Étude numérique de la diffusion de particules

À l'échelle microscopique, la diffusion s'interprète par les mouvements aléatoires des particules sujettes à leur agitation thermique. Nous avons utilisé cette description pour relier le coefficient de Fick D au libre parcours moyen ℓ et au temps de vol τ. Pour cela, nous avons exprimé l'éloignement quadratique moyen à l'issue d'une marche aléatoire (paragraphe IV.3), puis nous l'avons l'identifié à celui obtenu par résolution de l'équation de la diffusion (paragraphe III.3). On obtient ainsi $D = \frac{\ell^2}{2\tau}$ en dimension 1 et $D = \frac{\ell^2}{4\tau}$ en dimension 2.

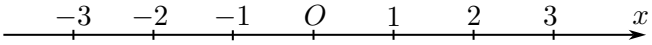
Plutôt que de mener un calcul d'espérance, on peut obtenir l'éloignement quadratique moyen par une « expérience numérique ». On simule le mouvement d'un grand nombre de particules identiques, puis on note pour chacune le point d'arrivée et sa distance à l'origine, et enfin on calcule la moyenne des carrés de ces distances. Selon la loi des grands nombres, la moyenne empirique ainsi obtenue se confond presque avec l'espérance trouvée par le traitement probabiliste. Par cette approche, nous pourrons aussi observer l'étalement progressif de l'ensemble des particules.

Les codes qui suivent utilisent les modules `numpy`, `matplotlib.pyplot` et `random` avec les préfixes respectifs `np`, `plt` et `rd`.

I Cas unidimensionnel

I.1 Marche alétoire individuelle

Pour commencer, simulons la marche aléatoire unidimensionnelle d'une particule effectuant N_{pas} pas aléatoires. À intervalles réguliers (durée τ), elle se déplace vers la droite ou vers la gauche d'une longueur ℓ. Par commodité, on choisit τ comme unité de temps et ℓ comme unité de longueur. Avec ce choix, les dates sont les entiers $t_i = 0, 1, 2, 3, ...$, les abscisses $x(t_i)$ valent $0, \pm1, \pm2, \pm3, ...$, et le coefficient de diffusion $D = \ell^2/(2\tau)$ prend la valeur $D = 1/2$. D'un instant au suivant, l'abscisse de la particule varie aléatoirement de $+1$ ou de -1 avec une égale probabilité.



La fonction `marche_alea` ci-dessous renvoie sous la forme d'un tableau numpy de $N_{\text{pas}} + 1$ entiers, les abscisses successivement occupées en commençant par la position initiale $x = 0$.

```
def marche_alea(N_pas, option) :

    positions = np.zeros(N_pas+1, dtype = int)

    if option == 0 :                                                    5
        for i in range(N_pas) :
            positions[i+1] = positions[i] + 2 * rd.randint(0, 1) - 1

    if option == 1 :
            deplacements = 2 * np.random.randint(0, 2, size = N_pas) - 1    10
            positions[1:] =  np.cumsum(deplacements)

    return positions
```

Deux options sont disponibles au choix de l'utilisateur.

— Dans l'option 0, on ajoute tour à tour les déplacements aléatoires valant ±1. En effet, `rd.randint(0,1)` renvoie aléatoirement 0 ou 1 donc `2 * rd.randint(0, 1)- 1` renvoie 1 ou -1.

— Dans l'option 1, on produit d'un seul coup un tableau contenant les N_{pas} pas aléatoires (ligne 10). Noter qu'avec `rd.randint`, la borne supérieur est incluse, alors qu'elle est exclue avec `np.random.randint`. À la ligne 11, plutôt que d'ajouter itérativement les déplacement, on utilise `np.cumsum` qui réalise la *somme*

cumulative des éléments du tableau. Pour un tableau A de N éléments $a_i, i \in [\![0, N-1]\!]$, le tableau B des sommes cumulatives possède lui aussi N éléments définis par

$$b_k = \sum_{i=0}^{k} a_i \quad .$$

Pour $N_{\text{pas}} = 10$ par exemple, un tirage aléatoire donne `deplacements = array([-1, 1, 1, 1, 1, 1, 1, -1, -1, -1])`. Alors `cumsum(deplacements)` renvoie `array([-1, 0, 1, 2, 3, 4, 5, 4, 3, 2])`. Le premier terme de ce tableau est `deplacements[0]`, puis les autres s'obtiennent en ajoutant tour à tour les éléments de `deplacements`.

L'option 1 est *beaucoup* plus rapide que l'option 0, avec un avantage d'autant plus sensible que N_{pas} grandit. Dans la suite, nous l'utiliserons systématiquement, l'option 0 n'ayant été présentée que dans un but pédagogique.

Voyons à quoi ressemble la marche aléatoire d'une particule. Comme la trajectoire se déroule le long de l'axe (Ox), la représenter ferait apparaître un segment ne présentant pas beaucoup d'intérêt. Il est plus pertinent d'observer les variations de x en fonction du temps. Le résultat est visible sur la figure 1 pour quatre marches aléatoires distinctes. Bien entendu, chacune des quatre particules possède un mouvement qui lui est propre. Certaines terminent leur course à proximité de l'origine, d'autres plus loin.

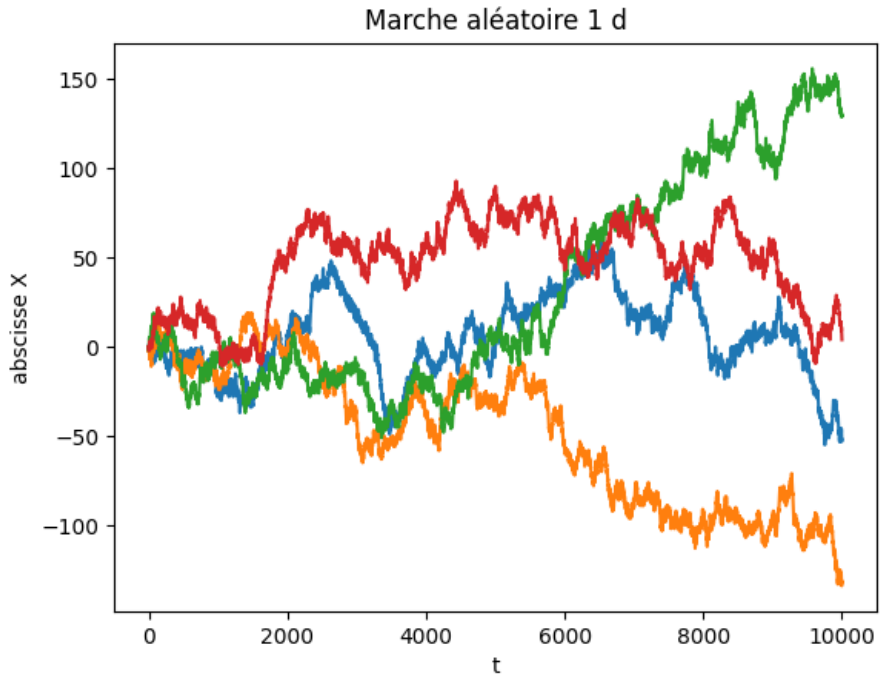


FIGURE 1 – Exemples de quatre marches aléatoires unidimensionnelles de 10000 pas.

I.2 Marche aléaoire d'un ensemble de particules

Les particules, au nombre de N_{part}, se trouvent initialement à l'origine O des coordonnées, puis chacune d'elles effectue une marche aléatoire de N_{pas} pas comme dans le paragraphe précédent. On stocke les positions de toutes les particules à tous les instants dans un tableau `positions` de $N_{\text{pas}} + 1$ lignes et de N_{part} colonnes. L'élément `positions[i, j]` est l'abscisse $x_j(t_i)$ de la particule d'indice j à l'intant i. La ligne du tableau `positions[i] = positions[i,:]` contient les abscisses de toutes les particules à l'instant i. La colonne `positions[:,j]` contient les abscisses de la particule d'indice j à tous les instants. Par exemple, le mouvement d'un ensemble de 3 particules sur 4 pas de temps est représenté par le tableau `positions` ci-dessous.

```
positions = array([[ 0.,  0.,  0.],
                   [ 1., -1.,  1.],
                   [ 0.,  0.,  0.],
                   [-1.,  1.,  1.],
                   [-2.,  2.,  0.]])
```

On obtient aisément ce résultat en itérant la fonction `marche_alea` du paragraphe précédent.

```
def marche_ensemble(N_pas, N_part) :
    positions = np.zeros((N_pas+1, N_part))
    for j in range(N_part) :
        positions[:, j] = marche_alea(N_pas, 1)
    return positions
```

I.3 Éloignement quadratique moyen

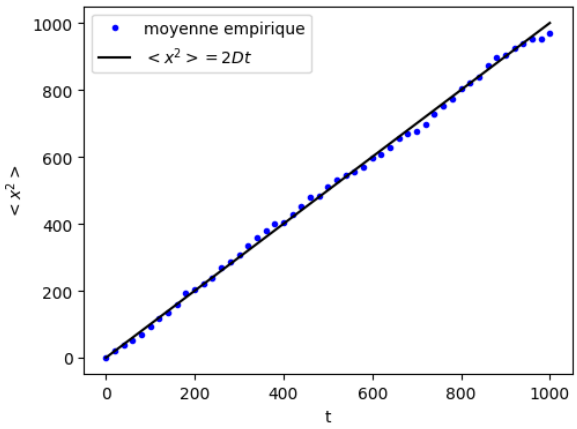
Sur un tableau numpy, les opérations arithmétiques se font terme à terme. En calculant `positions**2`, on obtient donc les carrés des abscisses de toutes les particules à tous les instants, sous la forme d'un tableau de taille $(N_{\text{pas}} + 1) \times N_{\text{part}}$. Les lignes de code suivantes permettent d'observer l'évolution temporelle de $< x^2 >$. La fonction `np.average` calcule la moyenne d'un tableau et le paramètre `axis = 1` signifie qu'il faut effectuer une moyenne en parcourant les colonnes, sur chaque ligne séparément. On obtient ainsi un tableau unidimensionnel de $N_{\text{pas}} + 1$ valeurs dont l'élément de d'indice i est la moyenne de x^2 à l'instant i.

```
positions**2
array([[0., 0., 0.],
       [1., 1., 1.],
       [0., 0., 0.],
       [1., 1., 1.],
       [4., 4., 0.]])
```

```
np.average(positions**2, axis = 1)
array([0., 1., 0. , 1., 2.67])
```

```
positions = marche_ensemble_bis(0, N_pas, N_part)
x2_moy = np.average(positions**2, axis = 1)
plt.figure()
plt.plot(x2_moy)
plt.plot(2 * D * t )
```

Ces instructions simples permettent de visualiser la croissance progressive de $< x^2 >$ au fil du temps. La courbe ci-contre a été obtenue avec 500 particules et 1000 pas de temps. On constante que $< x^2 >$ augmente linéairement au fil du temps, conformément à la loi d'échelle des phénomènes diffusifs. En trait plein, on a tracé la droite d'équation $< x^2 >= 2Dt$ exprimant le résultat établi par un calcul d'espérance. Avec un nombre de particules plus élevé, la moyenne empirique serait encore plus proche de l'espérance attendue.



I.4 Évolution de la répartition des particules

Le calcul de $< x^2(t) >$ donne une idée de l'étalement des particules, mais une description plus fine s'obtient en analysant leur répartition spatiale à t donné. Pour cela, on divise l'axe (Ox) en intervalles et on compte le nombre de particules que chacun contient. Il s'agit de construire un *histrogramme* des positions en les répartissant dans des *classes*, comme le montre la figure (2) sur un exemple. Pour ces calculs, nous utiliserons les deux outils suivants :

— la fonction `np.histogram(X, options)` qui calcule et renvoie l'effectif de chaque classe ;

— la fonction `plt.hist(X, options)` qui représente l'histogramme sans qu'il soit besoin de le calculer préalablement.

Ces fonctions admettent divers paramètres optionnels qui permettent de préciser le nombre, la largeur et la positions des classes de l'histogramme. Le paramètre booléen optionnel `density` permet de représenter soit des fréquences (des effectifs), soit des densités de probabilité. Dans la suite, nous utilisons des fréquences.

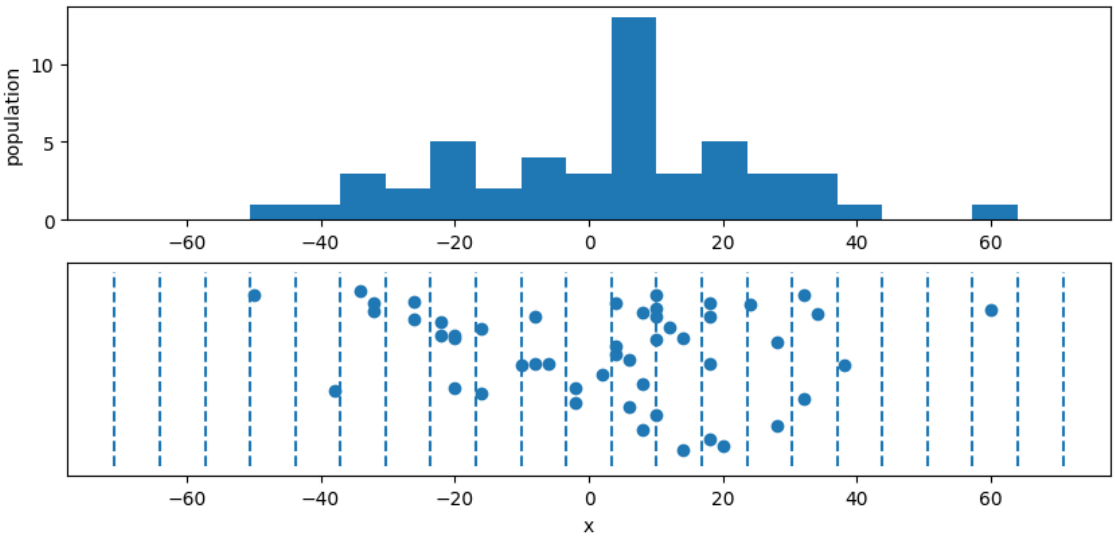


FIGURE 2 – Histogramme des abscisses d'un ensemble de 50 particules réparties en 21 classes de largeur $\Delta x = 6,7$. Pour une meilleure visualisation, les particules ont été dispersées selon la verticale mais en réalité, elles se déplacent seulement selon (Ox).

La fonction `distribution_particules` trace, calcule et renvoie l'histogramme des positions de l'ensemble des particules décrites par le tableau `position`, à un `instant` particulier. Les limites des classes sont ici calculées et spécifiées par `bin = bords_boites`, mais il serait possible de laisser `numpy` et `matplotlib` gérer cela automatiquement. Cette fonction calcule aussi l'abscisse du centre de chaque classe, ce qui s'avèrera utile pour comparer les fréquences obtenues à celles qu'on attend.

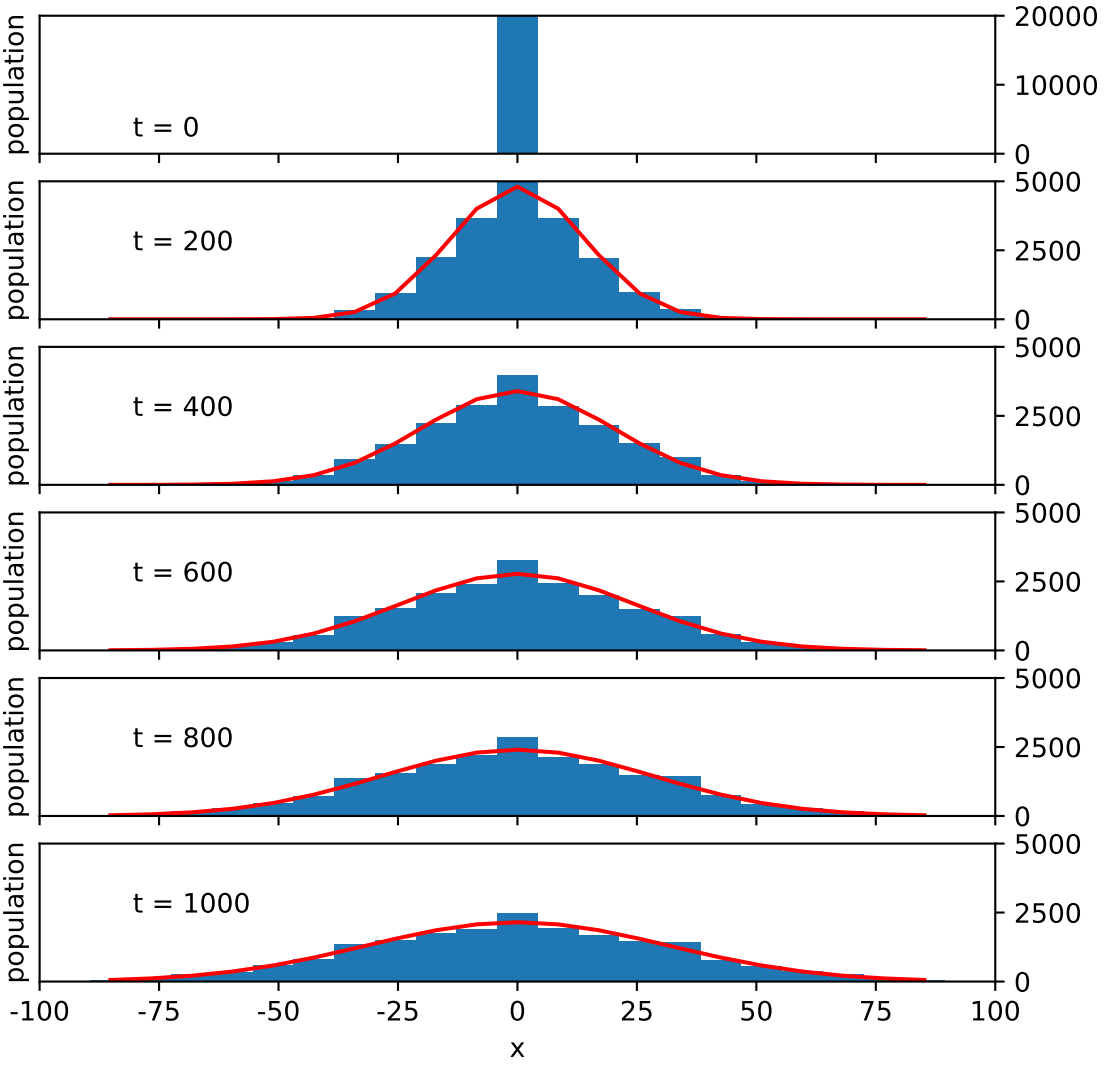
```
def distribution_particules(positions, instant, N_boites, x_min, x_max, numfig) :

    positions_instant = positions[instant, :]
    largeur_boites = (x_max - x_min) / N_boites

    plt.figure(numfig)
    plt.xlabel('x')
    plt.ylabel('population')
    bords_boites = [x_min + largeur_boites * i for i in range(N_boites+1)]
    plt.hist(positions_instant, bins = bords_boites)  # tracé automatique

    popu, bds_boites = np.histogram(positions_instant, bins = bords_boites,
        density = False)
    x_centres = (bds_boites[:-1] + bds_boites[1:])/2 # centres des boites

    return popu, x_centres
```

En appliquant itérativement cette fonction à divers instants, il est possible de suivre l'évolution de l'ensemble des particules. La figure ci-dessous montre le résultat obtenu pour $N_{\text{part}} = 20000$. On voit l'ensemble des particules s'étaler peu à peu, leur nombre restant constant.

L'équation de la diffusion prédit que la densité particulaire évolue selon

$$n(x,t) = \frac{N_{\text{part}}}{\sqrt{4\pi D t}}\, e^{-\frac{x^2}{4Dt}} \quad .$$

Dans ce problème unidimensionnel, $n(x,t)$ est un nombre de particules par unité de longueur, donc un segment de petite longueur Δx contient $n(x,t)\,\Delta x$ particules. En prenant pour x l'abscisse du centre d'une classe et pour Δx sa largeur, on peut donc prédire la population de chaque classe de l'histogramme. La courbe en trait plein représente $n(x,t)\,\Delta x$ et on constate qu'elle se confond, à quelques fluctuations près, avec les fréquences empiriques [1]. On confirme ainsi que l'équation de la diffusion exprime l'évolution d'un ensemble de particules effectuant des marches aléatoires.



II Cas bidimensionnel

Dans cette partie, vous devez compléter les codes à trous fournis par le professeur dans le fichier `marche_aleatoire_2d_eleve.py`.

II.1 Marche aléatoire individuelle

Partant de l'origine, la particule se déplace en effectuant une succession de N_{pas} pas unitaires aléatoires, chacun parallèle à l'un ou l'autre des vecteurs $\vec{u}_x$ ou $\vec{u}_y$. La trajectoire est typée comme un tableau numpy de

1. Ces fluctuations se réduisent en augmentant N_{part}. Par ailleurs, selon que t est pair ou impair, toutes les particules se trouvent sur des abscisses paires ou impaires, ce qui introduit des fluctuations artificielles liées à la position des frontières des classes.

taille $(N_{\text{pas}}+1)\times 2$, les entiers `trajectoire[i, 0]` et `trajectoire[i, 1]` désignant respectivement l'abscisse et l'ordonnée de la particule après le i^{e} pas. Les quatre déplacements équiprobables vers le nord, le sud, l'est et l'ouest sont stockés dans une variable globale `NSEW`.

```
NSEW = [np.array([0, 1]), np.array([-1, 0]), np.array([1, 0]) ,np.array([0, -1])]
```

Pour simuler l'aléa, on dispose de la méthode `rd.choice()`. Appelée sous la forme `rd.choice(L)`, où `L` est un itérable non vide, elle renvoie l'un des éléments de `L` aléatoirement choisi. En générant un à un les pas aléatoires et en calculant itérativement les positions successives de la particule, il est aisé d'obtenir la trajectoire (option 0 de la fonction `marche_alea_2d`).

Comme dans la partie I.1, on améliore l'efficacité en générant d'abord le tableau des déplacements puis en calculant les positions par `np.cumsum`. Ce tableau est ici de taille $N_{\text{pas}}\times 2$: `deplacements[i]` est l'un des vecteurs de `NSEW` ; `deplacements[i, 0]` et `deplacements[i, 1]` valent 0, 1 ou -1. Par exemple, 5 déplacements aléatoires sont décrits par le tableau ci-dessous.

```
[[ 0., -1.],
 [-1.,  0.],
 [ 1.,  0.],
 [ 0.,  1.],
 [-1.,  0.]]
```

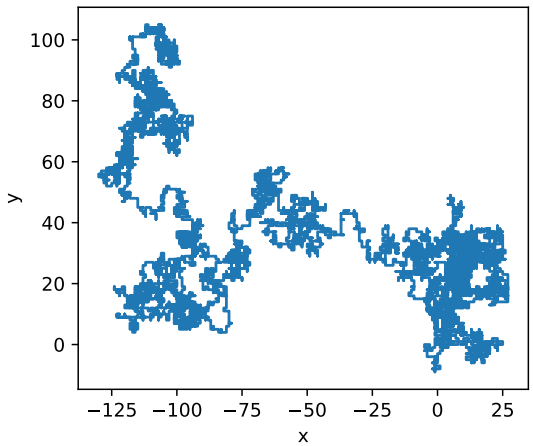
Pour obtenir les positions successives, il convient d'effectuer une somme cumulative en parcourant toutes *les lignes* pour chaque colonne. Cela s'obtient par `np.cumsum(tableau, axis = 0)`, le paramètre optionnel `axis = 0` signifiant qu'on somme sur le premier indice du tableau.

Pour produire ce tableau de N_{pas} déplacements, deux options sont possibles. On peut tout d'abord les obtenir un par un, itérativement, avec `rd.choice` (option 1 du code). Mais on gagne en efficacité en les générant *tous* d'un seul coup avec `np.random`, comme dans I.1 (option 2 du code). Il se trouve que `np.random` possède une méthode `choice` qui permet de générer un tableau d'éléments choisis aléatoirement dans un itérable.

```
In [15] : np.random.choice([1, 2, 3], size = 10)
Out[16]: array([2, 1, 3, 3, 3, 3, 2, 2, 2, 2])
```

Malheureusement, l'appel `np.random.choice(NSEW, size = N_pas)` produit un message d'erreur parce que `np.random.choice(L, size)` fonctionne seulement si les éléments de `L` ne sont pas eux-mêmes des itérables. Pour contourner cet écueil, on traite séparément les déplacements selon $\vec{u}_x$ et selon $\vec{u}_y$. Le déplacement selon $\vec{u}_x$ vaut 1, -1, ou 0, avec des probabilités 1/4, 1/4 et 1/2. Si le déplacement selon $\vec{u}_x$ est nul, celui selon $\vec{u}_y$ vaut ± 1, sinon il vaut 0. Le codage efficace de cette idée utilise la technique des *masques*, tableaux de booléens permettant de sélectionner certains éléments d'un autre tableau.

Compléter la fonction `marche_alea_2d(N_pas, option)` qui implémente ces trois options. Utiliser ce code pour produire quelques marches aléatoires, puis les représenter. Pour $N_{\text{pas}} = 10000$, comparer les temps de calcul des trois options.



II.2 Marche d'un ensemble de particules

Les particules, au nombre de N_{part}, se trouvent initialement à l'origine O des coordonnées, puis chacune d'elles effectue une marche aléatoire de N_{pas} pas comme dans le paragraphe précédent. On stocke les positions de toutes les particules à tous les instants dans un tableau `positions` de taille $(N_{\text{pas}}+1) \times N_{\text{part}} \times 2$. Ainsi, `positions[i, j]` est la position $(x_j(t_i), y_j(t_i))$ de la particule d'indice j à l'intant i ; `positions[i, j, 0]` est son abscisse et `positions[i, j, 1]` son ordonnée. La portion du tableau `positions[i]` est de taille $N_{\text{part}} \times 2$; elle contient les positions de toutes les particules à l'instant i. Au contraire, `positions[:,j]` est de taille $N_{\text{pas}} \times 2$; il contient les positions de la particule d'indice j à tous les instants, c'est à dire la trajectoire de cette particule.

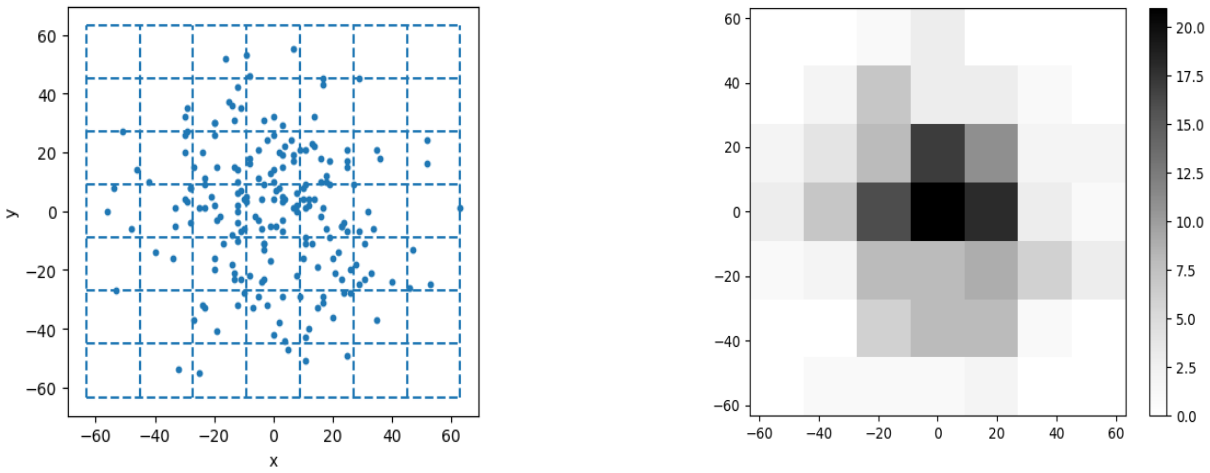
On obtient aisément ce tableau à trois indices en itérant la fonction `marche_alea` du paragraphe précédent. Compléter la fonction `mouvement_ensemble_2d(N_pas, N_part)`. L'utiliser conjointement à la fonction `dessine_ensemble` (fournie) pour observer l'étalement progressif d'un ensemble de 10000 particules pendant 1000 pas de temps.

II.3 Éloignement quadratique moyen

Après i pas de temps, la particule d'indice j se trouve au point de coordonnées $x_j(t_i), y_j(t_i)$ et le carré de sa distance à l'origine vaut $r_j^2(t_i) = x_j(t_i)^2 + y_j(t_i)^2$. Pour un ensemble de 10000 particules sur 1000 pas de temps, calculer le tableau de taille $(N_{\text{pas}}+1) \times N_{\text{part}}$ contentant les $r_j^2(t_i)$. Puis calculer le tableau de taille $(N_{\text{pas}}+1)$ contenant les moyennes d'ensemble $< r^2(t_i) >$ à chaque instant. Représenter l'évolution de $< r^2 >$ en fonction du temps et comparer au résultat issu d'un calcul d'espérance $< r^2 >= 4Dt$.

II.4 Évolution de la distribution des particules

Pour analyser finement la distribution des particules, on divise le plan (Oxy) en carrés et on compte le nombre de particules que chacun contient. Il s'agit de construire un histogramme bidimensionnel des positions et cette opération s'effectue commodément avec la fonction `np.hist2d(X, Y, options)`. Indépendamment du calcul de l'histrogramme, on peut le représenter par `plt.hist2d(X, Y, options)` qui produit une jolie image en couleur. Comme en dimension 1, ces fonctions admettent divers paramètres optionnels qui permettent de préciser le nombre, la largeur et la position des classes. Voici un exemple de résultat. L'échelle de couleur indique le nombre de particules dans chaque case.

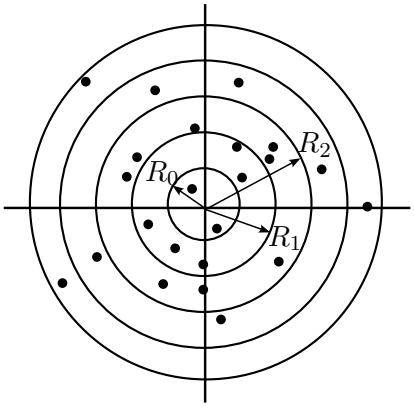


Compléter la fonction `distribution_particules_2d(positions, instant, N_boites, d_max, numfig)`. L'utiliser pour visualiser l'étalement progressif d'un ensemble de quelques milliers de particules sur quelques milliers de pas de temps. Il vous suffit pour cela de produire des images à quelques dates bien choisies.

II.5 Répartition radiale

Définissons la position d'une particule par ses coordonnées polaires (r, θ) plutôt que par ses coordonnées cartésiennes. Pour décrire l'éloignement progressif des particules depuis l'origine, seule la coordonnées

radiales r est pertinente et on peut passer θ sous silence. Traçons des cercles concentriques équidistants de rayons R_0, R_1, ... R_N et comptons le nombre de particules situées dans la couronne comprise entre les cercles de rayons R_k et R_{k+1}. On définit ainsi un histogramme qui renseigne sur la manière dont les particules se distribuent radialement. Cet histogramme se calcule aisément avec `np.histogram` et on le représente avec `plt.hist`. Nous laisserons Python déterminer automatiquement les rayons des cercles frontières (ils sont nommés `bords_r` dans le code), en spécifiant seulement le nombre de classes N_{boites} par l'option `bins = N_boites`. Compléter la fonction `repartition_radiale(positions, instant, N_classes, numfig)` qui affiche cet histogramme, calcule le nombre de particules dans chaque zone et le renvoie, avec le tableau des rayons frontières. On précise que la fonction `np.histogram` renvoie la population de chaque classe *et* ces rayons frontières.



Dans le modèle mésoscopique, la densité particulaire n solution de l'équation de la diffusion ne dépend que de r et de t à cause de l'invariance par rotation. Elle s'exprime par

$$n(r,t) = \frac{N_{\text{part}}}{4\pi Dt}\, e^{-r^2/(4Dt)}$$

Compléter la fonction `trace_popu_radiale_theorique(bords_r, instant)` qui trace la population théorique attendue dans chaque zone.

En utilisant ces fonctions, obtenir empiriquement la distribution radiale des particules à un instant particulier et la comparer à celle attendue. Vous devez obtenir un résultat analogue à celui représenté ci-dessous. En répétant cet affichage à diverses dates, vous pouvez observer l'étalement progressif des particules.