Lycée Louis-Le-Grand, Paris
MPSI 4 – Informatique commune
N. Carré, A. Troesch

2018/2019

TP n° 8 : Calcul numérique d'intégrales

Nous étudions dans ce TP plusieurs méthodes de calcul approché d'intégrales.

Correction de l'exercice 1 –

1. Nous obtenons les fonctions suivantes :

```
from scipy.integrate import quad
from math import exp

def rectanglegauche(f,a,b,n):
    S = 0
    x = a
    pas = (b - a) / n
    for i in range(n):
        S += f(x)
        x += pas
    return (S * pas)

def rectangledroit(f,a,b,n):
    S = 0
    pas = (b - a) / n
    x = a + pas
    for i in range(n):
        S += f(x)
        x += pas
    return (S * pas)

def pointmilieu(f,a,b,n):
    S = 0
    pas = (b - a) / n
    x = a + pas / 2
    for i in range(n):
        S += f(x)
        x += pas
    return (S * pas)

def trapeze(f,a,b,n):
    S = f(a)+f(b)
    pas = (b - a) / n
    x = a + pas
    for i in range(n-1):
        S += 2 * f(x)
        x += pas
    return (S * (b-a) / (2*n))
```

```
def simpson(f,a,b,n):
    S = 0
    pas = (b-a) / (2*n)
    valgauche = f(a)
    x = a
    for i in range(n):
        x += pas
        S += valgauche + 4 * f(x)
        x+= pas
        valgauche = f(x)
        S += valgauche
    return (S * pas / 3)
```

Nous avons fait le test sur $\int_0^2 \mathrm{e}^{-\frac{x^2}{2}}\,\mathrm{d}x$:

```
from scipy.integrate import quad
from math import exp

n = eval(input('n? '))

def f(x):
    return (exp(-(x**2)/2))

print('Méthode rectangle gauche:')
print(rectanglegauche(f,0,2,n))
print('Méthode rectangle droit:')
print(rectangledroit(f,0,2,n))
print('Méthode point milieu:')
print(pointmilieu(f,0,2,n))
print('Méthode trapèze:')
print(trapeze(f,0,2,n))
print('Méthode simpson:')
print(simpson(f,0,2,n))
print('Valeur obtenue avec quad, et erreur:')
print(quad(f,0,2))
```

Remarquez la possibilité de passer une fonction en paramètre. Remarquez également qu'on aurait pu se servir de `lambda` permettant d'introduire une fonction sans la nommer : `quad(lambda x: exp(-(x**2)/2),0,1)`

Le résultat retourné est, pour $n = 1000$:

```
Méthode rectangle gauche:
1.1967203231251315
Méthode rectangle droit:
1.1958556584083682
Méthode point milieu:
1.1962880246005698
Méthode trapèze:
1.1962879230990897
Méthode simpson:
1.1962880133226301
Valeur obtenue avec quad, et erreur:
(1.1962880133226081, 1.3281464964738456e-14)
```

On peut déjà se rendre compte à ce stade de la qualité comparative de ces méthodes :

- Les deux méthodes du rectangle sont comparables et peu efficaces ;
- Les méthodes du point milieu et des trapèzes sont déjà meilleures, relativement comparables, avec un avantage toutefois pour la méthode du point milieu ;
- La méthode de simpson est la plus efficace, puisqu'avec une subdivision en 1000 intervalles, on obtient 13 décimales correctes.

Ces résultats sont cohérents avec la convergence en $O\left(\frac{1}{n}\right)$ trouvée dans le cours pour les deux premières méthodes, en $O\left(\frac{1}{n^2}\right)$ pour les deux suivantes, et en $O\left(\frac{1}{n^4}\right)$ pour la méthode de Simpson.

2. De façon synthétique, en passant la méthode choisie en paramètre :

```
def f(x):
    return (exp(-(x**2)/2))

def rangnecessaire(f,a,b,err,methode):
    I, err1 = quad(f,a,b)
    err2 = err - err1
    n = 1
    J = methode(f,a,b,n)
    while abs(I-J)> err2:
        n += 1
        J = methode(f,a,b,n)
    return(n)

err = eval(input('err?'))
print('Erreur␣autorisée:␣', err)
if err > 1e-5:
    print('rectangle␣droit:␣', rangnecessaire(f,0,1,err,rectangledroit))
    print('rectangle␣gauche:␣', rangnecessaire(f,0,1,err,rectanglegauche))
if err > 1e-11:
    print('point␣milieu:␣', rangnecessaire(f,0,1,err,pointmilieu))
    print('trapèze:␣', rangnecessaire(f,0,1,err,trapeze))
print('simpson:␣', rangnecessaire(f,0,1,err,simpson))
```

Pour une erreur de 10^{-4}, 10^{-8} et 10^{-10}, 10^{-13}, on trouve successivement :

```
Erreur autorisée: 0.0001
rectangle droit: 1968
rectangle gauche: 1968
point milieu: 16
trapèze: 23
simpson: 2

Erreur autorisée: 1e-08
point milieu: 1590
trapèze: 2249
simpson: 15

Erreur autorisée: 1e-10
point milieu: 15895
trapèze: 22474
simpson: 46

Erreur autorisée: 1e-13
```

```
simpson: 261
```

Si on essaye de lancer les deux premières méthodes pour 10^{-8}, l'ordinateur ne répond plus. Pour les deux suivantes et une erreur de 10^{-10}, le temps de réponse dépasse la minute (avec mon ordinateur). Dans tous les cas, la méthode de Simpson renvoie une réponse de façon quasi-immédiate.

Ces résultats confortent donc les résultats comparatifs théoriques, y compris la comparaison relative de la méthode du point milieu, et de la méthode du trapèze. Remarquez au passage la redoutable efficacité de la méthode de Simpson !

3. La définition de Φ est :

```
def Phi(x):
    return simpson(lambda y: exp(-y**2/2),0,x,100)
```

À quelques constantes près, il s'agit de la fonction de répartition d'une loi normale centrée réduite. Plus précisément, la fonction de répartition de la loi normale centrée réduite s'exprime par :

$$\forall x \in \mathbb{R}, \quad \tilde{\Phi}(x) = \frac{1}{2} + \frac{1}{\sqrt{2\pi}}\Phi(x).$$

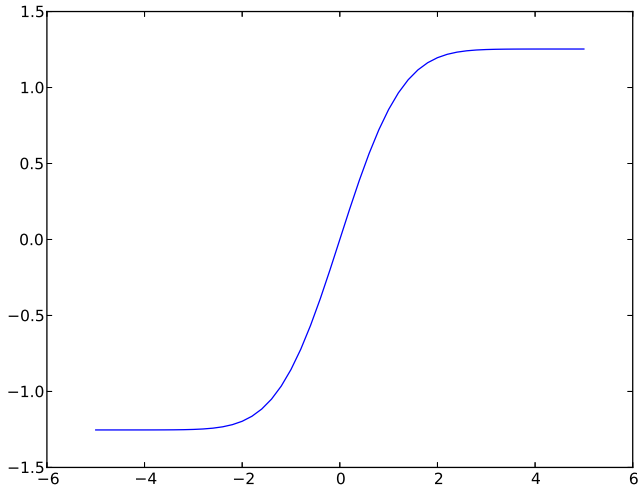
Il n'existe pas d'expression de Φ à l'aide des fonctions usuelles. La résolution numérique de nombreux problèmes de probabilités et de statistiques passe par le calcul approché de Φ (par des tables de valeurs, ou par l'outil informatique).

4. On obtient alors le graphe de Φ de la manière suivante :

```
import matplotlib.pyplot as pl

def Graphe(f,a,b,n):
    abscisses = []
    ordonnees = []
    x = a
    pas = (b-a)/n
    for i in range(n+1):
        abscisses.append(x)
        ordonnees.append(f(x))
        x += pas
    return(abscisses,ordonnees)

abs, ord = Graphe(Phi,-5,5,50)

pl.show(pl.plot(abs,ord))
```

Remarquez la possibilité de donner un alias à un nom de module, de sorte à ne pas avoir à l'écrire entièrement par la suite.

On nous renvoie alors dans une fenêtre séparée le graphe suivant :



5. Pour calculer $I = \int_{-\infty}^{+\infty} \mathrm{e}^{-\frac{x^2}{2}}\,\mathrm{d}x$, on peut intégrer entre $-A$ et A, pour une grande valeur de A, ce qui donne un résultat assez satisfaisant du fait de la convergence rapide de l'exponentielle vers 0. On peut aussi faire subir un prétraitement à cette intégrale, à l'aide du changement de variable $y = \mathrm{Arctan}(x)$, qui donne :

$$I = \int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} (1+\tan^2(y))\mathrm{e}^{-\frac{\tan(y)^2}{2}}\,\mathrm{d}y.$$

Remarquez qu'on a fait un changement de variable sur une intégrale impropre, ce qui est autorisé (pour un changement de variable de classe $\mathcal{C}^1$ et strictement croissant). Pour vous en persuader, vous pouvez commencer par faire le changement de variables sur $[-A, A]$, puis faire tendre A vers l'infini. Vous remarquerez aussi que l'on obtient après changement de variable une intégrale définie, quitte à prolonger l'intégrande par continuité par 0 aux deux bornes. On obtient alors :

```
from math import exp, tan, pi, sqrt

def g(x):
    return(exp(-tan(x)**2/2)* ((1+tan(x)**2)))

print(simpson(g,-pi / 2,pi / 2,100))
print(sqrt(2*pi))
```

On obtient les deux valeurs suivantes, la première étant la valeur calculée, la seconde étant la valeur théorique $\sqrt{2\pi}$.

```
2.5066282746133557
2.5066282746310002
```

On peut constater qu'en élevant le résultat obtenu au carré et en le divisant par 2, on obtient : 3.141592653545564, ce qui nous donne π avec 10 décimales correctes.

Correction de l'exercice 2 – Méthode de Monte-Carlo

1. Voilà ce qu'on obtient pour le calcul d'une intégrale avec cette méthode, en effectuant n tirages aléatoires :

```
from random import random
from scipy.integrate import quad
from math import exp, sqrt
```

```
def montecarlo1(f,a,b,n):
    S = 0
    for i in range(n):
        S += f(random()*(b-a)+a)
    return(S *(b-a) / n)
```

Pour évaluer le nombre moyen de tirages à effectuer pour obtenir une approximation à une erreur donnée près, on ne peut pas réutiliser cette fonction, qui nécessite de connaître à l'avance le nombre de tirages nécessaires. Nous écrivons donc (avec les mêmes imports) :

```
def rangnecessaire(f,a,b,erreur):
    S =  f(random()*(b-a)+a)
    (I,err)= quad(f, a,b)
    err1 = 1e-6 - err
    n = 1
    while abs(S * (b-a) / n - I) > 1e-6:
        S += f(random()*(b-a)+a)
        n += 1
    return(S *(b-a) / n, n)

erreur=eval(input('erreur?'))
integrale, rang = rangnecessaire(lambda x: exp(-x**2/2),0,1,erreur)
print('valeur:␣', integrale,'\nrang:␣',rang)
```

En essayant plusieurs fois, on obtient une grande variabilité des résultats. Voici par exemple le résultat de 4 essais successifs :

```
 erreur?1e-6
valeur: 0.8556243046444224
rang: 474

erreur?1e-6
valeur: 0.855623780427462
rang: 153245

erreur?1e-6
valeur: 0.8556240221576596
rang: 2425

erreur?1e-6
valeur: 0.8556234785208694
rang: 355
```

Nous allons effectuer une moyenne sur plusieurs expériences pour essayer d'obtenir une valeur significative.

```
erreur=eval(input('erreur?'))
rg = 0
for i in range(1000):
    integrale, rang = rangnecessaire(lambda x: exp(-x**2/2),0,1,erreur)
    rg += rang
print('rang␣moyen:␣',rg / 1000)
```

Après plus d'une heure de calculs, mon ordinateur me renvoie 3600065.1, ce qui semble excessif au vu des 4 valeurs précédentes. Des essais (avec un nombre plus petit de termes) semblent montrer que globalement, les valeurs sont nettement plus petites, mais que certains essais ratent complètement.

2. On tire cette fois aléatoirement les deux coordonnées, indépendamment. Cela donne la fonction suivante :

```
def montecarlo2(f,a1,b1,a2,b2,n):
    S = 0.
    for i in range(n):
        S += f(random()*(b1-a1)+a1, random()*(b2-a2)+a2)
    return(S *(b1-a1)*(b2-a2) / n)
```

En intégrant la fonction caractéristique d'un quart de disque, on obtient l'aire de ce quart de disque, à savoir $\frac{\pi}{4}$.

```
def disque(x,y):
    if x**2+y**2 <= 1:
        return(1)
    else:
        return(0)

print(4 * montecarlo2(disque,0,1,0,1,1000000))
```

Lors d'un essai, j'ai obtenu la valeur 3.140608. Cela reste assez peu efficace.

3. On rappelle le volume d'une boule de rayon r : $V = \frac{4}{3}\pi r^3$.

```
def boule(x,y):
    if x**2+y**2 <= 1:
        return(sqrt(1-x**2-y**2))
    else:
        return(0)

print(2 * montecarlo2(boule,-1,1,-1,1,1000000))
print(4 / 3 * pi )
```

Les deux valeurs renvoyées (valeur calculée, valeur théorique) sont :

```
4.186619283081323
4.1887902047863905
```

Correction de l'exercice 3 –
Si une fonction de n variable admet un minimum en $A = (a_1, \dots, a_n)$ de l'intérieur de son domaine, la fonction d'une variable obtenue en fixant $n-1$ des variables égales aux a_i et en faisant varier la variable restante admet aussi un minimum en ce point. On en déduit que toutes les dérivées partielles $\dfrac{\partial F}{\partial x_i}(A)$ sont nulles, ce qui s'exprime synthétiquement par $\nabla F(A) = 0$. Nous avons utilisé le fait que A soit à l'intérieur du domaine et non au bord pour nous assurer que les fonctions partielles (obtenues en fixant toutes les variables sauf une) sont définies sur tout un voisinage de a_i, condition à donner pour pouvoir affirmer qu'un extremum correspond à un point critique.
La fonction F est continue sur le fermé borné $\{(x_1, \dots, x_n) \in [0,1]^n \mid x_1 \leqslant x_2 \leqslant \cdots \leqslant x_n\}$, elle admet donc un minimum en un certain point $(a_1, \dots, a_n)$. Ce point vérifie nécessairement $0 < a_1 < \cdots < a_n < 1$. En effet, si l'une de ces inégalités est une égalité, cela permet de subdiviser une fois de plus un des intervalles de longueur non nulle de la subdivision, ce qui donnera une valeur plus petite, par décroissance de f, et $(a_1, \dots, a_n)$ ne peut dans ce cas pas être le minimum. Ainsi, F atteint son minimum dans l'ouvert $\{(x_1, \dots, x_n) \in [0,1]^n \mid x_1 \leqslant x_2 \leqslant \cdots \leqslant x_n\}$, ce qui assure qu'en ce point, le gradient est nul.
Ceci conduit aux égalités :

$$\forall k \in [\![1, n-1]\!],\ f(a_{k-1}) - f(a_k) + (a_{k+1} - a_k) f'(a_k) = 0$$

soit encore : $\forall k \in [\![1, n-1]\!]$, $a_{k+1} = a_k - \frac{f(a_{k-1}) - f(a_k)}{f'(a_k)} = 2a_k + \frac{\sqrt{1-a_{k-1}^2}\sqrt{1-a_k^2} - 1}{a_k}$.
Ces égalités permettent de définir toutes les valeurs a_k en fonction de la seule valeur a_1 (avec $a_0 = 0$) et donc de ramener le problème à la recherche de la valeur minimale d'une fonction d'une variable.
Il est temps de définir cette fonction à minimiser :

```
from math import sqrt

def f(x):
    return sqrt(1-x**2)

def F(x, n=200):
    a = [0, x]
    for k in range(1,n):
        a.append(2*a[-1]+(f(a[-1])*f(a[-2])-1)/a[-1])
    a.append(1)
    s = 0
    for k in range(0,n+1):
        s += (a[k+1]-a[k])*f(a[k])
    return 4*s
```

Quelques essais numériques montrent que la fonction F cesse d'être définie pour $x > 0.04$. On débute donc la recherche du minimum de F à partir de $x = 0,035$ et d'un pas $p = 0,001$ en procédant ainsi :

- tant que $F(x+p) < F(x)$ on remplace x par $x+p$;
- lorsque $F(x+p) \geq F(x)$ on remplace x par $x-p$ et p par $p/10$;

et ce jusqu'à atteindre $p \leq 10^{-11}$.
Ceci conduit au script :

```
x = 0.035
p = 0.0001
while p > 1e-11:
    if F(x+p) < F(x):
        x += p
    else:
        x -= p
        p /= 10
print F(x)
```

qui fournit finalement la valeur optimale $\pi \approx 3.1486734435$.