b, n)
        integral_2n = midpoint(f, a, b, 2*n)
        diff = abs(integral_2n - integral_n)
        print 'midpoint diff: ', diff
        while (diff > eps) and (n < n_limit):
            integral_n  = midpoint(f, a, b, n)
            integral_2n = midpoint(f, a, b, 2*n)
            diff = abs(integral_2n - integral_n)
            print 'midpoint diff: ', diff
            n *= 2
    else:
        print 'Error - adaptive integration called with unknown par'
    # Now we check if acceptable n was found or not
    if diff <= eps:   # Success
        print 'The integral computes to: ', integral_2n
        return n
    else:
        return -n   # Return negative n to tell "not found"

def application():
    """...Tasks b) and c)"""

    def f(x):
        return x**2
    def g(x):
```

```
        return sqrt(x)

    #eps = 1E-1           # Just switch between these two eps values
    eps = 1E-10
    #a = 0.0
    a = 0.0 + 0.01;       # If we adjust a, sqrt(x) is handled easily
    b = 2.0
    # ...f
    n = adaptive_integration(f, a, b, eps, 'midpoint')
    if n > 0:
        print 'Sufficient n is: %d' % (n)
    else:
        print 'No n was found in %d iterations' % (n_limit)

    n = adaptive_integration(f, a, b, eps, 'trapezoidal')
    if n > 0:
        print 'Sufficient n is: %d' % (n)
    else:
        print 'No n was found in %d iterations' % (n_limit)

    # ...g
    n = adaptive_integration(g, a, b, eps, 'midpoint')
    if n > 0:
        print 'Sufficient n is: %d' % (n)
    else:
        print 'No n was found in %d iterations' % (n_limit)

    n = adaptive_integration(g, a, b, eps, 'trapezoidal')
    if n > 0:
        print 'Sufficient n is: %d' % (n)
    else:
        print 'No n was found in %d iterations' % (n_limit)

    # Task c, make plot for both midpoint and trapezoidal
    eps = linspace(1E-1,10E-10,10)
    n_m = zeros(len(eps))
    n_t = zeros(len(eps))
    for i in range(len(n_m)):
        n_m[i] = adaptive_integration(g, a, b, eps[i], 'midpoint')
        n_t[i] = adaptive_integration(g, a, b, eps[i], 'trapezoidal')

    import matplotlib.pyplot as plt
    plt.plot(log(eps),n_m,'b-',log(eps),n_t,'r-')
    plt.xlabel('log(eps)')
    plt.ylabel('n for midpoint (blue) and trapezoidal (red)')
    plt.show()
    print n
    print eps

if __name__ == '__main__':
    application()
```

b) Testez la méthode sur $\int_0^2 x^2 dx$ et $\int_0^2 \sqrt{x} dx$ pour $\epsilon = 10^{-1}, 10^{-10}$ et notez l'erreur exacte.

Solution. Voir le code ci-dessus. Notez que, dans notre code suggéré, il est prévu que le programmeur bascule entre les deux valeurs epsilon en utilisant des commentaires, c'est-à-dire en ajoutant/supprimant `#`. Cela peut bien sûr

être évité, si cela est souhaitable, en demandant à l'utilisateur d'entrer un valeur d'epsilon.

c) Faites un tracé de n en fonction de $\epsilon \in [10^{-1}, 10^{-10}]$ pour $\int_0^2 \sqrt{x}dx$. Utilisez l'échelle logarithmique pour ϵ.

Solution. La figure 14 montre que plus la valeur d'epsilon est stricte, plus la différence entre les méthodes du point milieu et des trapèzes est grande. Pour une valeur donnée pour epsilon, la méthode du point médian se situe dans la tolérance avec moins d'intervalles (valeur inférieure de n) que la méthode des trapèzes. Il faut s'y attendre, car la méthode du point milieu est un peu plus précise que la méthode des trapèzes.

Une façon de produire ce tracé est d'utiliser les lignes de code incluses à la fin de `integration_adaptive.py` (voir ci-dessus).

Remarks. Le type de méthode exploré dans cet exercice est appelé *adaptatif*, car il essaie d'adapter la valeur de n pour répondre à un critère d'erreur donné. La vraie erreur peut très rarement être calculée (car nous ne connaissons pas la réponse exacte au problème de calcul), il faut donc trouver d'autres indicateurs de l'erreur, comme celui ici où les changements de la valeur intégrale, comme le nombre d'intervalles est doublé, est pris pour refléter l'erreur.

Exercise 4: Intégration de x élevé à x

Considérons l'intégrale

$$I = \int_0^2 x^x \, dx \, .$$

L'intégrande x^x n'a pas de primitive qui peut être exprimé en termes de fonctions standard (visitez `http://wolframalpha.com` et tapez `integral x^x dx from 0 to 2` pour vous convaincre que notre affirmation est juste. Notez que Wolfram alpha vous donne une réponse, mais cette réponse est une approximation, elle n'est pas *exacte*. C'est parce que Wolfram alpha utilise également des méthodes numériques pour arriver à la réponse, comme vous le ferez dans cet exercice). Par conséquent, nous sommes obligés de calculer l'intégrale par des méthodes numériques. Calculez un résultat composé de quatre chiffres.

Indication. Utilisez des idées de l'exercice 3.

Solution. Lorsque la fonction `integration_adaptive` est disponible, le code peut s'écrire:

```
from integration_adaptive import integration_adaptive

def f(x):
    return x**x
```

```
eps = 1E-4
a = 0.0;  b = 2.0

# Choose midpoint method
n = integration_adaptive(f, a, b, eps, 'midpoint')
if n > 0:
    print 'Sufficient n is: %d' % (n)
else:
    # The negative n is returned to signal that the upper limit of n
    # was passed
    print 'No n was found in %d iterations' % (abs(n))
```

N'oubliez pas que `integration_adaptive` affiche l'intégrale calculée, donc aucun effort à cet égard n'est requis ici.

L'exécution du programme donne une impression à l'écran montrant comment la différence devient de plus en plus petite à chaque fois. Les deux dernières lignes de l'impression se lisent comme suit:

```
The integral computes to: 2.83384395958
Sufficient n is: 256
```

La valeur calculée peut être comparée à ce que donne Wolfram alpha. N'oubliez pas qu'étant donné que nous n'avons pas calculé l'erreur exacte, nous ne pouvons garantir que le "résultat est correct à quatre chiffres". Cependant, nous avons des raisons de croire que nous "sommes proches". Typiquement, lorsque l'on sait que la mesure d'erreur utilisée n'est pas précise, la tolérance est rendue plus stricte.

Exercise 5: Orbitales atomiques

Pour décrire la trajectoire d'un électron autour d'un noyau, une description probabiliste est adoptée : l'électron n'est plus caractérisé par ses coordonnées spatiales mais par sa *probabilité de présence* en un point de l'espace.

Pour simplifier le problème, on considérera que cette probabilité de présence ne dépend que de la variable r, distance entre l'électron et le centre du noyau. Pour une orbitale $1s$, la probabilité de trouver l'électron entre les rayons r_1 et r_2 s'écrit :

$$P_{s1} = \int_{r_1}^{r_2} \underbrace{4 \times \frac{r^2}{a_0^3} \times e^{-2\times\frac{r}{a_0}}}_{\text{densité radiale}} \, dr$$

avec $a_0 = 0.529$ Å, appelé le rayon de Bohr.

La densité radiale, représentée dans la figure 1, est maximale pour $r = a_0$. Ce rayon qui maximise la densité radiale est appelé le *rayon orbitalaire*.

À noter
Dans ce problème, les distances seront conservées en Angström.

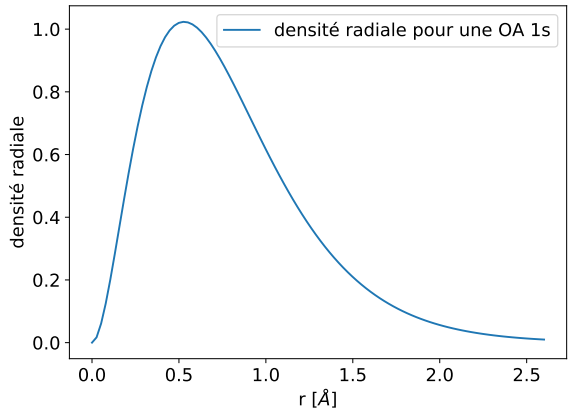

Figure 1: Densité radiale pour une orbitale atomique 1s.

a) Définir une fonction `densite_radiale()`, définie entre 0 et ∞ qui prend comme paramètre variable un rayon r et comme paramètre par défaut $a_0 = 0.529$ Å et renvoie la valeur $4 \times \frac{r^2}{a_0^3} \times e^{-2\times\frac{r}{a_0}}$.

b) Tracer la densité radiale pour $r \in [0, 2.6]$ Å, afin d'obtenir le même graphique sur la figure 1.

c) On souhaite déterminer la probabilité de présence de l'électron entre 0 et a_0. Évaluer cette probabilité à l'aide de 100 rectangles. On pourra vérifier que la réponse obtenue est proche de 0.32.

d) Déterminer le nombre entier n, tel que l'électron ait une probabilité supérieure ou égale à 90% de se trouver entre 0 et $n * a_0$.

e) On souhaite désormais évaluer la probabilité de trouver l'électron proche du rayon de Bohr, c'est-à-dire entre $0.9 * a_0$ et $1.1 * a_0$. Évaluer cette probabilité à l'aide de 100 rectangles.

f) D'après la valeur obtenue à la question précédente, que penser de la description des trajectoires des électrons par orbite autour du noyau ?

À noter
On répondra en commentaire dans le programme.

Solution. La solution de l'exercice est dans le programme python suivant:

```python
## NOM DU PROGRAMME: OrbitalesAtomiques.py
#% IMPORTATION
import numpy as np
import matplotlib.pyplot as plt
from midpoint_integral import midpoint
import matplotlib
matplotlib.rcParams.update({'font.size': 14})
# a)
def densite_radiale(r, a0 = 0.529):
    return 4 * (r**2/a0**3) * np.exp(-2*(r/a0))
# b)
r = np.linspace(0,2.6, 100)

plt.figure(figsize=(7,5))
plt.plot(r, densite_radiale(r), label = "densité radiale pour une OA 1s")
plt.xlabel("r "+r"$[\AA]$")
plt.ylabel("densité radiale")
plt.legend()
plt.savefig("../imgs/densite_radiale.pdf", dpi = 200); plt.savefig("../imgs/densite_radiale.png")
plt.show()

# c) probabilité de présence de l'électron entre 0 et a0
a0 = 0.529
Pa0 = midpoint(densite_radiale, 0, a0, 100)
print("La probabilité de présence de l'électron entre 0 et a0 est: ", Pa0)

# d) rayon moyen de l'OA 1s

for n in range(21):
    Prn = midpoint(densite_radiale, 0, n*a0, 100)
    if Prn >= 0.90:
        print("n = %d, Pr%d = %.4f"%(n,n,Prn))

'''
r90 = 3*a0. On dit donc que la rayon moyen de l'OA 1s de l'atome d'hydrogène
est une sphère de rayon 3*a0, soit à peu près 1.6 Å.
'''

# e)
Pr = midpoint(densite_radiale, 0.9*a0, 1.1*a0, 100)
print("La probabilité de présence de l'électron entre 0.9*a0 et 1.1*a0 est: ", Pr)

# La probabilité de présence de l'électron entre 0.9*a0 et 1.1*a0 est:  0.10790737203009312

# f)

'''
* La densité radiale de probabilité de présence est maximale
pour r = a0 (rayon de Bohr) On dit que c'est le rayon le plus probable.

* Ce résultat est trompeur, car entre 0.9*a0  et 1.1*a0, la probabilité de présence
de l'électron n'est que de 11%.

* l'atome d'hydrogène est une sphère de rayon 3*a0, soit à peu près 1.6 Å.

* La probabilité de présence de l'électron 1s est plus élevée à l'extérieur
de l'orbite de Bohr.
'''
```