TP Introduction au monde quantique
Simulation d'interférences particule par particule.

Objectif : illustrer la dualité onde-corpuscule en réalisant une simulation d'interférences.

- Rédiger un code de façon claire avec des fonctions
- Respecter le nom de variables imposées
- Réaliser des graphiques scientifiques complets
- Réaliser une simulation d'interférences
- Comparer un résultat à un modèle théorique
- Présenter un résultat numérique

Ce TP se propose de simuler le résultat de l'expérience de Tonomura et al. (1992) (voir Fig. 1) vu en cours. On se limitera à une simulation à une dimension (moins gourmande en calcul).

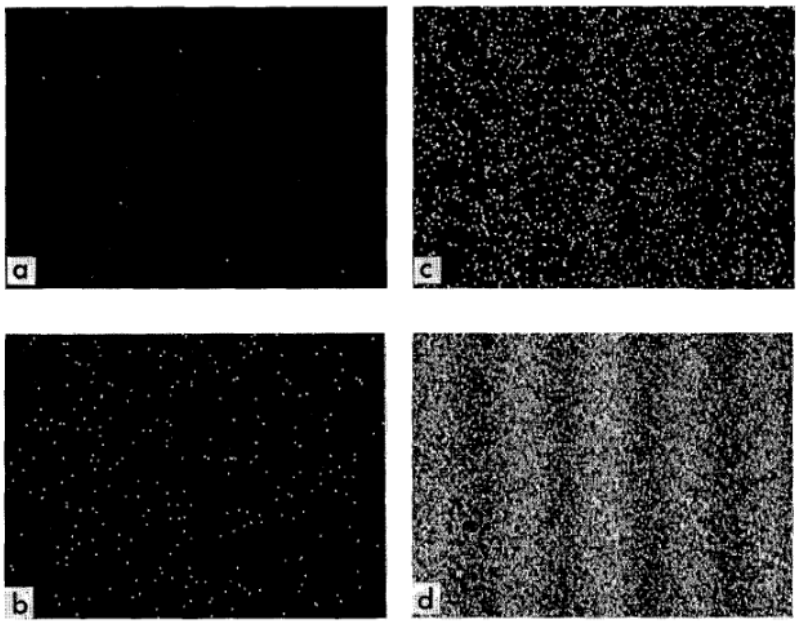

FIGURE 1 – Expérience d'interférences d'électrons. La figure d'interférences se construit au fur et à mesure que le nombre de particules augmente. Source : Tonomura et al, *Electron-holographic interference microscopy (1992).*

1 Principe

La simulation est dite à évènements :

— Création d'une particule et son trajet optique

— Détection de cette particule par un des capteurs sur le détecteur

Aucune information sur les particules ne sont précisées, elles peuvent être un photon ou un électron (expérience de Tonomura). Le résultat numérique que l'on doit obtenir est présenté en Fig. 2. La généralisation de cet algorithme à 3 dimensions donne le résultat en Fig. 3.

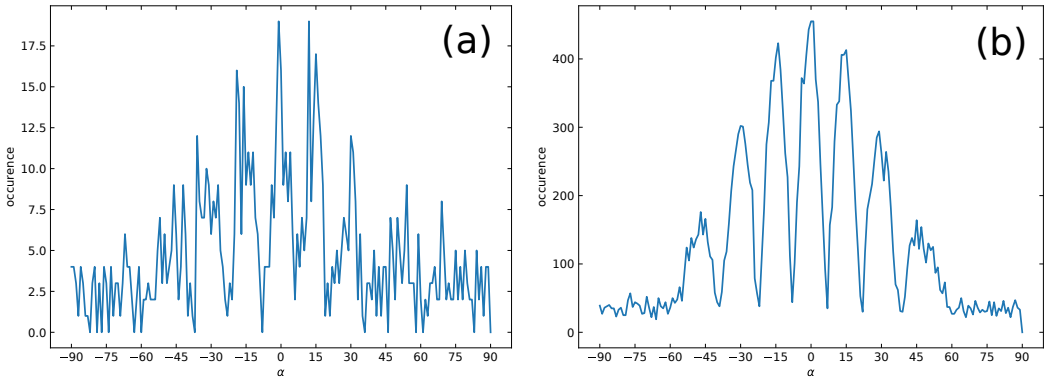

FIGURE 2 – Résultats de simulations 2D. (a) 10^4 particules. (b) 10^5 particules.

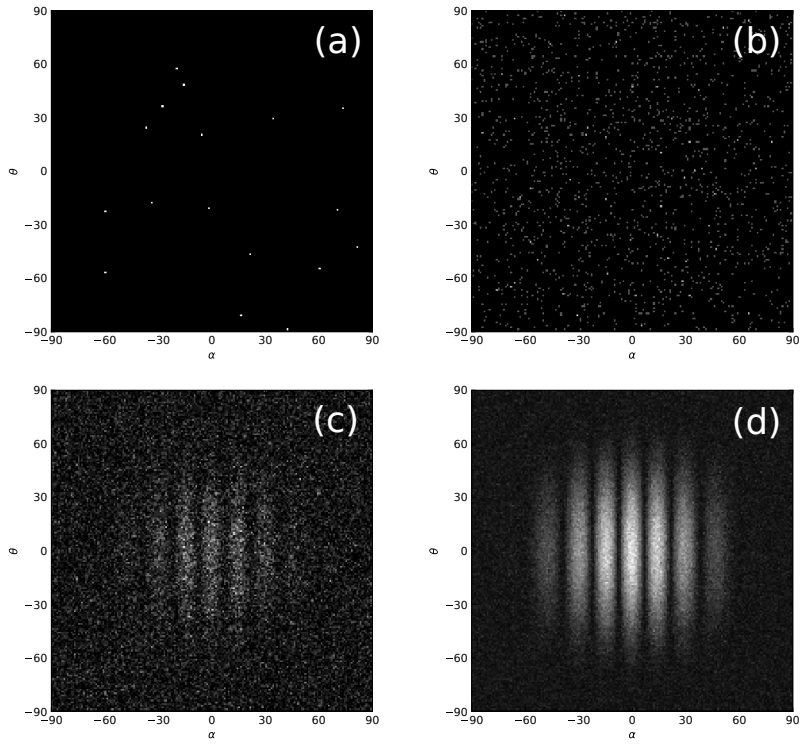

FIGURE 3 – Résultats de simulations 3D avec le même algorithme. (a) : 10^3 particules. (b) : 10^5 particules. (c) : 10^6 particules. (d) : 10^7 particules.

Les particules sont émises une par une par une source en amont et elles se propagent selon la direction x. Considérons une particule (voir la Figure 4) :

- La particule passe aléatoirement (avec une probabilité de 1/2) par une des fentes de largeur `a` $= \lambda$ et espacées de `b` $= 4\lambda$ avec une ordonnée `y` aléatoire. La distribution de probabilité sur l'ordonnée est uniforme.
- La particule est ensuite diffractée d'un angle `beta` aléatoire compris entre $-\pi/2$ et $\pi/2$. La distribution de probabilité sur l'angle est uniforme.
- La particule arrive ensuite au point M sur le détecteur circulaire de rayon `R` avec un champ électrique `Ex` et `Ey`.
- Le détecteur est circulaire et composé de `n = 181` capteurs individuels qui sont repérés par un indice `k` avec `k = 0, ..., 179, 180`.
- Un capteur possède une polarisation $\vec{p}_k$ notée `px_k` et `py_k` ; et un état de comptage `occ_k`.
- La détection de la particule dépend du champ électrique de la particule et de la polarisation du capteur. La condition est précisée plus bas.

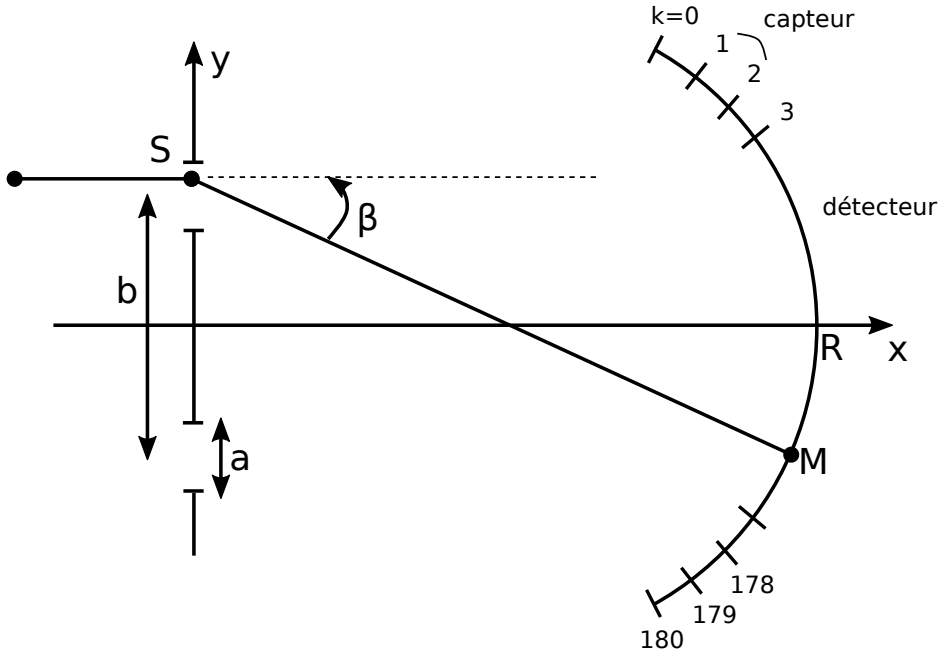

FIGURE 4 – Schéma de l'expérience. Une particule passe par la fente du haut au point S et arrive sur le détecteur au point M.

2 Algorithme

Une bonne façon d'aborder un code ambitieux est de vérifier le résultat d'une étape avant de passer à l'étape suivante. On commence par importer les librairies nécessaires :

```
import numpy as np
import matplotlib.pyplot as plt
```

2.1 Tirage de l'ordonnée

La probabilité de passer par une fente est 1/2. Le choix se fait par un tirage d'un nombre entier que l'on affecte à une fente. Ensuite on tire un deuxième nombre pour l'ordonnée `y` de la particule dans la fente. La largeur des fentes est `a = 1` et elles sont distantes de `b = 4`.

```
a    = 1     # largeur des fentes
b    = 4     # distance entre les fentes
R    = 100   # rayon de courbure du capteur
y    = []    # ordonnee de la particule a la fente
N    = 1000  # nombre de particules

for i in range(N):
    choix_fente = np.random.randint(...,...)
    if choix_fente == ...:
        y.append(np.random.uniform(-b/2 -a/2, -b/2 +a/2))
    else:
        y.append(...)

plt.hist(...,...)
plt.show()
```

1. Remplacer les pointillés par les arguments appropriés à l'aide de l'annexe.

2. Représenter la distribution des valeurs de `y` dans un histogramme de 50 points.

3. La distribution des valeurs de `y` est-elle vraiment uniforme ?

À présent il n'est plus nécessaire de conserver dans une liste les informations sur `y`. On peut simplifier le code en remplaçant les instructions :

```
    y.append(np.random.uniform(...,...))
```

par :

```
    y = np.random.uniform(...,...)
```

et ces instructions peuvent être supprimées :

```
plt.hist(beta,50)
plt.show()
```

4. Effectuer ce changement pour la variable `y`.

2.2 Tirage de l'angle de diffraction

Le tirage de l'angle `beta` est aléatoire et distribué uniformément entre `-np.pi/2` et `np.pi/2`.

5. Implémenter le choix d'un angle de diffraction `beta` pour les `N` particules.

2.3 Chemin optique

6. Exprimer le chemin optique `L` $= (SM)$ en fonction de `R`, `y` et `np.sin(beta)`.

7. Implémenter dans une fonction `longueur(y,beta)` prenant en entrée `y` et `beta` de la particule et en sortie `L`. Commenter les valeurs obtenues.

2.4 Champ électrique

L'amplitude du champ électrique de la particule source en $S(0, y)$ est 1 et il faut calculer sa valeur en $M(R\cos\beta, R\sin\beta)$. On pose $\lambda = 1$. L'amplitude du champ électrique en M est (résultat admis) :

$$\begin{cases} E_x(M) = \cos\left(\dfrac{2\pi}{\lambda}(SM)\right) = \cos(2\pi L), \\ E_y(M) = \sin\left(\dfrac{2\pi}{\lambda}(SM)\right) = \sin(2\pi L). \end{cases} \tag{2.1}$$

8. Implémenter dans une fonction `champ(L)` prenant en entrée le chemin optique `L` de la particule et en sortie `Ex` et `Ey`.

2.5 Indice du capteur

Le détecteur est circulaire de rayon `R` et est composé de `n = 181` capteurs individuels. Chaque capteur `k` est repéré par un angle α_k compris entre $-\pi/2$ et $\pi/2$. La particule sort de la fente avec un angle `beta` et on souhaite savoir sur quel capteur elle est enregistrée, c'est-à-dire le numéro `k`.

9. Implémenter dans une fonction `capteur(beta)` prenant en entrée `beta` et en sortie le numéro `k` du capteur (un entier) sur lequel la particule est enregistrée. Pour simplifier on considère que la particule est située initialement à l'origine. On utilisera les instructions suivantes :

```
n     = 181                                # nombre de capteurs
alpha = np.linspace(-np.pi/2,np.pi/2,n) # distribution angulaire du detecteur
```

2.6 Détection (admis)

Le champ électrique d'une particule arrivant sur une cellule `k` entraîne un changement de polarisation du capteur. La polarisation du capteur `k` est traitée en perturbation [1] :

$$\vec{p_k} \leftarrow g\vec{p_k} + (1-g)\vec{E}, \tag{2.2}$$

avec `g = 0.99` le terme de "mémoire" du capteur. En python cela s'écrit :

```
px_k = g*px_k+(1-g)*Ex
py_k = g*py_k+(1-g)*Ey
```

1. Les curieux sont encouragés à se renseigner davantage.

La détection d'une particule unique n'est pas assurée et peut être confondue avec le bruit. Le `bruit` est modélisé par un nombre aléatoire[2] compris entre 0 et 1 :

— Si la norme `P` de la polarisation est plus grande que le `bruit` : on ajoute +1 au comptage.

— Sinon le comptage ne change pas.

Le code suivant réalise cet algorithme :

```
def detection(px_k,py_k,Ex,Ey):
    g = 0.99        # "memoire" du capteur
    px_k = g*px_k+(1-g)*Ex
    py_k = g*py_k+(1-g)*Ey
    P    = (px_k**2+py_k**2)**0.5
    bruit= np.random.sample()
    test = P-bruit

    if test >= 0:
        m = 1
    else:
        m = 0
    return px_k,py_k,m
```

10. Initialiser une polarisation `px_k` et `py_k` pour chaque capteur ainsi que son état de comptage `occ_k` à l'aide de l'annexe.

11. Ajouter la fonction `detection(px_k,py_k,Ex,Ey)` au script.

12. D'après l'annexe, qu'effectue l'instruction `np.random.sample()` ?

13. Que représente la variable `m` ?

À présent le programme doit être proche du code suivant :

```
for i in range(N):
    choix_fente = np.random.randint(...,...)
    if choix_fente == ...:
        y = np.random.uniform(-b/2 -a/2, -b/2 +a/2)
    else:
        y = ...

    beta = np.random.uniform(...,...)
    L  = longueur(y,beta)
    Ex,Ey = champ(L)
    k  = capteur(beta)

    # entree: pol. initiale + Ex,Ey / sortie: pol. finale + detection
    px_k[k],py_k[k],m = detection(px_k[k],py_k[k],Ex,Ey)
    occ_k[k]          = occ_k[k] + m     # comptage de particules
```

2. La nature du bruit n'est pas importante. Ici on se propose un bruit distribué uniformément mais on pourrait prendre une distribution normale.

3 Étude quantitative

14. Représenter graphiquement le nombre de détection en fonction de l'abscisse angulaire. Réaliser une simulation pour $N = 10^3, 10^4,$ et 10^5. **Enregistrer les figures dans une page à rendre au format pdf.** ***Ne pas rendre le code.***

15. Commenter l'évolution des résultats obtenus en fonction de `N`.

16. Mesurer l'interfrange angulaire des simulations.

17. Comparer l'interfrange angulaire à sa valeur attendue.

18. En se basant sur le cours, relier les grandeurs de la simulation aux indéterminations spatiales.

19. À quel aspect ondulatoire ou corpusculaire ce résultat est-il associé ? Conclure.

Annexe

Librairies utilisées pour cette simulation :

```
import numpy as np
import matplotlib.pyplot as plt
```

Méthodes utiles pour cette simulation :

```
np.linspace(a,b,n)      # n valeurs regulierement espacees entre a et b
np.zeros(n)             # "liste" de n zeros (numpy array)
```

Nombres aléatoires :

```
np.random.randint(0,2)  # genere aleatoirement un entier entre 0 inclut et 2
↪ exclu
np.random.uniform(a,b)  # genere aleatoirement un nombre entre a inclut et b
↪ inclut
np.random.sample()      # genere aleatoirement un nombre entre 0 inclut et 1
↪ exclu
```

Méthodes mathématiques :

```
np.pi
np.sin(x)
np.sqrt(x)
```

Méthodes graphiques :

```
plt.plot(x,y)
plt.hist(x,bins=10)     # trace un histogramme de x sur 10 echantillons
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

Méthodes gestion de fichier :

```
np.savetxt("nom.txt",np.transpose([x,y]))
x,y = np.loadtxt("nom.txt",unpack=True)
```

```
import numpy as np
import matplotlib.pyplot as plt

def longueur(y,beta):
    l = (R**2 + y**2 - 2*R*y*np.sin(beta))**0.5
    return l

def champ(l):
    Ex = np.cos(2*np.pi*l)
    Ey = np.sin(2*np.pi*l)
    return Ex,Ey

def cellule(beta):
    for i in range(n-1):
        if alpha[i] <= beta <= alpha[i+1]:
            k = i
    return k

def detection(px_k,py_k,Ex,Ey):
    g = 0.99        # "memoire" du capteur
    px_k = g*px_k+(1-g)*Ex
    py_k = g*py_k+(1-g)*Ey
    P    = (px_k**2+py_k**2)**0.5
    bruit= np.random.sample()
    test = P-bruit

    if test >= 0:
        m = 1
    else:
        m = 0
    return px_k,py_k,m

N  = 50000

# parametres fentes
a  = 1
b  = 4
# parametres detecteur
R  = 100
n  = 181            # chaque capteur est espace de 1 degre
px_k  = np.zeros(n) # chaque capteur possede une polarisation
py_k  = np.zeros(n)
occ_k = np.zeros(n)
alpha = np.linspace(-np.pi/2,np.pi/2,n)

for i in range(N):   # a chaque iteration, creation d'une particule
    choix_fente = np.random.randint(0,2)  # 2 est exclu donc 0 ou 1
    if choix_fente == 0:
        y = np.random.uniform(-a/2-b/2 , a/2-b/2)
    else:
        y = np.random.uniform(-a/2+b/2 , a/2+b/2)

    beta  = np.random.uniform(-np.pi/2,np.pi/2)
    L     = longueur(y,beta)
    Ex,Ey = champ(L)
    k = cellule(beta)
    px_k[k],py_k[k],m = detection(px_k[k],py_k[k],Ex,Ey)
    occ_k[k] += m

plt.plot(alpha*180/np.pi,occ_k)
plt.xticks([-90,-75,-50,-25,0,25,50,75,90])
plt.xlabel(r'$\alpha$') # r et $ pour utiliser du latex
plt.ylabel('occurence')
plt.show()
```