Formation de franges d'interférences particules par particules

N. Schlosser

*
* *

I. Objectif :

Construire une animation simulant l'impact des photons arrivant un par un sur un écran où les franges d'interférence sont sinusoïdales

II. Programmation :

1. Tirage d'une variable aléatoire

On désire effectuer un tirage de différentes valeurs d'une grandeur *X :* x_1, x_2, ... x_N , sachant que X possède une densité de probabilité continue *p(X)*.

L'idée consiste à utiliser *R*, une variable aléatoire qui suit la loi uniforme sur l'intervalle [0, 1], puis de former, à partir de *R*, la variable aléatoire *X* qui suit la loi *p(X)* demandée. Ainsi à partir d'une simulation r_1, r_2, ... , r_N (en principe connue) de *R*, on obtiendra les valeurs x_1, x_2, ... x_N cherchées.

On utilise la *méthode de la transformation inverse* pour obtenir une simulation numérique d'une loi de densité continue *p(X)*. Cette méthode se déduit immédiatement du théorème ci dessous :

Théorème : Soit *p* la densité d'une loi et P la fonction de répartition de cette loi (densité de probabilité cumulée). P étant une bijection croissante de l'intérieur du support de p sur]0, 1[, il existe une fonction réciproque P^{-1}.

Alors, la variable aléatoire X= F^{-1}(R) suit la loi de densité p (où R, conformément à nos notations, suit la loi uniforme sur]0, 1[ )

Preuve « à la physicienne » : Supposons que les grandeurs Y et X soit liées par $Y=P(X)$ où P est la densité de probabilité cumulée de la variable X. Si la densité de probabilité de Y est uniforme sur [0,1] , la probabilité élémentaire de trouver Y entre y_0 et y_0+dy est de $dp=1\times dy=dy=P'(x_0)dx=p(x_0)dx$. A cause de la relation entre Y et X, c'est également la probabilité élémentaire de tirer X entre $x_0=P^{-1}(y_0)$ et x_0+dx .

L'algorithme est donc évident :

- On tire un nombre aléatoire `Alea` avec une répartition uniforme entre 0 et 1.
- On calcule x en appliquant la réciproque de P à `Alea` si elle est explicitement connue : $x_0=P^{-1}(Alea)$
- Si la réciproque P^{-1} n'est pas connue, on effectue la résolution numérique de l'équation $P(x)-Alea=0$ qui n'a qu'une unique solution puisque P est une bijection.

2. Distribution de probabilité normalisée et distribution cumulée

Pour des franges sinusoïdales, la densité de probabilité en x est de la forme :

$$p(x)=A(1+\cos(2\pi x))$$

La probabilité cumulée est donc de :

$$P(x)=\int_{X_{min}}^{x} A(1+\cos(2\pi x))\,dx=A\left[x-X_{min}+\frac{\sin(2\pi x)-\sin(2\pi X_{min})}{2\pi}\right]$$

La constante de normalisation vaut donc : $A=\frac{1}{X_{max}-X_{min}+\frac{\sin(2\pi X_{max})-\sin(2\pi X_{min})}{2\pi}}$

3. Description du programme et résultats :

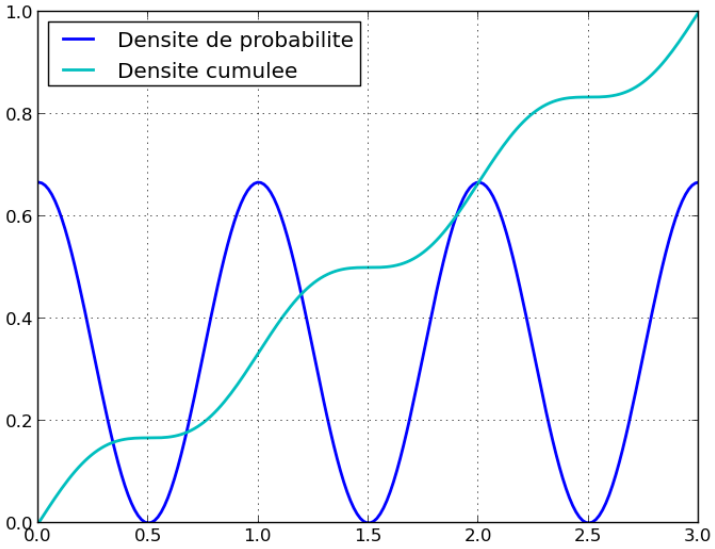
a. Méthode en deux étapes

Une première solution consiste à travailler étape par étape, pour bien contrôler au fur et à mesure ce que l'on programme.

On commence par importer les modules, définir les paramètres généraux et les distributions de probabilité et de probabilité cumulée que l'on trace pour vérifier leurs propriétés.

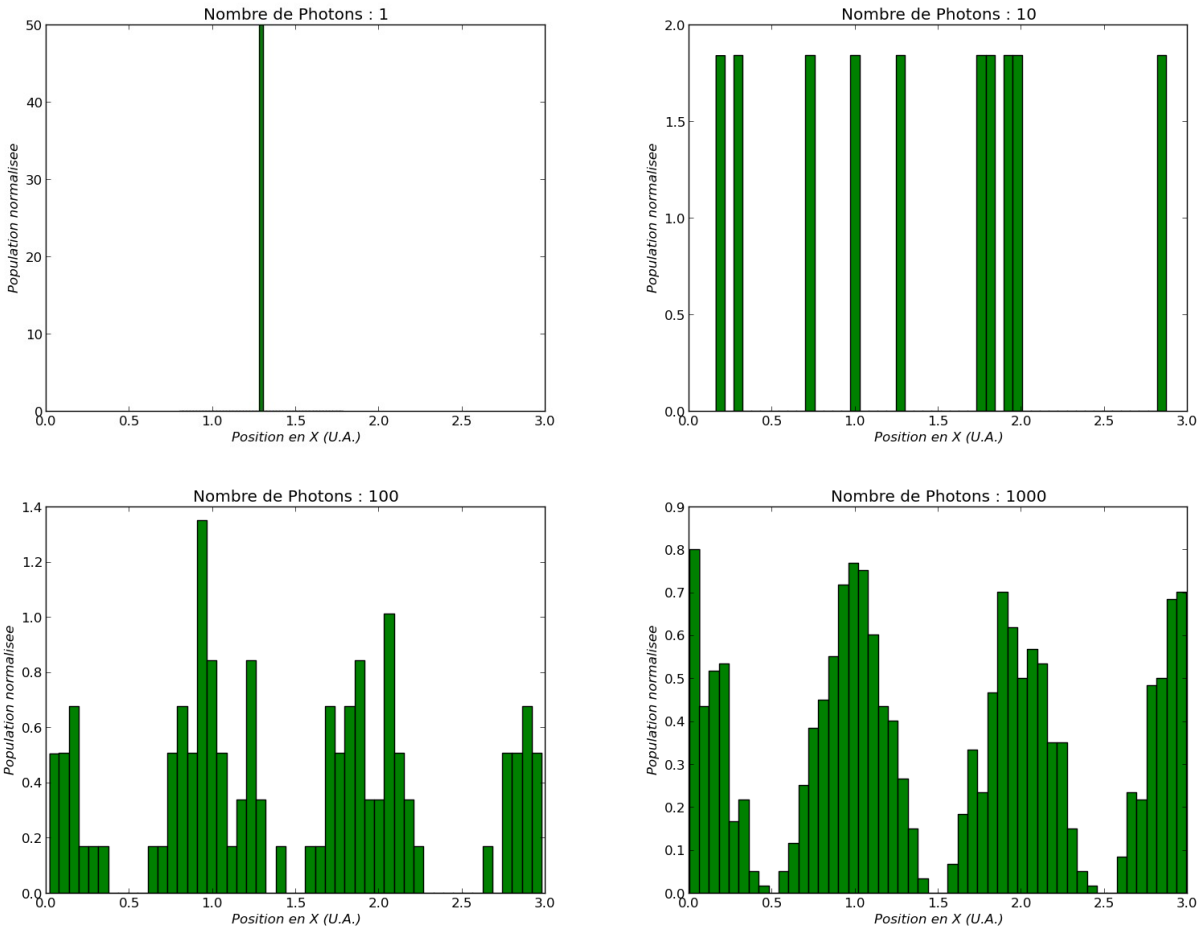
```
import numpy as np
import random as rd
import scipy.optimize
import matplotlib as mp
import matplotlib.pyplot as plt
from math import sin,cos,pi

#########################################################
Xmin=0  ; Xmax=3
Ymin=-1  ; Ymax=1
Xpixel=50 #Nombre de pixels selon X
Ypixel=50 #Nombre de pixels selon Y
Nbre_Photons=10000
#########################################################

def p(x,Xmin,Xmax):
    """Densite de probabilite en x, elle est uniforme sur y"""
    y=(1+cos(2*pi*x))/(Xmax-Xmin+(sin(2*pi*Xmax)-sin(2*pi*Xmin))/(2*pi))
    return y

def P(x,Xmin,Xmax):
    """Densite de probabilite cumulee en x"""
    y=(x-Xmin+(sin(2*pi*x)-sin(2*pi*Xmin))/(2*pi))\
            /(Xmax-Xmin+(sin(2*pi*Xmax)-sin(2*pi*Xmin))/(2*pi))
    return y

#Trace des densité de probabilité et probabilité cumulées
X=np.linspace(Xmin,Xmax,200)
Y1=[p(x,Xmin,Xmax) for x in X]
Y2=[P(x,Xmin,Xmax) for x in X]
plt.plot(X,Y1,'b-',label='Densite de probabilite')
plt.plot(X,Y2,'c-',label='Densite cumulee')
plt.grid()  ; plt.legend(loc='upper left')  ; plt.show()
```

On obtient les courbes suivantes :



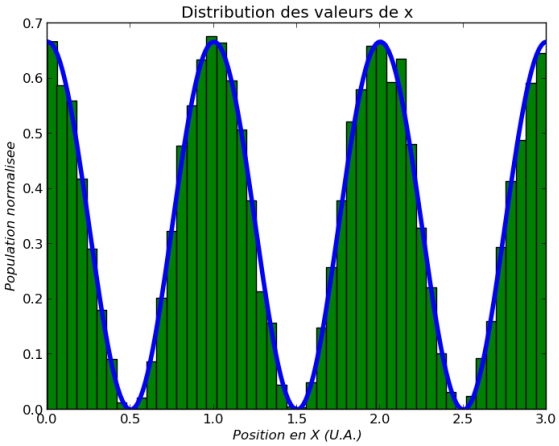
Ensuite, on peut tirer les positions (X,Y) des points d'impact des photons successifs. On stocke ces valeurs dans deux listes : ListeX et ListeY. Au cour de la construction, on visualise les histogrammes de répartition en X des photons reçus pour certaines valeurs du nombre de photons reçus, ici, 1, 10, 100, et 1000.

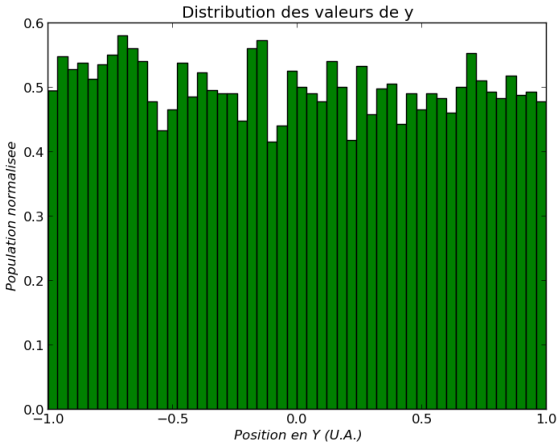
```
#initialisation des listes des positions des photons
ListeX=[]
ListeY=[]

#Tirage des valeurs de x et y
for i in range(Nbre_Photons):
    # Tirage des valeurs de x et y que l'on range dans deux listes
    # x par la méthode de la transformation inverse du cours :
            # Simulation_Process_Markov.pdf
    # La réciproque n'est pas explicite : résolution d'une équation
    alea=rd.random()
    ListeX.append(scipy.optimize.fsolve(lambda t:P(t,Xmin,Xmax)-alea,0)[0])
    # la densité de proba pour y est uniforme
    ListeY.append(Ymin+(Ymax-Ymin)*rd.random())
    #affichage de l'histogramme pour certaines
    #valeurs du nombre de photons
    if i in [0,9,99,999]:
        plt.hist(ListeX,50,normed=True,color='g')
        plt.xlabel('Position en X (U.A.)',fontstyle='italic')
        plt.ylabel('Population normalisee',fontstyle='italic')
        plt.title('Nombre de Photons : '+str(i+1))
        plt.xlim(Xmin,Xmax)
        plt.show()
```



On vérifie les distributions des valeurs de X et Y en traçant les histogrammes respectifs. Celui de la variable X est comparé avec la densité de probabilité $p(x)$.

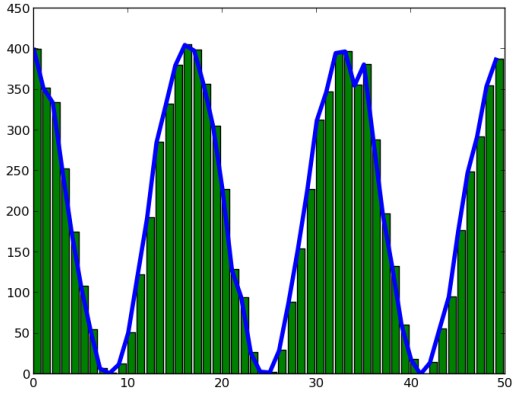
```
#Répartition finale en histogramme des valeurs de x et de y
plt.hist(ListeX,50,normed=True,color='g')
plt.plot(X,Y1,'b-',linewidth=4)
plt.xlabel('Position en X (U.A.)',fontstyle='italic')
plt.ylabel('Population normalisee',fontstyle='italic')
plt.title('Distribution des valeurs de x')
plt.show()
plt.hist(ListeY,50,normed=True,color='g')
plt.xlabel('Position en Y (U.A.)',fontstyle='italic')
plt.ylabel('Population normalisee',fontstyle='italic')
plt.title('Distribution des valeurs de y')
plt.show()
```





Dans une première approche, on commence par créer une image à une dimension de Xpixel de long. Cela revient en fait à construire un histogramme pour la variable X, mais avec un axe des abscisses gradué en pixels, qui peut être interprété comme un numéro de classe pour l'histogramme.

```
#Création de l'image d'interférences à 1D pour voir
Image=Xpixel*[0]
for i in range(Nbre_Photons):
    PositionX=int(Xpixel*(ListeX[i]-Xmin)/(Xmax-Xmin))
    #print PositionX,PositionY
    Image[PositionX]+=1
    #print Image

#Visualisation 1D
plt.bar(range(len(Image)),Image,color='g')
plt.plot(Image,'b',linewidth=4)
plt.show()
```



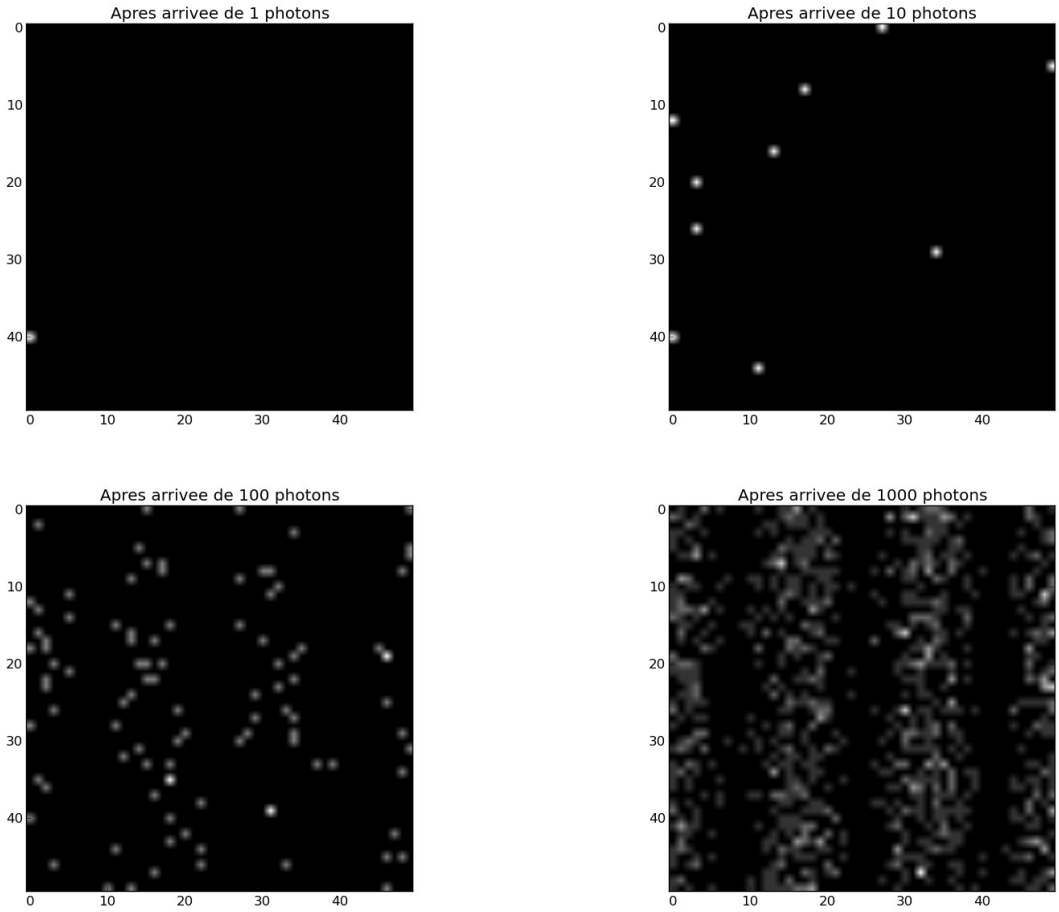
On termine par la création de l'image 2D observée sur la caméra. Pour cela on crée une image vide : tableau de dimension Xpixel par Ypixel remplis de 0. Pour chaque photon, on calcule le pixel sur lequel il tombe et on incrémente le pixel en question d'une unité. Au fur et à mesure de la construction de l'image, on visualise en niveau de gris les images pour certaines valeurs particulières du nombre de photons[(a)]. On termine par la visualisation de l'image finale.

```
# Création de l'image d'interférences 2D
Image=[[0 for i in range(Xpixel)] for j in range(Ypixel)]
for i in range(Nbre_Photons):
    # Calcul de la Position X et Y du photon reçu (en pixel)
    # C'est un facteur d'échelle pour passer de [Xmin,Xmax] à [0,Xpixel]
    PositionX=int(Xpixel*(ListeX[i]-Xmin)/(Xmax-Xmin))
    PositionY=int(Ypixel*(ListeY[i]-Ymin)/(Ymax-Ymin))

    # Incrémentation de la valeur du pixel où arrive le photon.
    Image[PositionY][PositionX]+=1

    # Affichage de l'image pour certaines valeurs
    if i in [0,9,99,999,9999]:
        plt.imshow(Image,cmap=mp.cm.gray)
        plt.title('Apres arrivee de '+str(i+1)+' photons')
        plt.show()

#Visualisation 2D
```

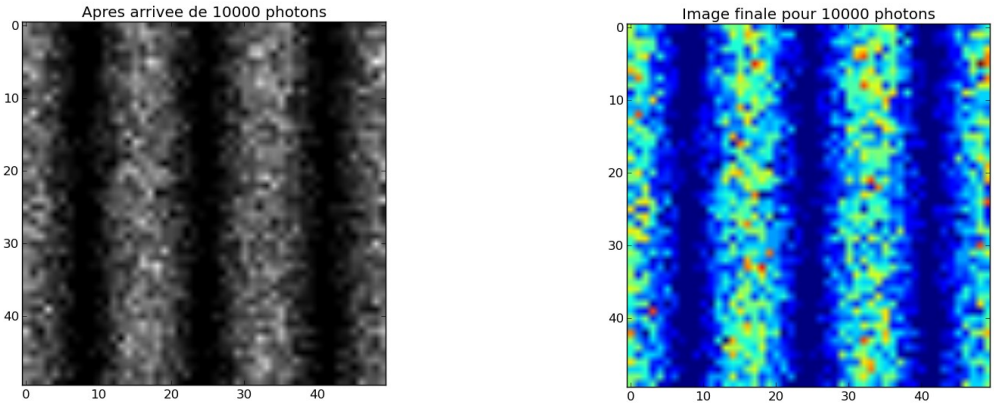
(a) Il subsiste un problème de normalisation (en intensité) qui n'est pas la même pour l'ensemble de images. Le pixel de nombre de photon maximum a toujours un niveau de gris maximum, soit un blanc parfait.

```
plt.imshow(Image)
plt.title('Image finale pour '+str(Nbre_Photons)+' photons')
plt.show()
```

On obtient successivement les images suivantes pour 1 10 100 et 1000 photons. On voit progressivement les franges d'interférence apparaître.



Sur l'image finale qui suit, correspondant à 10000 photons, on voit clairement les franges.



b. Méthode économisant la mémoire :

Pour économiser de la mémoire, on peut construire l'image au fur et à mesure des tirage des valeurs de X et Y. Pour un grand nombre N de photons, cela évite d'avoir à créer les deux N-listes, ListeX et ListeY. Seule l'image est stockée en mémoire, soit un tableau de $Xpixel \times Ypixel$. Le

programme est identique au précédent jusqu'au tracé des densités de probabilité et diffère ensuite.

```
for i in range(Nbre_Photons):
    # Tirage des valeurs de x et y
    # x par la méthode de la transformation inverse du cours :
            # Simulation_Process_Markov.pdf
    # la densité de proba pour y est uniforme
    Alea=rd.random()
    x=scipy.optimize.fsolve(lambda t:P(t,Xmin,Xmax)-Alea,0)[0]
    y=Ymin+(Ymax-Ymin)*rd.random()

    # Création de l'image d'interférences 2D
    # Calcul de la Position X et Y du photon reçu (en pixel)
    # C'est un facteur d'échelle pour passer de [Xmin,Xmax] à [0,Xpixel]
    PositionX=int(Xpixel*(x-Xmin)/(Xmax-Xmin))
    PositionY=int(Ypixel*(y-Ymin)/(Ymax-Ymin))

    # Incrémentation de la valeur du pixel où arrive le photon.
    Image[PositionY][PositionX]+=1

    # Sauvegarde ou affichage de l'image pour certaines valeurs
    if i in [1,10,50,100,1000,5000,9999]:
        # Cree la chaine NumeroImage à 8 caractères
        NumeroImage="{:8d}".format(i)
        # remplace les espaces précédents i par des 0
        NumeroImage=NumeroImage.replace(' ','0')
        plt.imshow(Image,cmap=mp.cm.gray)
        plt.title('Apres arrivee de '+NumeroImage+' photons')
        plt.savefig('Image/image'+NumeroImage)
#Visualisation 2D
plt.imshow(Image)
plt.title('Image finale avec '+str(Nbre_Photons)+' photons')
plt.show()
```

Ici, au lieu d'une visualisation des images 2D, on les sauvegarde en notant le nombre de photons reçus dans le nom du fichier. On finit toujours par la visualisation de l'image finale.

Pour finir, on utilise ImageMagick en ligne de commande pour créer un gif animé. Dans un terminal de commande, on se place dans le répertoire 'Image/' contenant les images numérotées. Puis on supprime les images ayant servi à faire l'animation.

```
> convert image*.png animation.gif
> rm image*.png
```