Simulation de Monte-Carlo d'un fluide

1. Introduction

En physique statistique, une simulation de Monte-Carlo d'un système consiste à calculer des grandeurs moyennes en générant un très grand nombre de configurations du système. Pour un système à température fixée, ces configurations obéissent à la distribution de probabilité de Boltzmann. L'objetif est d'aborder la méthode d'échantillonnage de Metropolis à travers l'exemple d'un fluide bidimensionnel.

2. Modèle de fluide

2.a. Configurations

On s'intéresse à un modèle de fluide bidimensionnel, constitué de molécules sphériques identiques se déplaçant sur un plan selon les lois du mouvement de la mécanique classique. La distribution statistique des vitesses des molécules est indépendante des forces qui s'exercent entre les molécules. Chaque molécule possède une énergie cinétique $e_c = \frac{1}{2}m(v_x^2 + v_y^2)$. D'après le théorème d'équipartition de l'énergie (voir cours de physique statistique), une molécule possède une énergie cinétique moyenne kT où k est la constante de Boltzmann. En conséquence, l'énergie cinétique moyenne du système est :

$$\overline{E_c} = NkT \tag{1}$$

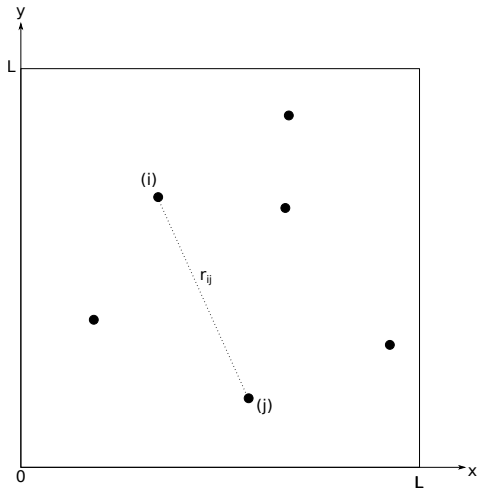
Dans le cas d'un gaz parfait, l'énergie potentielle d'interaction entre les molécules est nulle ; en conséquence, l'énergie du fluide se réduit à E_c.

L'interaction entre deux molécules dont la distance des centres est r est caractérisée par une énergie potentielle $U(r)$. Cette énergie ne dépend que de la distance en raison de l'hypothèse de molécules sphériques. Chaque paire (i, j) de molécules a donc une énergie $U(r_{ij})$ où r_{ij} est la distance entre ces deux molécules. L'énergie potentielle totale du système s'écrit comme une somme sur toutes les paires de molécules :

$$E_p = \sum_{(i,j)} U(r_{ij}) \tag{2}$$

Soient (x_i, y_i) les coordonnées du centre de la molécule i. Les N molécules se déplacent dans un volume fini, défini comme un carré de largeur L.

La configuration spatiale du système est définie par les coordonnées de ses N molécules. Elle peut donc être représentée dans un espace à $2N$ dimensions (appelé *espace des phases*). La figure suivante montre une configuration d'un système de $N = 6$ molécules.



En physique statistique, on s'intéresse aux probabilités des différentes configurations. Le système est supposé à l'équilibre thermique avec un thermostat de température T. La loi de Boltzmann (voir cours de physique statistique) exprime la probabilité d'une configuration sous la forme d'une densité de probabilité égale à :

$$p(x_0, x_1, \ldots, x_N, y_0, y_1, \ldots y_N) = \frac{1}{Z} \exp\left(-\frac{E_p}{kT}\right) \tag{3}$$

L'énergie potentielle doit être exprimée en fonction des coordonnées des N molécules et la densité de probabilité est donc une fonction de ces coordonnées. À chaque point de l'espace des phases, c'est-à-dire à chaque configuration, est associé une valeur de densité. La probabilité pour que le système soit dans une configuration située dans un volume élémentaire au voisinage d'une configuration donnée s'écrit :

$$dP = \frac{1}{Z} \exp\left(-\frac{E_p}{kT}\right) dx_0 dx_1 \ldots dx_N dy_0 dy_1 \ldots dy_N = \frac{1}{Z} \exp\left(-\frac{E_p}{kT}\right) dxdy \tag{4}$$

où $dxdy$ est une notation simplifiée pour le volume élémentaire. La constante Z permet d'assurer la condition de normalisation de la densité de probabilité :

$$\int \frac{1}{Z} \exp\left(-\frac{E_p}{kT}\right) dxdy = 1 \tag{5}$$

ce qui conduit à

$$Z(L,T) = \int \exp\left(-\frac{E_p}{kT}\right) dxdy \tag{6}$$

$Z(L,T)$ est la *fonction de partition*. Il s'agit d'une intégrale multiple dans un espace de dimension $2N$. Elle dépend du volume et de la température. Sachant que N peut être très grand, seule une méthode de Monte-Carlo permet d'évaluer cette intégrale.

La densité de probabilité permet de calculer la moyenne (c'est-à-dire l'espérance) d'une grandeur physique. En particulier, l'énergie potentielle moyenne est l'espérance de l'énergie potentielle, c'est-à-dire :

$$\overline{E_p} = \frac{1}{Z} \int E_p \exp\left(-\frac{E_p}{kT}\right) dxdy \tag{7}$$

On cherche à évaluer les intégrales (7) et (6) en utilisant la méthode de Monte-Carlo de calcul d'une intégrale.

L'énergie potentielle d'interaction est supposée nulle au delà d'une distance de coupure, qui correspond à la portée des interactions :

$$U(r) = 0 \text{ si } r \geq r_c \tag{8}$$

On remarque que le facteur exponentiel de la loi de Boltzmann ce calcule comme un produit sur les paires de molécules :

$$\exp\left(-\frac{E_p}{kT}\right) = \prod_{(i,j)} \exp\left(-\frac{U(r_{ij})}{kT}\right) \tag{9}$$

2.b. Modèle des sphères dures

L'exemple le plus simple de potentiel d'interaction est celui du modèle des *sphères dures*, qui consiste à supposer que les molécules ressentent une force répulsive infinie en dessous d'une distance r_c mais aucune force attractive. L'énergie d'interaction est définie par :

$$U(r) = \infty \text{ si } r < r_c$$
$$U(r) = 0 \text{ si } r \geq r_c$$

Dans ce cas, une configuration comportant deux sphères dont la distance est inférieure à r_c a une probabilité nulle. Le rayon d'une sphère est donc $r_c/2$. Par ailleurs, toutes les configurations ne présentant aucune paire dont la distance est inférieure à r_c sont équiprobables. Dans le modèle des sphères dures, la fonction de partition ne dépend pas de la température : $Z(L)$. Elle est égale à la probabilité qu'une configuration comporte aucune sphère en intersection avec une autre.

L'énergie potentielle de toutes les configurations de probabilité non nulle est nulle. En conséquence, l'énergie potentielle moyenne est nulle :

$$\overline{E_p} = 0 \tag{10}$$

En raison de sa simplicité, ce modèle a fait l'objet de nombreuses études théoriques et de simulations informatiques.

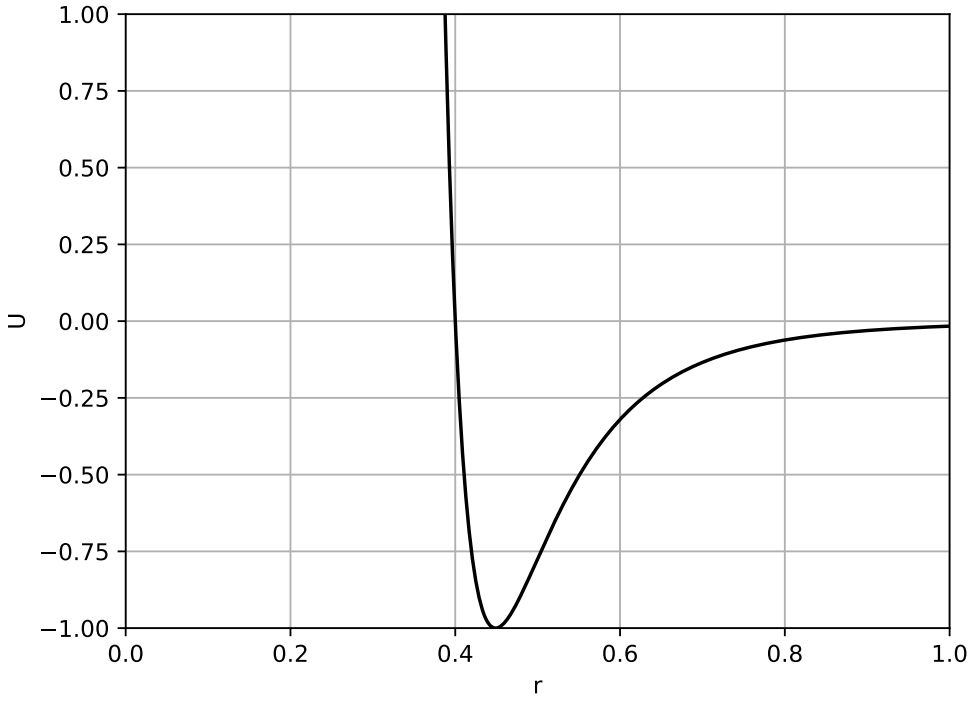
2.c. Potentiel d'interaction

Dans la réalité les molécules s'attirent par des forces de van der Waals. Voici un exemple de potentiel qui présente une partie attractive :

$$U(r) = 4\left(\left(\frac{b}{r}\right)^{12} - \left(\frac{b}{r}\right)^{6}\right) \text{ si } r < r_c$$
$$U(r) = 0 \text{ si } r \geq r_c$$

On choisit $r_c = 1$. Voici le potentiel pour $b = 0.4$:

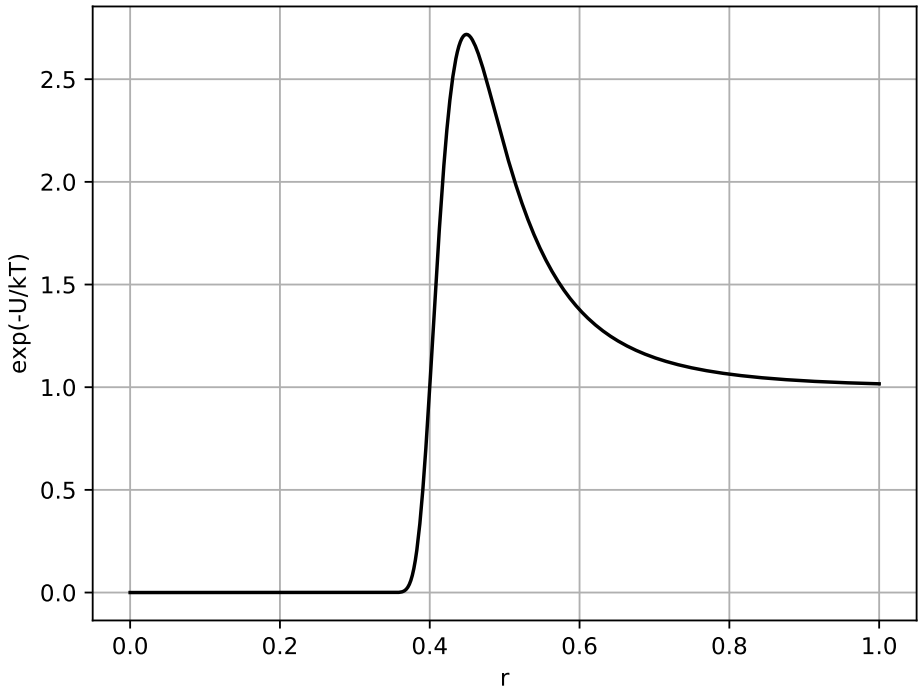
```
from matplotlib.pyplot import *
import numpy

P=500
rc=1
r=numpy.linspace(0,1,P)
b=0.4
U=4*(numpy.power(r/b,-12)-numpy.power(r/b,-6))
figure()
plot(r,U,"k-")
xlabel("r")
ylabel("U")
xlim(0,1)
ylim(-1,1)
grid()
```



La force est répulsive lorsque le potentiel est décroissant, attractive lorsqu'il est croissant. Voici le facteur de Boltzmann relatif à ce potentiel, c'est-à-dire $\exp(-U(r)/kT)$, pour $kT = 1$:

```
exp_U = numpy.zeros(P)
for i in range(1,P):
    if U[i]<500:
        exp_U[i] = numpy.exp(-U[i])
figure()
plot(r,exp_U,"k-")
```

```
xlabel("r")
ylabel("exp(-U/kT)")
grid()
```



Ce facteur exponentiel sera précalculé pour une température donnée et stocké dans un tableau `exp_U`. Voici comment se fera le calcul du facteur exponentiel pour une distance quelconque :

```
def calcul_exp_U(r):
    if r >= 1:
        return 1
    else:
        return exp_U[int(r*(P-1))]
```

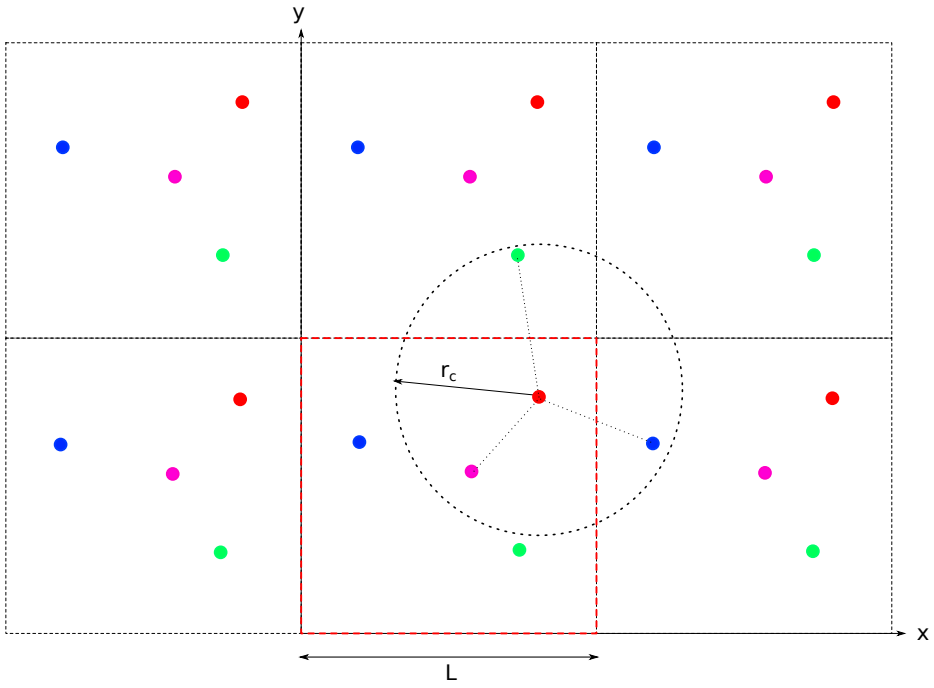
```
print(calcul_exp_U(0.4))
--> 0.92845033010441247
```

L'utilisation d'une table à P éléments revient à approcher la fonction par une fonction constante par morceaux.

Le facteur de Boltzmann du système de N molécules est égal au produit des facteurs exponentiels des paires de molécules.

2.d. Conditions limites périodiques

Afin de simuler un domaine dans lequel les effets de bords sont négligeables, on adopte des conditions limites périodiques. Cette méthode consiste à considérer que le carré se répète périodiquement dans les deux directions, comme le montre la figure suivante.



Le nombre de molécules est $N = 4$. Chaque molécule interagit avec les molécules du carré mais aussi avec toutes les répliques périodiques. Cependant, l'interaction est nulle au delà d'une distance r_c. En choisissant L assez grand pour que $L > 2r_c$, une molécule (i) peut interagir avec une molécule (j) située dans le carré ou bien avec sa réplique dans un carré voisin, mais pas avec les deux. Cette propriété est illustrée sur la figure ci-dessus pour la molécule représentée en rouge, avec un cercle de rayon $r_c = L/2$ contenant les molécules avec lesquelles elle interagit.

Voyons comment calculer la distance entre deux molécules (i) et (j) compte tenu des conditions limites périodiques, dans le but de calculer l'énergie potentielle d'interaction $U(r_{ij})$ et le facteur exponentiel associé.

- ▷ Calculer $X = x_j - x_i$ et $Y = y_j - y_i$.
- ▷ Si $X > L/2$ remplacer X par $X - L$.
- ▷ Si $X < -L/2$ remplacer X par $X + L$.
- ▷ Si $|X| >= r_c$ alors $U = 0$ et on s'arrête.
- ▷ Si $Y > L/2$ remplacer Y par $Y - L$.
- ▷ Si $Y < -L/2$ remplacer Y par $Y + L$.
- ▷ Si $|Y| >= r_c$ alors $U = 0$ et on s'arrête.
- ▷ Calculer $r = \sqrt{X^2 + Y^2}$.
- ▷ Déterminer $U(r)$ à l'aide de la table `U` puis $exp(-U/(kT))$ à l'aide de la table `exp_U`.

On adoptera la valeur $r_c = 1$. La largeur du carré doit donc être supérieure à 2.

3. Méthodes de Monte-Carlo

3.a. Méthode directe

L'objectif est de calculer l'énergie potentielle moyenne (7). Il faut donc évaluer les deux intégrales suivantes :

$$Z = \int \exp\left(-\frac{E_p}{kT}\right) dxdy \tag{11}$$

$$Y = \int E_p \exp\left(-\frac{E_p}{kT}\right) dxdy \tag{12}$$

puis calculer $\overline{E_p} = Y/Z$.

Le calcul de ces intégrales par la méthode de Monte-Carlo consiste (dans sa forme élémentaire) à tirer aléatoirement un nombre Q de configurations avec une distribution de probabilité uniforme. Si l'on note E_q l'énergie potentielle de la configuration q, les intégrales sont estimées par les moyennes suivantes :

$$Z \approx \frac{1}{L^{2N}} \frac{1}{Q} \sum_{q=0}^{Q-1} \exp\left(-\frac{E_q}{kT}\right) \tag{13}$$

$$Y \approx \frac{1}{L^{2N}} \frac{1}{Q} \sum_{q=0}^{Q-1} E_q \exp\left(-\frac{E_q}{kT}\right) \tag{14}$$

Le tirage aléatoire des configurations avec une densité de probabilité uniforme se fait simplement en tirant les $2N$ coordonnées des molécules dans l'intervalle $[0, L]$ avec une densité uniforme.

Cette méthode d'échantillonnage direct est en réalité extrêmement inefficace car elle génère un grand nombre de configurations dont la probabilité est très faible, en raison de la forme exponentielle de la loi de probabilité. Ce problème est d'autant plus marqué que la densité du fluide est grande, car un tirage aléatoire uniforme a alors de bonnes chances de donner au moins deux molécules avec une distance trop faible. Par ailleurs, la présence d'un seul terme de très grande énergie dans la somme des énergies potentielles conduit, en raison de la précision limitée des calculs, à une perte d'information sur les termes de faible énergie. Ce dernier problème peut être résolu en stockant les termes de la somme dans un tableau avant de le trier et de calculer la somme par ordre croissant des termes.

Pour améliorer la méthode, il faudrait effectuer un échantillonnage préférentiel, c'est-à-dire générer les configurations avec une densité de probabilité qui favorise les configurations dont le facteur de Boltzmann est plus grand. C'est ce que fait la méthode décrite ci-après.

3.b. Méthode de Metropolis

La méthode de Metropolis (du nom d'un de ses inventeurs) est une méthode consistant à générer une chaîne de Markov de configurations. Une chaîne de Markov est une suite de configurations dont chacune est générée aléatoirement à partir de la précédente. Dans le cas présent, il s'agit de générer une suite de configurations telle que l'ensemble de ces configurations obéisse à la loi de probabilité suivante :

$$p(x_0, x_1, \ldots, x_N, y_0, y_1, \ldots y_N) = \frac{1}{Z} \exp\left(-\frac{E_p}{kT}\right) \tag{15}$$

Notons E_q les énergies potentielles des Q configurations générées. La moyenne de l'énergie potentielle se calcule par :

$$\overline{E_p} \approx \frac{1}{Q} \sum_{q=0}^{Q} E_q \tag{16}$$

La méthode de Metropolis donne les règles de transition d'une configuration (q) à la suivante $(q+1)$. Cette méthode ne nécessite pas la connaissance du facteur de normalisation $1/Z$. Voici l'algorithme de Metropolis :

- ▷ Choisir une molécule aléatoirement parmi les N molécules. Soit (i) l'indice de cette molécule.
- ▷ Proposer un déplacement de cette molécule dans un carré de largeur H, dont la valeur est fixée au début. Le déplacement est aléatoire avec une densité uniforme dans le carré.
- ▷ Calculer la variation d'énergie potentielle du système qui serait causée par ce déplacement : ΔE_p.
- ▷ Si $\Delta E_p \leq 0$ alors la configuration proposée est plus probable (ou autant) que la configuration présente. Elle est donc acceptée comme configuration $q+1$.
- ▷ Si $\Delta E_p > 0$ la configuration proposée est acceptée avec la probabilité $\exp\left(-\frac{\Delta E_p}{kT}\right)$. Pour ce faire, on tire un nombre aléatoire u avec une densité de probabilité uniforme sur l'intervalle $[0, 1]$. Si le facteur exponentiel ci-dessus est supérieur à u alors la configuration proposée est acceptée comme configuration $q+1$. Sinon, la nouvelle configuration est identique à la précédente.

La largeur du carré de déplacement (H) doit être ajustée afin que le taux de déplacements acceptés soit voisin de 0,5.

Il faut aussi choisir une condition initiale, de préférence une condition dont la probabilité est élevée. On peut par exemple répartir régulièrement les molécules sur le domaine carré. Il a été démontré que la suite de configurations générée selon cet algorithme obéit effectivement à la densité de probabilité (15) mais seulement après un certain nombre de configurations générées. Il faut donc d'abord générer un certain nombre de configurations avant de calculer la moyenne avec la somme (16). Cette étape est appelée *mise à l'équilibre* ; elle est d'autant plus longue que la probabilité de la configuration initiale est faible.

Voyons en détail comment calculer la variation d'énergie potentielle ΔE_p associée au déplacement de la molécule (i). L'énergie potentielle E_q et le facteur de Boltzmann associé $exp(-E_q/(kT))$ sont mémorisés dans deux variables nommées `Ep` et `exp_Ep`. Il est aussi intéressant de mémoriser les énergies d'interaction $U(r_{ij})$ et le facteur exponentiel $exp(-U/(kT))$ pour chaque paire de molécules. Pour cela, on utilisera deux tableaux carrés `U_paires` et `exp_U_paires`. L'énergie d'interaction d'une paire (i, j) est mémorisée dans `U_paires[i,j]` et dans `U_paires[j,i]` (deux éléments identiques pour une paire). Voici l'algorithme de calcul de la variation d'énergie potentielle :

- ▷ Pour chaque molécule j (différente de i), calculer l'énergie d'interaction $U(r_{ij})$ correspondant à la nouvelle position de la molécule i et placer le résultat dans

un tableau `U_paires_essai`. Calculer aussi les termes exponentiels et les placer dans un tableau `exp_U_paires_essai`.

- ▷ Calculer la variation d'énergie ΔE_p égale à la somme pour les paires de la nouvelle énergie moins l'ancienne.
- ▷ Calculer le facteur exponentiel $exp(-\Delta E_p/(kT))$ égal au produit des facteurs exponentiels pour la nouvelle configuration divisé par le produit des facteurs pour l'ancienne.
- ▷ Si la nouvelle configuration est acceptée, mettre à jour les variables `Ep` et `exp_Ep` ainsi que les tableaux `U_paires` et `exp_U_paires`.

4. Implémentation

4.a. Prototype

Les données et les fonctions sont encapsulées dans une classe dont voici le prototype : Fluide2D.py

```
class Fluide2D:
    def __init__(self,L,H,N,T):
        self.L = L
        self.H = H
        self.N = N
        self.T = T
        self.P = 500
        self.rc = 1
        self.e = L/(numpy.sqrt(N)+1)
        b = 0.4
        densite = N*(numpy.pi*(b/2)**2)/L**2
        print("Densité = %f"%densite)
        self.r = numpy.linspace(1e-10,1,self.P)
        self.U = 4*(numpy.power(r/b,-12)-numpy.power(r/b,-6))
        self.exp_U = numpy.zeros(self.P)
        for i in range(1,self.P):
            if self.U[i]/T < 500:
                self.exp_U[i] = numpy.exp(-self.U[i]/T)
        self.x = numpy.zeros(self.N)
        self.y = numpy.zeros(self.N)
        self.Ep = 0
        self.exp_Ep = 0
        self.U_paires = numpy.zeros((N,N))
        self.exp_U_paires = numpy.zeros((N,N))
        self.U_paires_essai = numpy.zeros((N,N))
        self.exp_U_paires_essai = numpy.zeros((N,N))

    def calcul_U(self,r):
       return self.U[int(r*self.P)]

    def calcul_exp_U(self,r):
```

```
        return self.exp_U[int(r*self.P)]

    def distance(self,x1,y1,x2,y2):
        # a completer
        return (valide,r)

    def initialiser_energies(self):
        # a completer
        pass

    def configuration_aleatoire(self):
        self.x = numpy.random.random(self.N)*self.L
        self.y = numpy.random.random(self.N)*self.L
        self.initialiser_energies()

    def configuration_initiale(self):
        x = self.e
        y = self.e
        for m in range(self.N):
            self.x[m] = x
            self.y[m] = y
            x += self.e
            if x >= self.L:
                y += self.e
                x = self.e
        self.initialiser_energies()

    def energie_moyenne_direct(self,Q):
        # a completer
        return Ep_moy

    def variation_energie(self,i,x,y):
        # a completer
        return (delta_Ep,exp_delta_Ep)

    def pas_metropolis(self):
        i = random.randint(0,self.N-1)
        # a completer
        return accept

    def energie_moyenne_metropolis(self,Q):
        # a completer
        return Ep_moy
```

Le constructeur définit les variables qui seront utilisées. Les arguments sont la taille du domaine carré, la taille du carré de déplacement, le nombre de molécules et la température. La constante de Boltzmann est égale à 1, ce qui fait que la température est en fait l'énergie kT.

Les fonctions `calcul_U` et `calcul_exp_U` renvoient l'énergie potentielle d'interaction pour une distance donnée et le facteur exponentiel correspondant. La valeur de `r` est supposée srictement inférieure à r_c.

La fonction `distance` calcule la distance entre deux molécules dont les coordonnées sont données, en tenant compte des conditions limites périodiques, et effectue un test pour savoir si elle est strictement inférieure à r_c. Si `valide=True` la distance est strictement inférieure à r_c. Dans le cas contraire, sa valeur ne sera pas utilisée.

La fonction `initialiser_energies` initialise les variables contenant les énergies pour la configuration en cours.

La fonction `configuration_aleatoire` génère une configuration aléatoire.

La fonction `configuration_initiale` génère une configuration ordonnée, pour la méthode de Metropolis.

La fonction `energie_moyenne_direct` calculer l'énergie potentielle moyenne avec la méthode d'échantillonnage direct. Cette fonction appelle Q fois la fonction `configuration_aleatoire`.

La fonction `variation_energie` calcule la variation d'énergie (et le terme exponentiel) correspondant au déplacement de la molécule d'indice `i` dont la nouvelle position est donnée par `x,y`.

La fonction `pas_metropolis` effectue un pas élémentaire de la méthode de Metropolis, c'est-à-dire le choix aléatoire d'une molécule et le déplacement éventuel de cette molécule. L'énergie du système doit être mise à jour. La fonction renvoie `True` si le déplacement proposé est accepté, `False` si la nouvelle configuration est identique à la précédente.

La fonction `energie_moyenne_metropolis` effectue le calcul de l'énergie potentielle moyenne. Pour cela, elle appelle `Q` fois la fonction `pas_metropolis` tout en calculant la moyenne par la somme (16). Il est supposé que la fonction `configuration_initiale` a été appelée au préalable. La fonction calcule et affiche le taux de déplacements acceptés.

4.b. Travaux pratiques

Implémenter les fonctions de la classe `Fluide2D`.

Tester avec $N = 10$, $L = 10$, $H = 0{,}1$ et $T = 0{,}1$. Cette dernière valeur signifie que l'énergie kT est égale à un dixième de la valeur du minimum de U.

On fera une dizaine de calculs pour observer qualitativement la variance du résultat, avec le code suivant, qui affiche l'énergie potentielle moyenne par molécule :

```
L=10
H=0.1
N=10
T=0.1
fluide = Fluide2D(L,H,N,T)
print("Echantillonnage direct")
for k in range(10):
    Ep = fluide.energie_moyenne_direct(10000)
    print(Ep/N)
print("Echantillonnage de Metropolis")
fluide.configuration_initiale()
for k in range(10):
    Ep = fluide.energie_moyenne_metropolis(100000)
    print(Ep/N)
```

Fluide2D-solution.py