2017-03-21-191413

March 21, 2017

Correction du TP9 d'Informatique
Activité 1
Q1)

On implémente dans un premier temps la méthode des rectangles, qui corrrespond aux approximations :

$\int_a^b f(x)\,dx \approx h\sum_{k=1}^{N} f(x_{k-1})$;
$\int_a^b f(x)\,dx \approx h\sum_{k=1}^{N} f(x_k)$.

où h est le pas de la subdivision (régulière) de l'intervalle $[a;\ b]$ en N intervalles de même longueur. Ainsi, $h = \dfrac{b-a}{N}$ et $x_k = a + kh$.

```
In [32]: #import des packages nécessaires
         from pylab import *
         from scipy.integrate import romb
```

```
In [33]: def rectangle(X,Y):
             #X correspond à l'échantillon des valeurs de xi
             #Y correspond à l'échantillon des valeurs de f(xi)
             Sd=0
             Sg=0
             N=len(X)-1
             h=X[1]-X[0]
             for k in range(1,N+1):
                 Sd=Sd+Y[k]
                 Sg=Sg+Y[k-1]
             return [h*Sg,h*Sd]
```

On code de la même façon, la méthode des trapèzes :

```
In [34]: def trapeze(X,Y):
             #X correspond à l'échantillon des valeurs de xi
             #Y correspond à l'échantillon des valeurs de f(xi)

             T=0
             h=X[1]-X[0]
             N=len(X)-1
             for k in range(1,N):
                 T=T+Y[k]
             return h*T +h/2*(Y[0]+Y[N])
```

Le point milieu :

```
In [35]: def pointmilieu(X,Y):
             #X correspond à l'échantillon des valeurs de xi
             #Y correspond à l'échantillon des valeurs de f(xi)
             #N est pair pour la methode du point milieu
             T=0
             h=X[2]-X[0]
             N=len(X)-1
             for k in range(0,N//2):
                 T=T+Y[2*k+1]
             return h*T
```

Ainsi que la méthode Simpson :

```
In [36]: def simpson(X,Y):
             #X correspond à l'échantillon des valeurs de xi
             #Y correspond à l'échantillon des valeurs de f(xi)

             S=0
             h=X[2]-X[0]
             N=len(X)-1
             for k in range(0,N//2):
                 S=S+Y[2*k]+4*Y[2*k+1]+Y[2*k+2]
             return h/6*S
```

Q2)

On teste chacune des méthodes pour évaluer $\int_0^{\pi/2} \cos(x)\, dx$, dont on sait que la valeur exacte est 1.

```
In [37]: def f1(x):
             return cos(x)
```

```
In [38]: X=linspace(0,pi/2,65)
         Y=[f1(x) for x in X]
         print(rectangle(X,Y))
         print(trapeze(X,Y))
         print(pointmilieu(X,Y))
         print(trapeze(X,Y))
```

```
[1.0122216463951863, 0.98767795378901602]
0.999949800092
1.00010040586
0.999949800092
```

Q3)

On calcule de la même façon l'intégrale I_1 :

```
In [39]: def f2(x):
             return cos(x*x)
```

```
In [40]: X=linspace(0,1,65)
         Y=[f1(x) for x in X]
         print(rectangle(X,Y))
         print(trapeze(X,Y))
         print(pointmilieu(X,Y))
         print(trapeze(X,Y))
```

```
[0.84504525320262802, 0.83786247673181757]
0.841453864967
0.841505225325
0.841453864967
```

On commence par créer une fonction, qui prend en argument la subdivision du temps (X) ainsiq que les valeurs de cos associées (Y), et retourne les différentes valeurs des approximations associées à nos quatre méthodes :

```
In [41]: def approx(X,Y):
             X1=[]
             Y1=[]
             N=len(X)-1
             for k in range(0,N//2):
                 X1.append(X[2*k])
                 Y1.append(Y[2*k])
             L1=["rectangle",rectangle(X1,Y1)]
             L2=["trapeze",trapeze(X1,Y1)]
             L3=["point_milieu",pointmilieu(X,Y)]
             L4=["simpson",simpson(X,Y)]
             #L5=["romb",romb(Y,dx=(X[N]-X[0])/N)]
             L=[L1,L2,L3,L4]#,L5]
             #for k in range(0,5):
             #    print(L[k])
             return [L1,L2,L3,L4]#L5]
```

Par exemple :

```
In [42]: X=linspace(0,pi/2,51)
         Y=[cos(x) for x in X]
         approx(X,Y)
```

```
Out[42]: [['rectangle', [1.027141673376661, 0.968255065002232 68]],
          ['trapeze', 0.99769836918944699],
          ['point_milieu', 1.000164512349313],
          ['simpson', 1.0000000054122518]]
```

On crée alors une fonction, prenant en argument n qui correspond au nombre d'itérations choisies par notre méthode (qui est en fait $2n$) et qui calcule les erreurs correspondantes correspondantes à chaque méthode :

```
In [43]: def erreur(n):
             X=linspace(0,pi/2,2*n+1)
             Y=[cos(x) for x in X]
             return [abs(1-approx(X,Y)[0][1][0]),abs(1-approx(X,Y)[1][1]),abs(1-app
```

Par exemple :

```
In [44]: erreur(30)
```

```
Out[44]: [0.023211161014461634,
          0.0015986256350245531,
          0.0001142406672705043,
          2.6099817862501595e-09]
```

On crée alors des fonctions qui retournent séparément les différentes erreurs de chaque méthode :

```
In [45]: def erreur_rectangle(n):
             return erreur(n)[0]

         def erreur_trapeze(n):
             return erreur(n)[1]

         def erreur_point_milieu(n):
             return erreur(n)[2]

         def erreur_simpson(n):
             return erreur(n)[3]
```

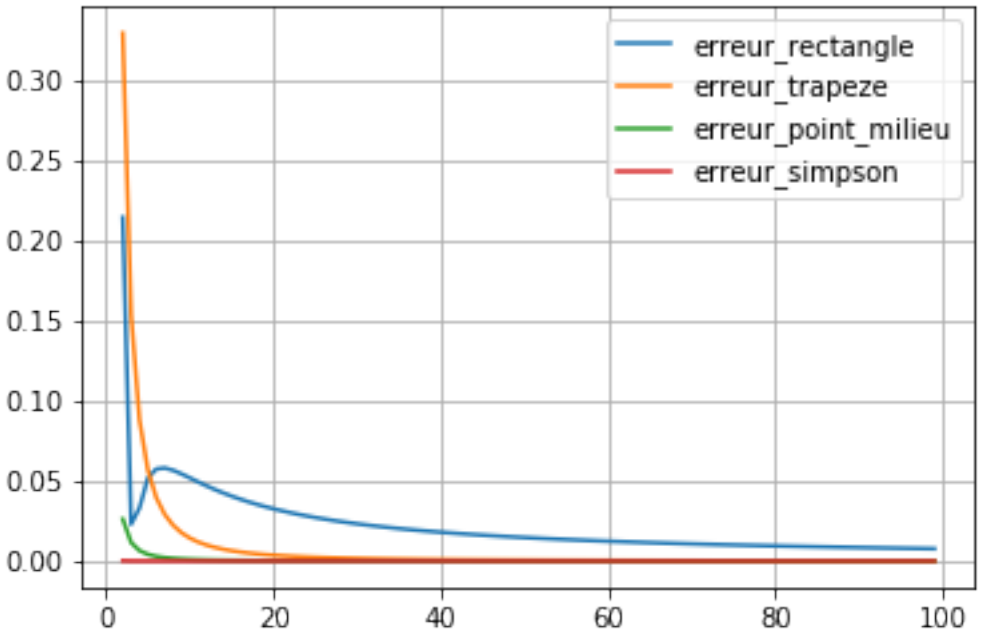
Il ne nous reste plus qu'à réaliser les tracés correspondants :

```
In [46]: T=[]
         Y1=[]
         Y2=[]
         Y3=[]
         Y4=[]

         for k in range(2,100):
             T.append(k)
             Y1.append(erreur_rectangle(k))
             Y2.append(erreur_trapeze(k))
             Y3.append(erreur_point_milieu(k))
             Y4.append(erreur_simpson(k))

         plot(T,Y1,label="erreur_rectangle")
         plot(T,Y2,label="erreur_trapeze")
         plot(T,Y3,label="erreur_point_milieu")
         plot(T,Y4,label="erreur_simpson")
         grid()
         legend()
```

```
Out[46]: <matplotlib.legend.Legend at 0x7f221c02c748>
```



Q5)

```
In [75]: help(romb)

Help on function romb in module scipy.integrate.quadrature:

romb(y, dx=1.0, axis=-1, show=False)
    Romberg integration using samples of a function.

    Parameters
    ----------
    y : array_like
        A vector of ``2**k + 1`` equally-spaced samples of a function.
    dx : float, optional
        The sample spacing. Default is 1.
    axis : int, optional
        The axis along which to integrate. Default is -1 (last axis).
    show : bool, optional
        When `y` is a single 1-D array, then if this argument is True
        print the table showing Richardson extrapolation from the
        samples. Default is False.

    Returns
    -------
    romb : ndarray
```

```
        The integrated result for `axis`.

    See also
    --------
    quad : adaptive quadrature using QUADPACK
    romberg : adaptive Romberg quadrature
    quadrature : adaptive Gaussian quadrature
    fixed_quad : fixed-order Gaussian quadrature
    dblquad : double integrals
    tplquad : triple integrals
    simps : integrators for sampled data
    cumtrapz : cumulative integration for sampled data
    ode : ODE integrators
    odeint : ODE integrators
```

```
In [83]: X=linspace(0,pi/2,2**(4)+1)
         Y=f1(X)
         print(romb(Y,dx=(X[1]-X[0])))
         print(abs(1-romb(Y,dx=(X[1]-X[0]))))
```

```
0.999999999998
1.98285832198e-12
```

On compare avec simpson qui semblait être la plus précise :

```
In [85]: X=linspace(0,pi/2,2**(4)+1)
         Y=f1(X)
         print(simpson(X,Y))
         print(abs(1-simpson(X,Y)))
```

```
1.00000051668
5.16684706353e-07
```

On constate donc sur cet exemple que la méthode de Romberg embarquée dans Python est plus précise que les quatre méthodes proposées dans le TP.

Activité 2

On commence par ouvrir puis lire le fichier sous python :

```
In [60]: fichier_import=open("TP_indexa.txt","r")
         L=fichier_import.read()
```

On peut regarder ce que renvoie L :

```
In [62]: L
```

```
Out[62]: '0.00\t-0.0120427358582045\n0.04\t0.00325665387564035\n0.08\t-0.006340728
```

Il s'agit d'une chaîne de caractères. On veut maintenant extraire les valeurs des temps ainsi que les vitesses mesurées à ces instants :

```
In [63]: L2=L.split('\n')
         L3=[]
         for element in L2:
                 L3.append(element.split('\t'))

         Temps=[float(element[0]) for element in L3]
         Mesure =[float(element[1]) for element in L3]

In [64]: L2

Out[64]: ['0.00\t-0.0120427358582045',
          '0.04\t0.00325665387564035',
          '0.08\t-0.00634072822320703',
          '0.12\t-0.00987054117817628',
          '0.16\t0.0210998962902619',
          '0.20\t0.00186918561278802',
          '0.24\t-0.00790792896917133',
          '0.28\t0.0138245913833571',
          '0.32\t0.0322789242399762',
          '0.36\t0.0545146955450237',
          '0.4\t0.0725736169389262',
          '0.44\t0.0734003863611763',
          '0.48\t0.100769117898697',
          '0.52\t0.0958518366461526',
          '0.56\t0.121940939392938',
          '0.6\t0.160776270837669',
          '0.64\t0.220840871766303',
          '0.68\t0.250510178914494',
          '0.72\t0.3122857666265',
          '0.76\t0.337742277418167',
          '0.8\t0.382321659517417',
          '0.84\t0.436462477272077',
          '0.88\t0.457202940756319',
          '0.92\t0.46526906867417',
          '0.96\t0.456268331842971',
          '1.00\t0.384068592728159',
          '1.04\t0.306088856532107',
          '1.08\t0.296117564468246',
          '1.12\t0.256433962160505',
          '1.16\t0.239151418603199',
          '1.2\t0.191293848971697',
          '1.24\t0.163874725849171',
          '1.28\t0.106599506636657',
          '1.32\t0.110618125456604',
          '1.36\t0.065693723495062',
```

```
           '1.4\t0.0698201818051356',
           '1.44\t0.0186484782336404',
           '1.48\t0.00401153030458099',
           '1.52\t0.00572251703684048',
           '1.56\t0.00550683805723576',
           '1.60\t-0.00790792896949524',
           '1.64\t0.00943206249228324',
           '1.68\t0.00565062404385483',
           '1.72\t-0.00416243701755972',
           '1.76\t-0.000410848557042695']
```

In [65]: L3

Out[65]:
```
[['0.00', '-0.0120427358582045'],
 ['0.04', '0.00325665387564035'],
 ['0.08', '-0.00634072822320703'],
 ['0.12', '-0.00987054117817628'],
 ['0.16', '0.0210998962902619'],
 ['0.20', '0.00186918561278802'],
 ['0.24', '-0.00790792896917133'],
 ['0.28', '0.0138245913833571'],
 ['0.32', '0.0322789242399762'],
 ['0.36', '0.0545146955450237'],
 ['0.4', '0.0725736169389262'],
 ['0.44', '0.0734003863611763'],
 ['0.48', '0.100769117898697'],
 ['0.52', '0.0958518366461526'],
 ['0.56', '0.121940939392938'],
 ['0.6', '0.160776270837669'],
 ['0.64', '0.220840871766303'],
 ['0.68', '0.250510178914494'],
 ['0.72', '0.3122857666265'],
 ['0.76', '0.337742277418167'],
 ['0.8', '0.382321659517417'],
 ['0.84', '0.436462477272077'],
 ['0.88', '0.457202940756319'],
 ['0.92', '0.46526906867417'],
 ['0.96', '0.456268331842971'],
 ['1.00', '0.384068592728159'],
 ['1.04', '0.306088856532107'],
 ['1.08', '0.296117564468246'],
 ['1.12', '0.256433962160505'],
 ['1.16', '0.239151418603199'],
 ['1.2', '0.191293848971697'],
 ['1.24', '0.163874725849171'],
 ['1.28', '0.106599506636657'],
 ['1.32', '0.110618125456604'],
 ['1.36', '0.065693723495062'],
```

```
          ['1.4', '0.0698201818051356'],
          ['1.44', '0.0186484782336404'],
          ['1.48', '0.00401153030458099'],
          ['1.52', '0.00572251703684048'],
          ['1.56', '0.00550683805723576'],
          ['1.60', '-0.00790792896949524'],
          ['1.64', '0.00943206249228324'],
          ['1.68', '0.00565062404385483'],
          ['1.72', '-0.00416243701755972'],
          ['1.76', '-0.000410848557042695']]

In [66]: Temps

Out[66]: [0.0,
          0.04,
          0.08,
          0.12,
          0.16,
          0.2,
          0.24,
          0.28,
          0.32,
          0.36,
          0.4,
          0.44,
          0.48,
          0.52,
          0.56,
          0.6,
          0.64,
          0.68,
          0.72,
          0.76,
          0.8,
          0.84,
          0.88,
          0.92,
          0.96,
          1.0,
          1.04,
          1.08,
          1.12,
          1.16,
          1.2,
          1.24,
          1.28,
          1.32,
          1.36,
```

```
           1.4,
           1.44,
           1.48,
           1.52,
           1.56,
           1.6,
           1.64,
           1.68,
           1.72,
           1.76]

In [67]: Mesure

Out[67]: [-0.0120427358582045,
          0.00325665387564035,
          -0.00634072822320703,
          -0.00987054117817628,
          0.0210998962902619,
          0.00186918561278802,
          -0.00790792896917133,
          0.0138245913833571,
          0.0322789242399762,
          0.0545146955450237,
          0.0725736169389262,
          0.0734003863611763,
          0.100769117898697,
          0.0958518366461526,
          0.121940939392938,
          0.160776270837669,
          0.220840871766303,
          0.250510178914494,
          0.3122857666265,
          0.337742277418167,
          0.382321659517417,
          0.436462477272077,
          0.457202940756319,
          0.46526906867417,
          0.456268331842971,
          0.384068592728159,
          0.306088856532107,
          0.296117564468246,
          0.256433962160505,
          0.239151418603199,
          0.191293848971697,
          0.163874725849171,
          0.106599506636657,
          0.110618125456604,
          0.065693723495062,
```

```
          0.0698201818051356,
          0.0186484782336404,
          0.00401153030458099,
          0.00572251703684048,
          0.00550683805723576,
          -0.00790792896949524,
          0.00943206249228324,
          0.00565062404385483,
          -0.00416243701755972,
          -0.000410848557042695]
```

Et voilà ! Pour obtenir la variation angulaire correspondante sur cet intervalle de temps, il nous faut en fait calculer : $\int_0^{1,76} \dot{\theta}(t)\, dt$. On obtient alors des approximations de cette intégrale à l'aide des valeurs numériques précédentes :

```
In [72]: print(rectangle(Temps,Mesure))
         print(trapeze(Temps,Mesure))
         print(pointmilieu(Temps,Mesure))
         print(simpson(Temps,Mesure))

[0.2504623977788076, 0.250927673270854]
0.25069503552483074
0.2529636547287675
0.251451241926143
```

Bref, la variation angulaire est de 0.25 radians, soit :

```
In [73]: 0.25*180/pi

Out[73]: 14.32394487827058
```

degrés.