TP2
Intégration numérique

26 octobre 2017

Réalisé par RAHARIJAONA Dylan et KUCAM Delphine

Encadré par Jean-Paul Cardinal

Le but de ce tp de calculer, approximativement, l'intégrale d'une fonction sur un intervalle compact (borné, fermé)

1. Création d'une fonction simple et représentation graphique
   (a) Pour commencer, nous avons écris la fonction $f(x) = \sqrt{1-x^2}$ en python. Cela fait maintenant 2 TP que l'on apprend à coder des fonctions en python, c'est donc sans difficultés que nous l'avons fait pour f(x).

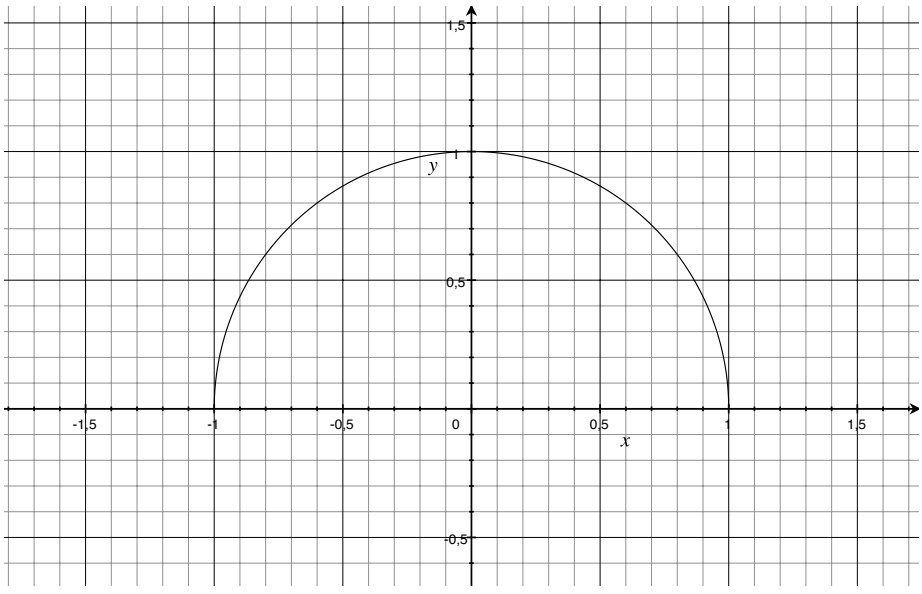
   Voici le code utilisé :

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

   (b) Suite à cela, nous avons représenté le graphique de la fonction f sur l'intervalle $[-0.5, 0.5]$ grâce au module matplotlib.plot et numpy que nous avons vu dans les précedents TPs.

   (c) Pour cette question, nous devons calculer, à la main, l'intégrale I de f sur l'intervalle donné. Nous l'avons fait sur papier que nous vous avons rendu.
   Nous avons trouvé I = $\pi/6 + \sqrt{3}/4$ Soit environ 0.9566

2. Calcul approché de I au moyen de la méthode du point milieu
   (a) Voici le graphique de la fonction f :



(b) On définit sur python une subdivision régulière de l'intervalle d'intégration [0,1]. On a donc : x0 = -0.5 ; x1 = -0.25 ; x2 = 0 ; x3 = 0.25 ; x4 = 0.5

Sur le graphique que nous vous avons remis à la question 2), nous avons dessiné les rectangles de bases $[X_{k-1}; X_k]$ et de hauteur f(c_k).

Puis nous avons calculé leur surface à l'aide de python et nous avons obtenu :
Sk = 0.95959
La méthode du point milieu consiste donc à calculer la somme des surfaces comme approximation de I.

(c) On cherche à trouver l'erreur commise soit la différence entre la somme des surfaces et l'intégrale de f pour tester la précision de la méthode du point milieu. À l'aide de python, on a l'erreur commise qui équivaut à environ 0.00298

(d) Pour cette question, nous devons calculer le temps des fonctions à l'aide de clock() du module time que l'on place sous chaques fonctions et on calcule ensuite la différence de temps entre elles.

(e) Par la suite, on a écrit une fonction sur python qui nous permet de trouver l'intégrale approchée de f sur $[a, b]$ au moyen de la méthode du point milieu. Nous nous sommes aidés des questions précédentes.

(f) Nous avons testé la fonction point_milieu avec $f(x) = \sqrt{1 - x^2}$, $a = -0.5, b = 0.5$ et $n = 10^k$ et k variant de 1 à 6.

(g) Nous avons le résultat de la question (f) sous forme de tableau :

n	erreur	temps
10	4,804*e-4	1.200e-05
100	4.811e-06	5.900e-05
1000	4.811e-08	6.710e-04
10000	4.811e-10	9.987e-03
100000	4.757e-12	5.542e-02
1000000	7.649e-13	3.524e-01

Pour cette question 2), nous avons rencontré plusieurs difficultés. En effet, notre façon d'écrire notre code était compliqué à comprendre et desfois nous nous perdions nous même devant notre code. À l'aide de notre professeur, nous avions simplifié notre code afin de l'éclairer,nous l'avons structuré et nous avons enlevé les choses inutiles.

3. la méthode du trapèze

Cette méthode consiste à approximer f sur l'intervalle $[x_{k-1}, x_k]$ par la fonction affine qui prend les mêmes valeurs que f en x_{k-1} et x_k.

On a repris le travail effectué à la question 2) pour la méthode du trapèze et on obtient :

n	erreur	temps
10	2.053e-02	1.800e-05
100	4.245e-02	6.200e-05
1000	4.500e-02	5.600e-04
10000	4.526e-02	5.435e-03
100000	4.529e-02	5.456e-02
1000000	4.529e-02	5.383e-01

Cependant, nous avons rencontré un problème avec cette méthode. L'erreur commise (dans le tableau) n'est pas celle escompté.
Quand on applique la fonction trapèze aux valeurs utilisés pour le point milieu, la fonction trapèze fonctionne (dans le code de la fonction trapeze dans module "integration" ligne 33-34), mais ne fonctionne pas pour les valeurs choisit dans la fonction trapèze.

4. méthode de Simpson

La méthode de Simpson consiste à approximer f sur l'intervalle $[x_{k-1}, x_k]$ par le polynôme de degré 2 qui prend les mêmes valeurs que f en x_{k-1}, c_k et x_k.

On a repris le travail effectué à la question 2) pour la méthode de Simpson et on obtient :

n	erreur	temps
10	2.116e-07	2.200e-05
100	2.138e-11	1.270e-04
1000	1.332e-15	1.262e-03
10000	6.761e-14	1.981e-02
100000	6.276e-13	1.521e-01
1000000	1.121e-11	8.550e-01

5. Comparaison des méthodes

Après avoir testé les 3 méthodes, nous avons conclus que la méthode Simpson était la plus efficace suivit de la méthode des trapèzes et enfin de la méthode du point milieu.
En effet ceci s'explique par le fait que le nombre de valeurs pris pour approximer la fonction diffère pour chacune des méthodes :
-Pour la méthode du Point Milieu, on approxime f par une fonction constante (de degré 0), on ne prend donc que 1 seul point ($f(c_k)$) pour approximer f.
-Pour la méthode des trapèzes, on approxime f par une fonction affine (de degré 1), on prend alors 2

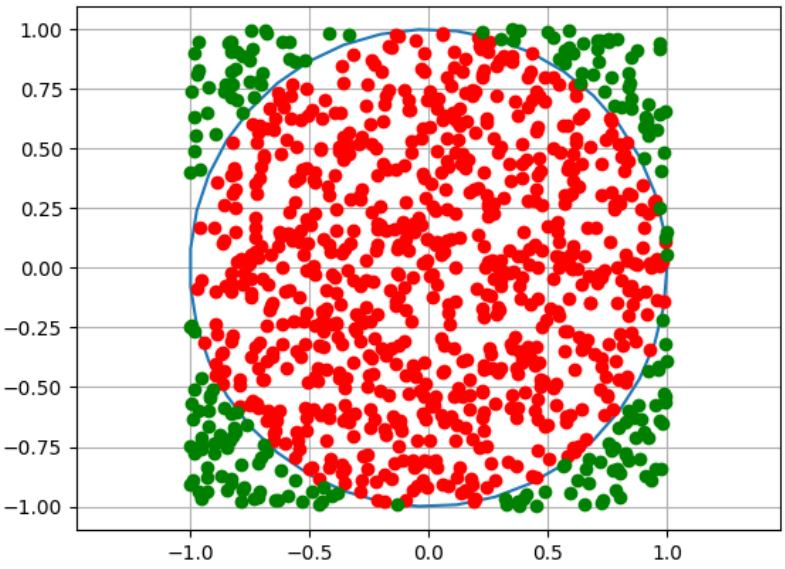
points ($f(x_{k-1})$) et $f(x_k)$) pour approximer f.
-Finalement, pour la méthode Simpson, on approxime f par une fonction du 2nd degré, on prend alors 3 points($f(x_{k-1})$, $f(x_k)$ et $f(c_k)$) pour approximer f. La précision est donc plus élevé pour la méthode Simpson.
De plus, le temps mis pour calculer cette approximation ne diffère pas beaucoup pour chacune des fonctions.
Ainsi, il est mieux d'utiliser la méthode Simpson

6. **méthode de Monte-Carlo**

   (a) Nous devions dessiner le cercle unité à l'aide de matplotlib.pyplot. Ceci étant nouveau pour nous, nous sommes allés regarder sur Internet.
   La surface du disque unité est donné par : S = π*(1**2) = π .

(b) A l'aide de la fonction **numpy.random.rand()**, on a généré 7 valeurs aléatoires suivant une distribution de loi uniforme sur [0,1] et [-1 ;1]. On a obtenu les valeurs suivantes :
-pour [0,1] : [0.95031147, 0.57742382, 0.58642541, 0.72867086, 0.90369628, 0.35445451, 0.6154403]
-pour [-1,1] : [0.5395885 , 0.79119839, -0.51028129, 0.30999009, 0.80638663, 0.06277604, 0.14590814]

(c) On a généré 1000 points suivant une distribution uniforme sur [-1,1]x[-1,1]. On a ensuite inséré, parmi ces 1000 points, les points intérieurs au disque unité en rouge et les points extérieurs en vert. Voici le graphique obtenu :



(d) Il y a environ 789 points rouges et 211 points verts. NOus avons calculé l'approximation de la surface S du disque avec le rapport $4 * I/N$ avec I le nombre de points intérieur (rouge) ainsi nous avons calculé l'erreur commise $abs(4 * I/N - S)$. Et à l'aide de **time.clock()** nous avons mesuré le temps de calcul de cette approximation.
L'erreur commise vaut : 3.359e-02
Le temps associé au calcul vaut : 1.317e-03 secondes.

(e) La fonction Monte Carlo est intégré dans le module que nous avons créé, qui se nomme "integration.py".

(f) Nous avons testé la fonction Monte Carlo avec N=10**k pour k allant de 1 à 6, voici le résultat obtenu :

n	erreur	temps
10	4.584e-01	5.400e-05
100	2.216e-01	2.400e-04
1000	4.641e-02	2.337e-03
10000	7.193e-03	2.014e-02
100000	3.113e-03	1.492e-01
1000000	6.353e-04	1.417e+00

(g) Nous avons repris le travail précédent avec le calcul du volume de la boule unité par la méthode de Monte Carlo et nous avons obtenu :

n	erreur	temps
10	1.942e+00	4.400e-05
100	8.159e-02	3,490e-04
1000	9.841e-02	3,759e-03
10000	5.993e-03	3,181e-02
100000	4.867e-03	2,197e-01
1000000	3.649e-03	2,212e+00