Cube de points

Année 2021 – 2022

CALONE Théo, DE NIJS William, DEFAUD Thomas, GARNAULT Chléa, LECHENET Marin, MANSIAT Alexia, PETIT Alice, PEREIRA Louna, élèves de Terminale
ANTHOINE Pierre, CORNU-RICHARD Louis et ROURA Valerian, élèves de Seconde

Établissements : Lycée SAINT-ETIENNE et Lycée Catherine et Raymond JANOT

Enseignants : SASSIAT Marie-Noëlle, CAVALIER Christophe

Chercheur : MASSUYEAU Gwénaël, Université de Bourgogne, Institut de Mathématiques de Bourgogne.

1. Présentation du sujet

Le sujet que nous avons choisi s'intitule « Cube de points ». Il s'articule autour des thèmes de l'aléatoire, de la géométrie dans l'espace et de la modélisation en 3D. Dans un premier temps, nous devions réaliser une fonction permettant de nous donner une suite de nombres ayant l'air aléatoires. Puis en utilisant cette même fonction nous devions modéliser un cube de points aléatoires en 3D situé sur un repère orthonormé allant de -1 à 1. Nous avions ensuite comme défi de modéliser les lettres X, Y et Z en négatif dans les axes du repère correspondant.

2. Méthode

2.1. Comment créer de l'aléatoire ?

Dans un premier temps, nous devions réussir à modéliser une suite de nombres ayant l'apparence d'avoir été choisis au hasard. Pour réaliser ceci, nous avons pris la décision de réaliser un programme en langage Python. Pour permettre à ce programme de retourner des nombres ayant une apparence aléatoire nous avons choisi d'utiliser une variable se basant sur le temps, la variable « time.time ». Ensuite, afin d'augmenter les écarts entre nos valeurs nous

avons mis au cube la succession de calcul infligée à cette valeur [1]. Cela permet qu'à chaque fois que nous lançons le programme, une suite différente sans aucun rapport apparent sorte, et cela même pour un écart de quelques secondes. Ensuite chaque valeur est définie à partir de la précédente grâce à un calcul complexe qui donne l'illusion d'une suite aléatoire. En effet ce calcul est très sensible à de minimes variations de valeur, ce qui empêche de pouvoir prédire facilement une valeur en fonction de la précédente sans connaitre le calcul [2]. De plus ce calcul n'admet pas de valeurs interdites qui pourrait empêcher de générer de nouvelles données à partir d'une certaine valeur.

```
ms = (time.time())**3
n = ms
def s(e):
    e=((-1)**int(e))*(pi*sqrt(3)*(sin(153+e)-5*cos(0.1*e-963)))**3
    return e
```

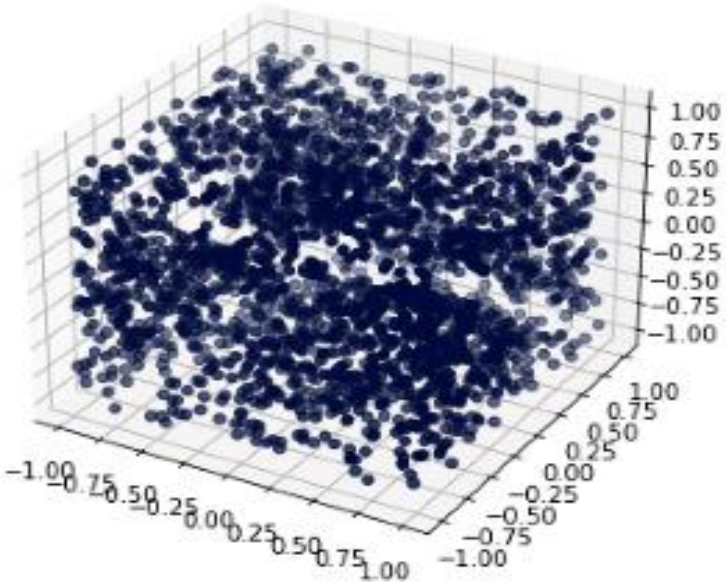
En vert, on initialise la suite grâce à la fonction time.time. En bleu, on nomme la fonction pour pouvoir l'utilisez plus tard. En jaune on définit par récurrence la suite.

2.2. Comment générer le cube ?

L'objectif après avoir trouvé une méthode générant des nombres aléatoires était de l'utiliser pour générer un nuage de points aléatoires dans un cube centré en l'origine et d'arête de longueur 2. Le cube devait être constitué d'un très grand nombre N de points aléatoires. Chaque point correspond donc à 3 coordonnées dont les valeurs sont comprises entre -1 et 1. La première étape a donc été de trouver un moyen de contraindre notre programme à donner des valeurs comprises dans cet intervalle. Pour cela nous avons appliqué la fonction cosinus aux valeurs aléatoires ainsi obtenus ce qui nous a permis de garantir leur appartenance au cube. De plus, ce cube devait pouvoir être modélisé sur un ordinateur de manière à ce que celui-ci puisse être déplacé dans toutes les directions. C'est pour cette raison que nous avons utilisé le langage python plus précisément la bibliothèque matplotlib et le logiciel Spyder. Nous avons ensuite dû écrire le programme en faisant en sorte d'utiliser notre fonction aléatoire afin de définir les coordonnées des points.

```
u = int(input("nombre de points")); x=0; y=0; z=0; p=0 ; xp = 0; yp=0; zp=0
w = []; i = []; l = []
while u>p:
        xp = s(n); x = cos(xp); yp= s(xp); y = cos(yp); zp = s(yp); z = cos(zp); n = zp
```

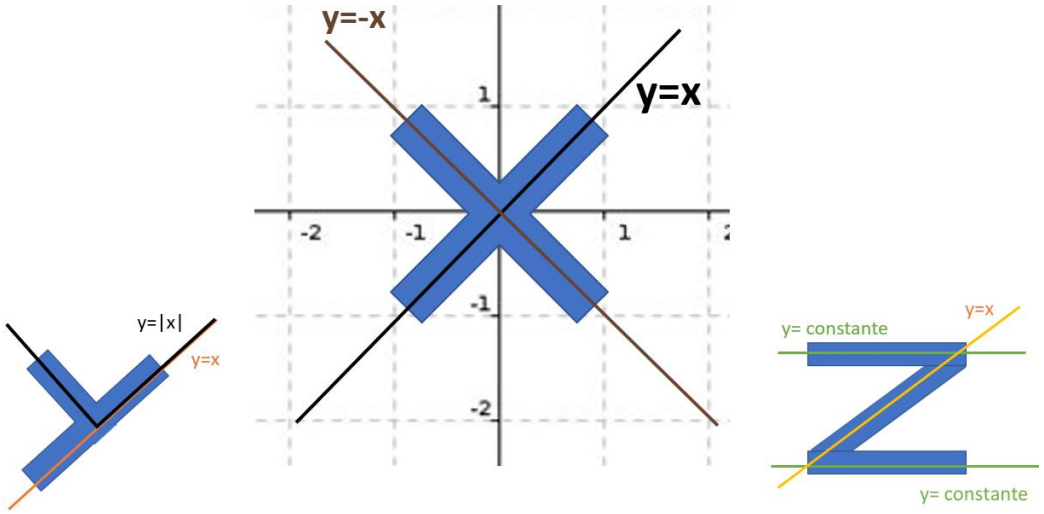
[3]



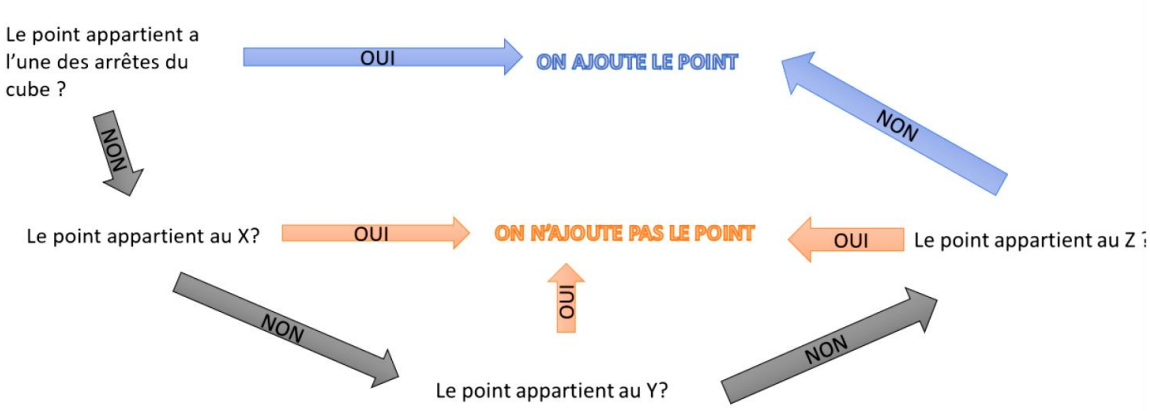
En bleu on définit u le nombre de points demandé, puis on initialise les valeurs des coordonnées, des valeurs aléatoires intermédiaires et on nomme et initialise p le nombre de points placé dans le cube. On définit ensuite des listes en vert qui seront utiles pour placer les points. Pour s'assurer d'avoir le nombre de points voulu, on définit une boucle qui s'arête quand le nombre de points voulu est identique au nombre de points réel. Cette boucle sera complétée par la suite. Puis, on utilise notre fonction s définit précédemment pour générer les valeurs aléatoires intermédiaire l'une en fonction de l'autre. Enfin, pour obtenir les coordonnées réelles on utilise la fonction cosinus.

2.3. Comment modéliser mathématiquement les lettres X, Y et Z ?

Une fois notre cube de point obtenu, nous devions faire apparaitre en négatif la lettre X lorsque l'on se place dans l'axe des X et faire de même avec la lettre Y dans l'axe des Y et la lettre Z dans l'axe des Z. Pour faire apparaître ces lettres nous devions donc supprimer les points du cube nous permettant de former ces lettres. Pour déterminer les points à supprimer, nous avons décidé d'étudier la forme des lettres que nous devions former. C'est ainsi, que nous avons remarqué que chacune de ces lettres pouvait être modélisée par des fonctions simples. La lettre X peut être modélisée par deux droites d'équation : Y=X et Y=-X ; la lettre Y par : Y=X et Y=|X|; et la lettre Z par : Y= a, Y=b et Y=X avec a et b des constantes.



Ainsi tous les points appartenant aux images de ces fonctions et n'appartenant pas aux arêtes du cube ne seront pas conservés dans notre cube de points. Les points se situant sur les arêtes du cube n'interfèrent pas dans la création de nos lettres nous les conservons donc automatiquement.
Afin de réaliser la sélection des points à conserver ou non dans notre cube, nous avons ajouté des conditions à notre programme initial. Ces conditions peuvent être expliquées en langage courant par une suite de questions.



Un point peut donc être ajouté dans deux cas de figures, soit il appartient aux arêtes du cube ou bien il n'appartient pas aux droites formant les lettres X, Y et Z.
Ainsi dans le langage Python ces différentes réflexions se traduisent de la manière suivante :
(Pour rappel cette partie du programme fait toujours partie de la boucle précédente.) [4]

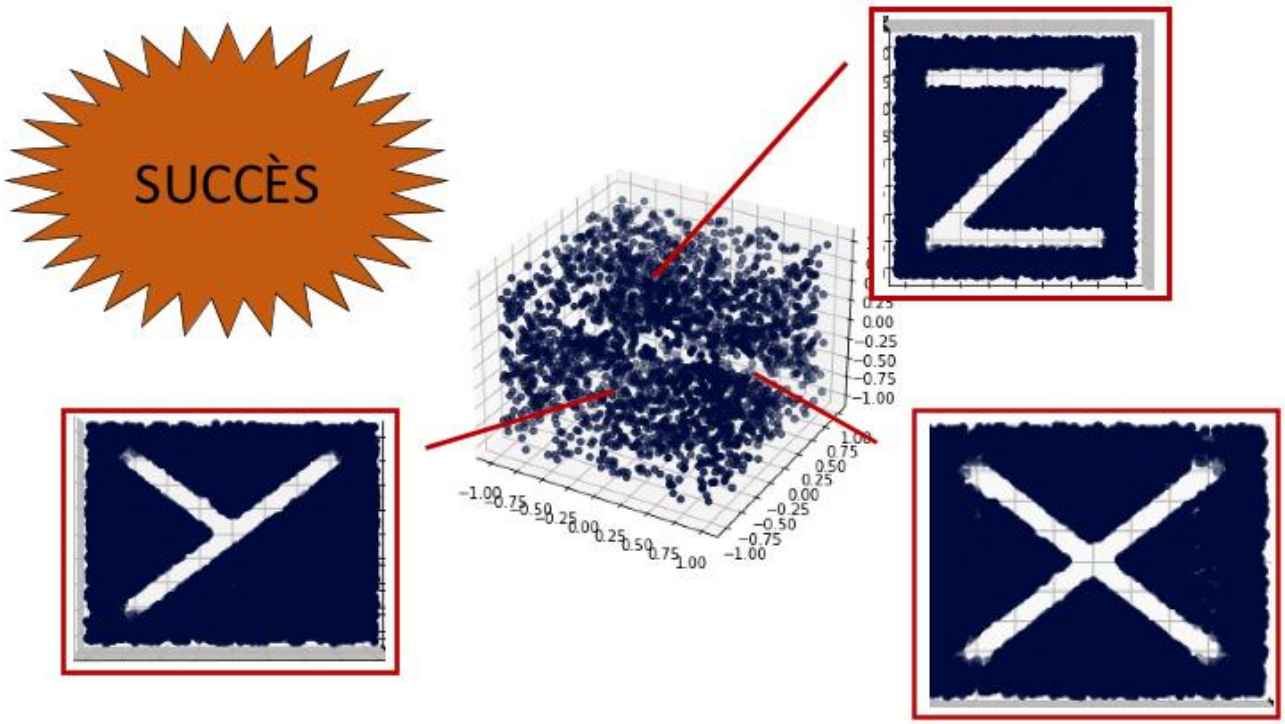
```
if ((x <= -0.8 or x >= 0.8) and (y <= -0.8 or y >= 0.8)) or((x <= -0.8 or x >= 0.8) and(z <=
-0.8 or z >= 0.8)) or ((z <= -0.8 or z >= 0.8) and (y <= -0.8 or y >= 0.8)):
        w.append(x)
        i.append(y)
        l.append(z)
        p=p+1
    elif y >= z-0.15 and y<= z+0.15 or y >= (-z)-0.15 and y<= (-z)+0.15:
            print (p)
    elif z  >= x-0.15 and z<= x+0.15 or z>= abs(x)-0.15 and z<= abs(x)+0.15:
        print (p)
    elif ((x >= y-0.15 and x<= y+0.15)and(y >= -0.85 and y <= 0.85)and(x >= -0.80 and x
<= 0.80)) or ((x>-0.80 and x<0.80)and (y>0.58 and y<0.80)) or ((x>-0.80 and x<0.80)and
(y>-0.80 and y<-0.58)):

        print (p)
    else:
        w.append(x)
```

En jaune, le programme ajoute le point. La variable p compte le nombre de points actuellement ajouté dans le cube ce qui permet au programme de s'arrêter quand le nombre de points demandé est atteint. La première ligne définit les bordures de chaque face du cube. Si le point appartient à cette bordure, il est ajouté. Ensuite chaque « elif » correspond à une lettre. S'il n'appartient à aucune alors le point est ajoute et le programme recommence, s'il appartient à l'une des lettres alors le point n'est pas ajouté et le programme recommence.

3. Conclusion

Après l'exécution du programme précédent voici ce que nous obtenons :



Nous obtenons donc un cube de points qui nous semble banal lorsqu'on ne se place pas en face de l'un des axes mais lorsque l'on se place en face des axes X, Y et Z la lettre correspondant à l'axe apparait en négatif. En effet l'absence de points à certains endroits nous permet d'apercevoir ces lettres.
Pour arriver à ce résultat nous avons choisi de décomplexifier au maximum le problème qui nous était posé avant de se lancer dans l'écriture du programme, ce qui nous a donné l'opportunité de penser à utiliser une approche fonctionnelle et qui nous a également permis d'anticiper les difficultés que nous pourrions rencontrer et de les prendre en compte dans notre réflexion afin d'éviter de devoir recommencer notre travail. Cette décomplexification du problème nous a donc permis de réaliser la tâche demandée à l'aide d'un programme court, lisible et aisément compréhensible.

SCRIPT PYTHON

```
from random import*
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import time
import math
from math import*
#On importe toutes les bibliothèques

ms = time.time() #On définit un nombre initiale qui dépend du temps afin qu'il soit différent à chaque fois que le programme est lancé
ms = ms**3 #On le met au cube afin de le rendre plus aléatoire

n = ms
def s(p):
    p=((-1)**int(p))*(pi*sqrt(3)*(sin(153+p)-5*cos(0.1*p-963)))**3
    return p
#On définit une fonction qui rend notre nombre aléatoire sur R

x = 0
y=0
z=0
xp = 0
yp=0
zp=0
#On crée des variables x, y et z et xp, yp et zp
w = []
i = []
l = []
#On crée les listes w, i et l des listes qui permettrons de stocker toutes les coordonnées
for k in range (5000):
    xp = s(n)
    x = cos(xp)
    yp= s(xp)
    y = cos(yp)
    zp = s(yp)
    z = cos(zp)
    n = zp
# On définit un x un y et un z aléatoire compris entre -1 et 1
    if ((x <= -0.8 or x >= 0.8) and (y <= -0.8 or y >= 0.8)) or((x <= -0.8 or x >= 0.8) and(z <= -0.8 or z >= 0.8)) or ((z <= -0.8 or z >= 0.8) and (y <= -0.8 or y >= 0.8)):
        w.append(x)
        i.append(y)
        l.append(z)
#Si le point appartient à l'une des arêtes du cube, on ajoute le point aux listes (w qui correspond aux valeurs x, i qui correspond aux valeurs y et l qui correspond aux valeurs z)
    else:
        if y >= z-0.15 and y<= z+0.15 or y >= (-z)-0.15 and y<= (-z)+0.15:
            print (k)
#Si le point n'appartient pas à l'une des arêtes du cube, on cherche à savoir s'il appartient au x creusé qui est défini par la fonction f(z)=y et f(z)=-y on ne le valide pas
```

```
    else:
        if z  >= x-0.15 and z<= x+0.15 or z>= abs(x)-0.15 and z<= abs(x)+0.15:
#Si le point n'appartient ni aux arêtes du cube ni au X creusé, on cherche à savoir s'il appartient au Y creusé qui est défini par la fonction f(x)=z et f(x)=|z|
            print (k)
        else:
            if ((x >= y-0.15 and x<= y+0.15)and(y >= -0.85 and y <= 0.85)and(x >= -0.80 and x <= 0.80)) or ((x>-0.80 and x<0.80)and (y>0.58 and y<0.80)) or ((x>-0.80 and x<0.80)and (y>-0.80 and y<-0.58)):
#Si le point n'appartient ni à l'une des arêtes du cube, ni au X creusé ni au Y on cherche à savoir si il appartient au Z creusé qui est défini par la fonction f(x)=y, f(x)=0.80 et f(x)=-0.80
                print (k)
            else:
                w.append(x)
                i.append(y)
                l.append(z)
#Si le point n'appartient ni à l'une des arêtes du cube, ni au X creusé ni au Y, ni au Z creusé alors on valide le point en on ajoute ses coordonnées aux listes w, i et l
fig = plt.figure(1)
ax = Axes3D(fig)
ax.scatter(w,i,l,
       color=['#000C3B'])
#On définit le graphique en 3d
plt.show()
#On crée le graphique
print (w)
#by Ouiliam, Toma, Mar1, Loupna, TO et Aleqca
```

NOTES DE L'ÉDITION

[1] Il n'est pas toujours vrai que l'élévation au cube augmente les différences entre deux nombres. On a $|a^3 - b^3| \leq |a - b|$ pour tous les nombres réels a et b tels que $|a^2 + ab + b^2| \leq 1$. C'est en particulier le cas si a et b sont tous deux inférieurs ou égaux à $\frac{1}{\sqrt{3}}$.

[2] En général, cette difficulté à prévoir le résultat du calcul ne suffit pas pour conclure qu'il engendre une suite de nombres pseudo-aléatoires. Il faut aussi que les termes successifs soient aussi indépendants que possible, et que les valeurs de la suite soient uniformément distribuées dans leur intervalle de valeurs. Cela nécessite des techniques mathématiques qui dépassent certainement les programmes de l'enseignement secondaire. Néanmoins, il faut être conscient du problème.
Il faut également noter que Python comporte un générateur de nombres aléatoires. Il aurait pu être utilisé dans ce travail.

[3] Dans cette définition, la fonction cosinus est utilisée pour transformer la suite s(n) en une suite de nombres dans l'intervalle [-1,1]. Mais cela pose problème avec l'uniformité de distribution mentionnée dans la note précédente. Si la suite s(n) est uniformément distribuée sur son intervalle de valeurs, cela n'est plus vrai pour la suite cos(s(n)). En gros, les voisinages de 1 ou de -1 contiendront "plus" de valeurs que les voisinages de 0, par exemple. Sachant que les valeurs de se répartissent dans l'intervalle $[(-6\pi\sqrt{3})^3, (6\pi\sqrt{3})^3]$, on aurait pu prendre $\frac{\text{xp}}{(6\pi\sqrt{3})^3}$ comme valeur de x, et de même pour y et z.

[4] En toute rigueur, il n'est pas correct de dire que les points (x,y,z) vérifiant les premières conditions ci-dessous appartiennent à une arête du cube, qui est un segment. Ces points sont répartis aléatoirement dans des parallélépipèdes rectangles dont une arête coïncide avec l'une des arêtes du cube et la longueur des arêtes orthogonales à celle-ci est égale à 0,2. Bien sûr, cette approximation est justifiée, mais il aurait fallu préciser que l'on considère des arêtes "épaissies".

[1] It is not always true that the elevation to the cube increases the differences between two numbers. We have $|a^3 - b^3| \leq |a - b|$ for all real numbers a and b that verify the inequality $|a^2 + ab + b^2| \leq 1$. This holds in particular if both a and b are less or equal to $\frac{1}{\sqrt{3}}$.

[2] In general, these properties of the proposed calculation are not enough to conclude that the definition generates a pseudo-random sequence of numbers. We must also have that successive terms be as independent as possible and that the values of the sequence are uniformly distributed over their range. This requires mathematical techniques that certainly go beyond the high school programs. Nevertheless, the authors must be aware of the problem.
It must also be observed that Python contains a random number generator. It could have been used in the work.

[3] In this definition the function cosine has the role of turning the sequence s(n) into a sequence of numbers in the interval [-1,1]. But this creates some problem with the uniform distribution that is mentioned in the previous note. If the sequence s(n) is uniformly distributed over its range, this is no longer true for the sequence cos(s(n)). Roughly speaking, neighborhoods of 1 or -1 will contain 'more' values than neighborhoods of 0, for instance. Given that the values of s(n) belong to the interval $[\left(-6\pi\sqrt{3}\right)^3, \left(6\pi\sqrt{3}\right)^3]$, the value x could have been defined as $\frac{\text{xp}}{\left(6\pi\sqrt{3}\right)^3}$, and similarly for y and z.

[4] Strictly speaking, it is not correct to say that the points (x,y,z) fulfilling the first conditions below belong to an edge of the cube, which is a segment. These points are randomly distributed in square cuboids with the length of the square sides equal to 0.2 and with an edge coinciding with one of the cube. Of course, the approximation makes sense, but it should have been pointed out that we are considering "non-segment" edges.