Diffusion et marche au hasard

On considère un phénomène de diffusion particulaire (diffusion d'une goutte de lait dans du café, diffusion de parfum dans l'air) à partir d'un point source O (origine du repère) pendant une durée τ.
En raison des chocs incessants entre les molécules diffusant (le lait ou le parfum) et les molécules du fluide support (le café ou l'air dans les exemples précédents), la diffusion de particules est modélisée par une **marche au hasard** : après un choc, la direction de la molécule est arbitraire.
Par définition, entre deux chocs successifs, la molécule diffusant parcourt une distance ℓ appelée **libre parcours moyen** (l.p.m. en abrégé).
On établit dans le cours de thermodynamique que la distance L parcourue par le front de diffusion au bout du temps τ est, en ordre de grandeur : $L = \sqrt{D\tau}$ où D est le **coefficient de diffusion** (constante caractéristique des particules impliquées dans la diffusion à T et P donnés).

Le **front de diffusion** délimite la zone de l'espace dans laquelle la diffusion est notable, cette notion deviendra plus claire grâce aux simulations de ce TP (une goutte d'encre sur un papier absorbant s'élargit au cours du temps : on voit le front de diffusion progresser).

Dans la suite, on envisage une diffusion à deux dimensions (dans un plan).

Objectifs ✔️

1. Simuler la diffusion par une marche au hasard et mettre en évidence qualitativement la notion de front de diffusion.
2. Vérifier que la relation $L = \sqrt{D\tau}$ est statistiquement correcte (elle est vérifiée dès qu'un « grand » nombre de molécules est impliqué, on cherchera à déterminer l'ordre de grandeur minimum de ce nombre).

📝 *import*

Il existe plusieurs façons d'importer un module (ici le module random).

1. `from random import *`
2. `import random`
3. `import random as rd`

Syntaxes correspondantes pour générer un entier e tel que $a \leq e \leq b$:

1. `randint(a,b)`
2. `random.randint(a,b)`
3. `rd.randint(a,b)`

L'inconvénient de la première syntaxe est qu'il existe une fonction randint dans plusieurs modules différents, il y a donc risque de confusion.
La deuxième syntaxe est un peu longue...
La troisième utilise un alias (rd) qui permet d'abréger le nom du module.

Module turtle

Les éléments de syntaxe suivants sont nécessaires dans la suite.
La bibliothèque turtle permet d'effectuer des dessins à l'aide d'un stylo nommé tortue.

`from turtle import *`	Charger le module
`tor = Turtle()`	Créer une tortue nommée tor
`tor.forward(10)`	Avancer la tortue de 10 pixel
`tor.left(30)`	Tourner la tortue de 30° vers la gauche(*)
`tor.right(...)`	
`tor.xcor()`	Récupérer l'abscisse de la tortue
`tor.ycor()`	
`tor.goto(20, 50)`	Aller au point de coordonnées x=20 et y=50. La position (0, 0) est au centre de la fenêtre
`tor.penup()`	Lever le stylo
`tor.pendown()`	Baisser le stylo
`tor.speed(0)`	Vitesse maximum
`tor.color()`	Couleur de la tortue
`done()`	⚠️ Indispensable : terminer (en fin de script)

(*) ⚠️ Après avoir fait tourner la tortue, il faut avancer car l'instruction tor.left(...) oriente la tortue mais ne la fait pas avancer.
✏️ Préliminaires : tester rapidement ces instructions (chargement de la bibliothèque, avancer, tourner, avancer).

```
from turtle import *
import random as rd

# Test module turtle
```

Marche au hasard et front de diffusion

La liste `couleurs` permet de définir plusieurs couleurs dans ce notebook (il existe des méthodes plus performantes mais elles ne sont pas implémentées dans ce notebook).
La liste `couleur` et l'entier `Index_max_coul` seront considérés comme des variables globales.

💻 Valider les deux cellules ci-dessous.

```
couleurs = ['aquamarine', 'bisque', 'blue', ... 'violet', 'violetred', 'yellow', 'yellowgreen']

# Index de la dernière couleur de la liste (utile pour le tirage aléatoire des couleurs)
Index_max_coul = len(couleurs) - 1
```

Etape 1 : marche au hasard d'une particule pendant le temps τ

La durée totale de la simulation est τ (dans une unité de temps arbitraire).
Les instants t_i de la simulation sont 1, 2, 3 ... τ.
A chacun de ces instants, un angle a est tiré au hasard et un pas de longueur ℓ (l.p.m) est effectué dans la direction choisie (dans une unité de distance arbitraire).

✏️ Exercice 1
Ecrire une fonction `marche_hasard(tor, lpm=20, tau=50)` traçant la marche au hasard d'une tortue nommée tor (créée en dehors de la fonction) sur un nombre tau de pas de longueur lpm chacun (paramètres possédant des valeurs par défaut).
Tester la fonction après avoir créé une tortue.

```
# Test marche_hasard

tor = Turtle()                # Création de la tortue
marche_hasard(tor)
done()                        # Instruction INDISPENSABLE
```

✏️ Exercice 2
Perfectionner la fonction précédente en choisissant une couleur de tracé aléatoire (pour la totalité du tracé et non pour chaque pas) ; utiliser la liste *couleurs* et l'entier *Index_max_coul*.
On souhaite tracer la marche de deux molécules.

```
# Test avec couleur et vitesse maximum - Deux particules

tor = Turtle()

# Vitesse maximum
tor.speed(0)

# Instructions pour matérialiser le point de départ par un disque rouge
tor.begin_fill() ; tor.fillcolor('red') ; tor.circle(5) ; tor.end_fill() ; tor.fillcolor('black')

marche_hasard(tor)
marche_hasard(tor)
done()
```

✏️ Les tracés se succèdent alors que chaque tracé devrait débuter au centre (les particules diffusent depuis le centre). Remédier au problème (modifier à nouveau la fonction) en veillant à ne tracer aucune ligne supplémentaire.

```
# Test fonction modifiée

tor = Turtle()
tor.speed(0)
tor.begin_fill() ; tor.fillcolor('red') ; tor.circle(5) ; tor.end_fill() ; tor.fillcolor('black')

marche_hasard(tor)
marche_hasard(tor)
done()
```

Etape 2 : marche au hasard de N particules pendant le temps τ

On simule la marche au hasard de N particules à partir du point d'injection O en répétant la marche au hasard précédente N fois depuis le point (0,0).

✏️ Exercice 3
Ecrire une fonction `diffusion(N, lpm=20, tau=50)` réalisant les opérations suivantes :
- création d'une tortue ;
- réglage de la vitesse au maximum ;
- tracé de N marches au hasard.

Tester avec $N = 3$.

```
# Test fonction diffusion

diffusion(3)
done()
```

Etape 3 : coefficient de diffusion D

Pour chacune des N marches au hasard, on calcule la quantité $D = L^2/\tau$ où L est la distance parcourue depuis le point de départ de la marche au hasard jusqu'au point d'arrivée (à vol d'oiseau et non suivant le chemin réellement suivi) puis on calcule le coefficient de diffusion moyen $Dmoy =< D >$.

✏️ Exercice 4
Modifier la fonction `diffusion(N, lpm=20, tau=50)` de façon à ce qu'elle calcule le coefficient de diffusion D déduit de chaque marche au hasard et qu'elle renvoie la valeur D_{moy}.
Remarques :

- Les coordonnées (x_f, y_f) du point final de la marche sont obtenues très facilement grâce aux instructions disponibles dans le module turtle.
- La quantité L^2 évaluée entre le point initial $(0, 0)$ et le point final (x_f, y_f) est alors très simple à calculer.

Tester avec $N = 3$ pour vérifier le bon fonctionnement puis augmenter N (ne pas dépasser quelques dizaines) et observer. Un "front de diffusion" est-il perceptible avec quelques dizaines de particules ?

```
# Test diffusion avec calcul Dmoy

N = 100
Dmoy = diffusion(N)
print(f"Dmoyen = {Dmoy:.1f} pour N = {N} marches au hasard")
done()
```

💻 👁️ Effectuer plusieurs simulations en notant la valeur de D_{moy} pour chacune d'elles.
Que constate-t-on pour la valeur moyenne de D avec la valeur de N proposée ? Commenter.

Marche au hasard et relation $L = \sqrt{D\tau}$

On souhaite à présent effectuer des simulations avec un nombre N de particules beaucoup plus élevé (quelques milliers), on va donc écrire un programme purement numérique sans utiliser le module turtle afin de gagner en rapidité.

```
import math
import matplotlib.pyplot as plt
from matplotlib.patches import Circle
import random as rd
```

✏️ Exercice 5.1
Ecrire une fonction `marche_hasard(lpm=20, tau=50)` renvoyant le tuple (x, y) où x et y sont les coordonnées de la particule à la fin d'une marche au hasard constituée d'un nombre tau de pas de longueur lpm chacun (abscisse et ordonnée finale de la particule).
Un angle (valeur entière) est tiré au hasard à chaque pas, les distances parcourues au cours d'un pas sur les deux axes x et y sont calculées (on pourra charger la bibliothèque math afin de disposer math.cos, math.sin et math.pi) et les distances parcourues depuis le point (0, 0) sont incrémentées.

```
# Test marche_hasard

marche_hasard(lpm=20, tau=500)
```

✏️ Exercice 5.2
Ecrire une fonction `diffusion(N, lpm=20, tau=50)` renvoyant X, Y et D_{moy} où X (resp. Y) est la liste des abscisses (resp. ordonnées) finales correspondant aux N simulations.
Cette fonction appelle la précédente, pour chaque particule les coordonnées x et y en fin de marche sont stockées et le coefficient D est calculé.

```
# Test diffusion

diffusion(10, lpm=20, tau=500)
```

✏️ Exercice 5.3
Tracer le nuage des points à l'aide des instructions suivantes :
`plt.scatter(X,Y) plt.axis('scaled') plt.show()`

```
# Test tracé     !!! Plusieurs secondes de calcul !!!

N = 10000  # Plusieurs dizaines de milliers pour "stabiliser" Dmoy
X, Y, Dmoy = diffusion(N)
print(f"Dmoyen = {round(Dmoy,3)} pour N = {N} marches au hasard")

plt.figure()
plt.scatter(X,Y)
plt.title(f"Dmoyen = {round(Dmoy,1)} pour N = {N} marches au hasard")
plt.axis('scaled')
plt.show()
```

💻 👁️ Exercice 6
Le script précédent peut être rendu beaucoup plus rapide en utilisant la bibliothèque **numpy**.

```
import numpy as np
```

💻 💡 6.1 Créer un tableau 2D (N lignes, tau colonnes) de valeurs aléatoires (flottants dans $[0, 1[$) avec numpy : exécuter le code ci-dessous.

```
N, tau = 5, 3
# Tableau N lignes x tau colonnes de valeurs aléatoires entre 0 et 1
T = np.random.random_sample((N, tau))
T
```

💻 💡 6.2 Il est alors possible d'utiliser ce tableau T dans une expression mathématique (par exemple 3*T**2+sin(T)/7-1) : la formule sera appliquée en utilisant les éléments des tableaux (de mêmes dimensions).

```
3*T**2 + np.sin(T)/7- 1  # Formule appliquée à chaque élément => résultat = tableau N . tau
```

💻 💡 6.3 Il également possible d'effectuer des opérations sur toutes les lignes (ou toutes les colonnes) en une seule instruction ou sur le tableau entier.

```
T = np.array([[1,2,3,4,5], [11,12,13,14,15]])
print(T)
Scol = np.sum(T, axis=0)   # axis=0 : appliquer la fonction à chaque colonne => résultat = tableau 1D
Slgn = np.sum(T, axis=1)   # axis=1 : appliquer la fonction à chaque ligne   => résultat = tableau 1D
Stab = np.sum(T)           # paramètre 'axis' non précisé : appliquer la formule au tableau entier
print('Somme des colonnes : ',Scol)
print('Somme des lignes :   ',Slgn)
print('Somme du tableau :   ',Stab)
```

💻 💡 6.4 Commenter le code ci-dessous et observer le graphe obtenu.

```
# Données de la modélisation
N = 20000
tau = 500
lpm = 20

#
a = np.random.random_sample((N, tau))*2*np.pi

#
x, y = lpm*np.cos(a), lpm*np.sin(a)

#
X, Y = np.sum(x, axis=1), np.sum(y, axis=1)

#
D = (X**2 + Y**2)/tau
```

```
#
Dmoy = np.sum(D)/N

# Tracé du nuage de points
plt.figure()
fig, ax = plt.subplots(1,1, figsize=(7,6))
plt.scatter(X, Y, s=5)
ax.set_aspect('equal')
plt.title(r"$D_{moyen}$" + f" = {round(Dmoy,1)} pour N = {N} marches au hasard")
plt.show()
```

✏️ Exercice 6.5 - Valeur de D fournie par la physique statistique
On peut justifier (cf. « Approche microscopique de la diffusion ») que le coefficient de diffusion D, le libre parcours moyen ℓ et la vitesse moyenne $\overline{v}$ entre deux collisions sont liés par la relation : $D \simeq \ell\overline{v}$.

1. Quelle est la valeur numérique (unités arbitraires de la simulation) de la durée Δt_{choc} entre deux chocs successifs dans les simulations précédentes ?
2. En déduire l'expression de la vitesse moyenne $\overline{v}$.
3. En déduire l'expression et la valeur numérique de D (dans les unités arbitraires de la simulation).
4. Cette valeur est-elle en accord avec l'estimation obtenue ci-dessus pour laquelle $\ell = 20$ (unités arbitraires) ?
5. Quelle valeur doit-on modifier dans le script précédent afin d'effectuer une nouvelle vérification ? Faire l'essai.

💻 💡 6.6 Commenter le code ci-dessous et commenter le graphe obtenu (quelle relation permet-il d'établir ?).

```
# Données de la modélisation
N = 20000
tau = 500
lpm = 20

a = np.random.random_sample((N, tau))*2*np.pi
x, y = lpm*np.cos(a), lpm*np.sin(a)

#
for i in range(1, tau):
    x[:, i] += x[:, i-1]
    y[:, i] += y[:, i-1]

#
dmoy2 = np.mean(x**2+y**2, axis=0)

# Tracé
plt.figure()
plt.plot(range(tau), dmoy2)
plt.xlabel('t')
plt.ylabel('dš(t)')
plt.title(f"N = {N} marches au hasard sur la durée tau = {tau} : dš(t)")
plt.show()
```

💻 👁️ Exercice 7.1 - Front de diffusion
Le script ci-dessous est identique à celui de la question 6.4, seules des informations graphiques ont été ajoutées (cf. légende du graphe).
Observer la distribution des particules dans l'espace.

```
#np.random.seed(100)

# Données de la modélisation
N = 20000
tau = 500
lpm = 20
```

```
# Angles tirés au hasard dans [0, 2pi[ = tableau N lignes, tau colonnes
a = np.random.random_sample((N, tau))*2*np.pi
# Abscisses et ordonnées = tableaux N x tau
x, y = lpm*np.cos(a), lpm*np.sin(a)
# Abscisses et ordonnées finales pour chaque marche = tableaux N lignes
X, Y = np.sum(x, axis=1), np.sum(y, axis=1)
# Coefficient de diffusion pour chaque marche
D = (X**2+Y**2)/tau
# Coefficient moyen
Dmoy = np.sum(D)/N

# Points ayant parcouru une distance supérieure à la distance moyenne
X1 = np.ma.masked_where(D < Dmoy, X)
# Points ayant parcouru une distance inférieure ou égale à la distance moyenne
X2 = np.ma.masked_where(D >= Dmoy, X)
# Points ayant parcouru une distance inférieure ou égale à 2 fois la distance moyenne
X3 = np.ma.masked_where(D >= 4*Dmoy, X)

# Tracés
plt.figure()
fig, ax = plt.subplots(1,1, figsize=(7,6))
# Cercle de rayon R = Rmoyen et R = 2 Rmoyen
r = np.sqrt(Dmoy*tau)
ax.add_artist(Circle((0, 0), r, fill=None, ec='b', label=r'$L_{parcourue} \leq L_{carac} = \sqrt{D\tau}$')
ax.add_artist(Circle((0, 0), 2*r, fill=None, ec='g', label=r'$L_{carac} \leq L_{parcourue} \leq 2 L_{carac
# Nuage de points
plt.scatter(X1, Y, c='red', s=5)
plt.scatter(X3, Y, c='green', s=3)
plt.scatter(X2, Y, c='blue', s=3)
# Titre, légende...
plt.title(r"$D_{moyen}$" + f" = {round(Dmoy,1)} pour N = {N} marches au hasard")
ax.set_aspect('equal')
plt.legend()
plt.show()
```

💻 👁️ Exercice 7.2 - Front de diffusion à $L = \sqrt{D\tau}$: combien de particules ?

```
#### Nombre de particules ayant parcouru une distance > L caractéristique
N1 = len(X1[~X1.mask]) # Pts rouges (y compris pts recouverts par les pts verts)

# Nombre de particules ayant parcouru une distance < L caractéristique
N2 = len(X2[~X2.mask]) # Pts bleus

# Nombre de particules ayant parcouru une distance < 2 L caractéristique
N3 = len(X3[~X3.mask]) # Pts verts et pts bleus

ps2L = round((N-N3)/N*100, 1)
p0L = round(N2/N*100, 1)
pL2L = round((N3-N2)/N*100, 1)
print(f"Pourcentage de particules ayant parcouru une distance inférieure à L : {p0L} %")
print(f"Pourcentage de particules ayant parcouru une distance comprise entre L et 2L : {pL2L} %")
print(f"Pourcentage de particules ayant parcouru une distance supérieure à 2L : {ps2L} %")

plt.figure()
pourcentages = [r'$d < L$', r'$L < d < 2L$', r'$d > 2L$']
valeurs = [p0L, pL2L, ps2L]
coul = ['blue', 'green', 'red']
plt.ylim(0,100)
plt.ylabel('Distance parcourue')
plt.bar(pourcentages, valeurs, color=coul)
for i in range(3):
```

```
    plt.text(i-0.1, valeurs[i]+1, str(valeurs[i])+' %')
plt.show()
```

💻 👁️ Exercice 8 - Détermination de D - Fluctuations statistiques
Le script ci-dessous donne une idée de la variabilité des résultats d'un tirage à l'autre.

```
def Nmarches(N, tau=500, lpm=20):
    a = np.random.random_sample((N,tau))*2*np.pi
    x, y = lpm*np.cos(a), lpm*np.sin(a)
    X, Y = np.sum(x, axis=1), np.sum(y, axis=1)
    D = (X**2+Y**2)/tau
    return np.sum(D)/N

lpm=30
N = 10000
nbre_tirages = 50
tirages = range(nbre_tirages)
Dm = [Nmarches(N, lpm=lpm) for t in tirages]
Dm_moyen = sum(Dm)/nbre_tirages

plt.figure()
plt.plot(tirages, Dm, 'b', label=f'D moyenne sur {N} particules (1 simulation)')
plt.plot(tirages, [Dm_moyen]*nbre_tirages, 'r',
         label=r"$D_{moyen}=\left\langle D \right\rangle_n$"+f" moyenne sur n = {nbre_tirages} simulations
plt.title(r"$D_{moyen}=\left\langle D \right\rangle_n$" +
          f" = {int(Dm_moyen)} pour {N} particules - n = {nbre_tirages} simulations")
plt.suptitle(r"Marche au hasard - Libre parcours moyen  $\ell$" + f" = {lpm}")
plt.legend()
plt.xlabel('nř simulation')
plt.ylabel('Coefficient de diffusion')
plt.show()
```