Rapport Projet Mathématiques et Informatique

Jeux de balles

Xavier Léonhart, Nathan Louvet, Charles Lupo, Timothée Mounier

Table des matières

1 Contextualisation du problème

Nous étudions le mouvement d'une particule se déplaçant à l'intérieur d'une boîte. L'objectif étant de compter le nombre de rebonds sur les parois et de savoir par quelle ouverture la particule s'échappe. **Le projet finale étant de réaliser le jeu du casse briques**.

2 Mise en pratique et implémentation informatique

2.1 Problème : Boîte avec une ouverture

2.1.1 Explication du problème

On suppose une ouverture sur la paroi de droite de la boîte de la figure 1.

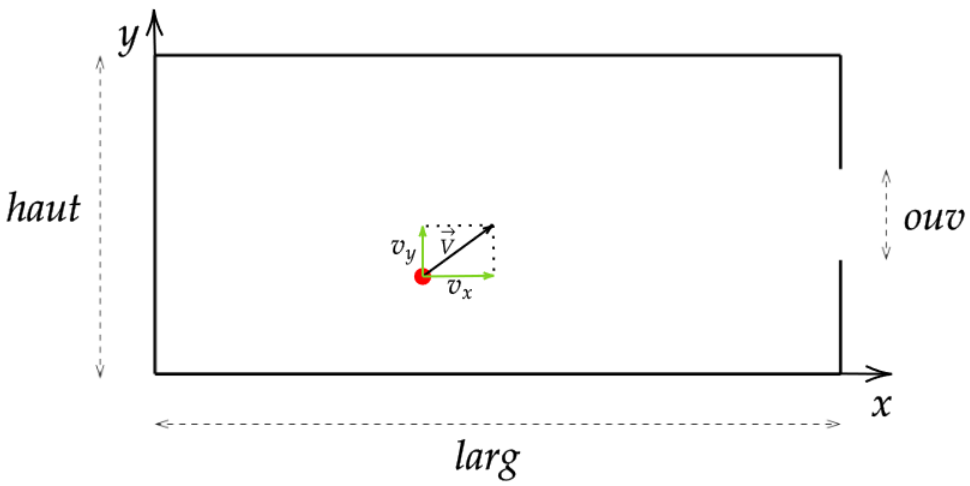


FIGURE 1: Géométrie du problème I

Hypothèses :

- on néglige tous les frottements
- la particule est soumise à aucune force (la résultante normale est compensé par le poids)
- on suppose que les chocs contre les parois sont élastiques

L'équation de la trajectoire de la particule entre deux chocs est donné par le Principe Fondamental de la Dynamique. Avec les hypothèses données précédemment cela correspond à une équation de droite du type $y = ax + b$. Ainsi, il nous suffit de tracer la droite et de chercher les points d'intersections entre la droite et les parois de la boîte figure 2 .

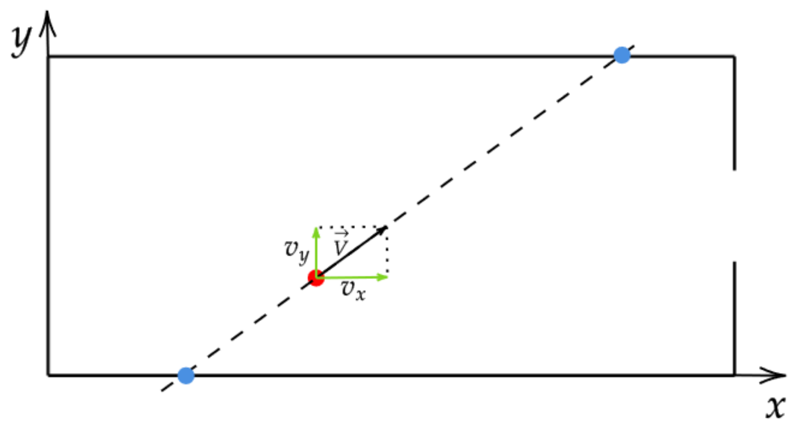


FIGURE 2: Géométrie du problème I

On obtient donc deux points d'intersections mais seulement un correspond à un choc. Pour le déterminer, on s'intéresse aux composantes de la vitesse. Le signe v_x et v_y nous permet de connaître la direction de la vitesse. On peut donc écrire $a = \frac{v_y}{v_x}$ et $b = y_0 - \frac{v_y}{v_x}x_0$ avec (x_0, y_0) un point quelconque de la droite.

Une fois les intersections trouvées, nous devons modéliser le rebond de la particule. On utilise une loi analogue à celui de Snell-Descartes : $|\theta_i| = |\theta_r|$

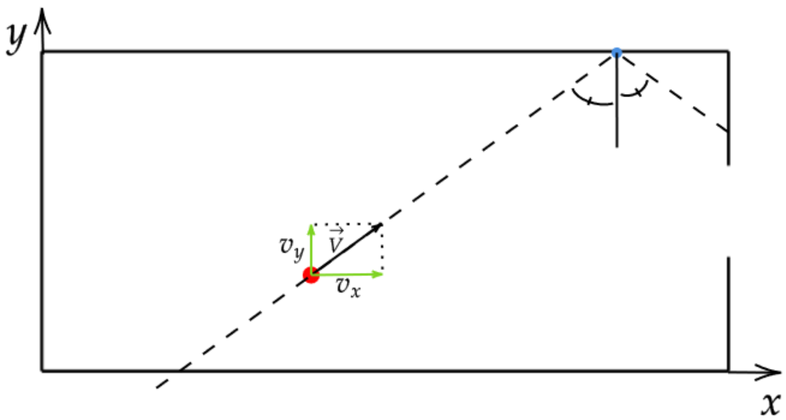


FIGURE 3: Géométrie du problème I

On réitère le même processus choc après choc jusqu'à obtenir un point d'intersection entre la trajectoire de la particule et le segment de l'ouverture figure 4.

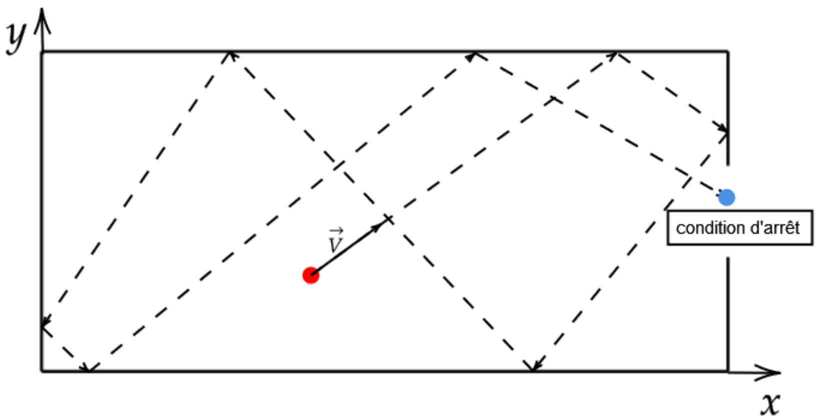


FIGURE 4: Géométrie du problème I

2.1.2 Implémentation informatique

Pour toute la suite du problème, nous travaillons avec les modules :

```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from math import *
```

Routines intersections

Dans un premier temps, nous avons établie des routines dans le cas générale pour pouvoir les réutiliser dans les autres problèmes. On a donc établie deux programmes pour détecter les points d'intersections entre la trajectoire de la particule et les parois de la boîte.

```python
def intersection_segment_droite(a1, b1, a2, b2, point1, point2, x, y, vx, vy):
    """prend en argument le coeff directeur, l'ordonner à l'origine de deux droites ainsi que
    les point d'extremité du segment et renvoie True s'ils s'intersect"""
```

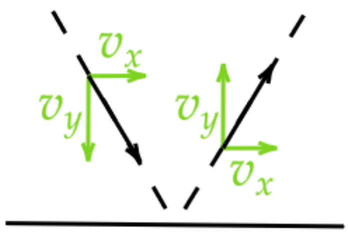
```python
    intersec = (b2 - b1) / a1 - a2
    if vx < 0 and vy > 0:
        return intersec > point1[0] and intersec < point2[0] and y < b2
    elif vx > 0 and vy > 0:
        return intersec > point1[0] and intersec < point2[0] and y < b2
    elif vx > 0 and vy < 0:
        return intersec > point1[0] and intersec < point2[0] and y > b2
    else :
        return intersec > point1[0] and intersec < point2[0] and y > b2

def intersection_segmentVerticale_droite(a, b, x, point1, point2, vx, vy):
    """prend en argument le coeff directeur, l'ordonner à l'origine de deux droites ainsi que les point d'extremité du segment et renvoie True s'ils s'intersect"""
    intersec = a * x + b
    if vx < 0 and vy > 0:
        return intersec > point1[1] and intersec < point2[1] and x == 0
    elif vx < 0 and vy < 0:
        return intersec > point1[1] and intersec < point2[1] and x == 0
    elif vx > 0 and vy > 0:
        return intersec > point1[1] and intersec < point2[1] and x == larg
    else:
        return intersec > point1[1] and intersec < point2[1] and x == larg
```

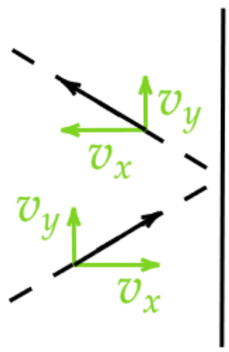
Les routines intersections vont être utilisé tout au long du sujet.

Le rebond

Le changement des composante de la vitesse dépend de la parois.



(a) Sur la face inférieur



(b) Sur la face de droite

FIGURE 5: Illustration des composante avec un rebond

Pour le cas (5a) on remarque $v_x^{reflechie} = v_x^{incident}$ et $v_y^{reflechie} = -v_y^{incident}$. Et pour le cas (5b) on remarque $v_x^{reflechie} = -v_x^{incident}$ et $v_y^{reflechie} = v_y^{incident}$. On procède de la même manière pour la face du haut de droite.

Ainsi, on peut donc écrire le programme :

```python
def trouver_intersec_composante(larg, haut, ouv, X, Y, vx, vy, FLAG):
    """prend un argument les dimensions de la boîte, la position  des chocs entre la particule et les parois, les composantes de la vitesse et return les nouvelles composante de la vitesse après le rebond, les listes X et Y sont actualisées avec les coordonnées du nouveau choc"""
    a = vy / vx            #coefficient directeur
    b = Y[-1] - a * X[-1]   #ordonnée à l'origine
    #pour  la face inférieur
    if intersection_segment_droite(a, b, 0, 0, [0, 0], [larg, 0], X[-1], Y[-1], vx, vy):
        vy = -vy
        X.append(-b / a)
        Y.append(0)
    #pour la face supérieur
    elif intersection_segment_droite(a, b, 0, haut, [0, haut], [larg, haut], X[-1], Y[-1], vx, vy):
        vy = -vy
        X.append((haut - b) / a)
```

```python
        Y.append(haut)
    #pour la face gauche
    elif  intersection_segmentVerticale_droite(a, b, 0, [0, 0], [0, haut], vx, vy):
        vx = -vx
        X.append(0)
        Y.append(b)
    #pour la face droite
    elif intersection_segmentVerticale_droite(a, b, larg, [larg, 0], [larg, (haut - ouv) / 2], vx, vy) or intersection_segmentVerticale_droite(a, b, larg, [larg, (haut + ouv) / 2], [larg, haut], vx, vy):
        vx = -vx
        X.append(larg)
        Y.append(a * larg + b)
    #si on est dans les coins
    elif abs(vx) == abs(vy) and (b == 0 or b == haut and b == haut - a * larg and b == -a * larg):
        vx = -vx
        vy = -vy
        if b == 0 or b == haut:             #coins haut et bas gauche
            X.append(0)
            Y.append(b)
        elif b == haut - a * larg:          #coin haut droit
            X.append(larg)
            Y.append(haut)
        else:                               #coin bas droite
            X.append(larg)
            Y.append(0)
    #pour l'ouverture
    else :
        X.append(larg)
        Y.append(a * larg + b)
        FLAG = 1
    return vx, vy, FLAG
```

Trajectoire

On utilise une variable $FLAG$ qui prend la valeur 1 quand la particule a atteint l'ouverture sinon $FLAG = 0$. On peut donc écrire un nouveau programme qui nous permettra d'avoir toutes les trajectoires de la particule :

```python
def trajectoire(larg, haut, ouv, X, Y, vx, vy):
    """prend en argument, les dimension de la boîte, la liste des position de la particule et les composantes de la vitesse et return le nombre de choc entre la particule et les parois et deux listes qui permette d'avoir les toutes les position de la particule pendant la simulation"""
    FLAG = 0
    count = 0
    while FLAG != 1:
        vx, vy, FLAG = trouver_intersec_composante(larg, haut, ouv, X, Y, vx, vy, FLAG)
        count += 1
    return (count - 1), X, Y
```

On peut donc exécuter le programme et on obtient :

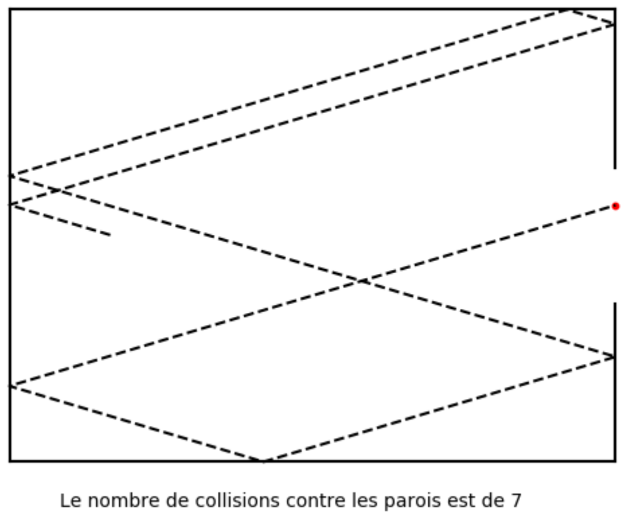


FIGURE 6: Mouvement de la particule dans la boîte

2.2 Problème : Le billard

2.2.1 Explication du problème

Ce problème est identique au problème numéro 1. On effectue juste un changement de géométrie pour la boite. L'ouverture disparaît pour laisser place un δ à chaque coins figure 7.

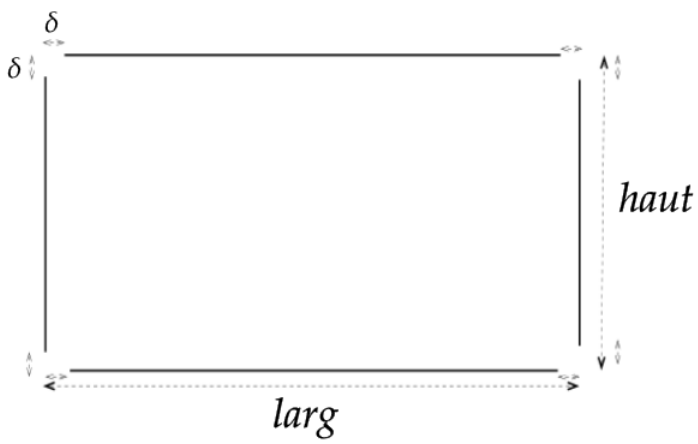


FIGURE 7: Géométrie du problème II

2.2.2 Implémentation informatique

On réutilise la routine trouve_intersec_composante que l'on adapte pour le billard :

```
def trouver_intersec_composante_billard(larg, haut, delta, X, Y, vx, vy, FLAG):
    """prend un argument les dimensions de la boîte, la position  des chocs entre la particule et les parois, les composantes de la vitesse et return les nouvelles composante de la vitesse après le rebond, les listes X et Y sont actualisées avec les coordonnées du nouveau choc"""
    a = vy / vx              #coefficient directeur
    b = Y[-1] - a * X[-1]    #ordonnée à l'origine
    #pour  la face inférieur
    if intersection_segment_droite(a, b, 0, 0, [0 + delta, 0], [larg - delta, 0], X[-1], Y[-1], vx, vy):
        vy = -vy
        X.append(-b / a)
        Y.append(0)
```

```python
    #pour la face supérieur
    elif intersection_segment_droite(a, b, 0, haut, [0 + delta, haut], [larg - delta, haut], X[-1], Y[-1], vx, vy):
        vy = -vy
        X.append((haut - b) / a)
        Y.append(haut)
    #pour la face gauche
    elif  intersection_segmentVerticale_droite(a, b, 0, [0, 0 + delta], [0, haut - delta], vx, vy):
        vx = -vx
        X.append(0)
        Y.append(b)
    #pour la face droite
    elif intersection_segmentVerticale_droite(a, b, larg, [larg, 0 + delta], [larg, haut - delta], vx, vy):
        vx = -vx
        X.append(larg)
        Y.append(a * larg + b)
    #dans les coins
    else:
        if abs(vx) == abs(vy) and (b == 0 or b == haut and b == haut - a * larg and b == -a * larg):
            vx = -vx
            vy = -vy
            if b == 0 or b == haut:     #coins haut et bas gauche
                X.append(0)
                Y.append(b)
            elif b == haut - a * larg:  #coin haut droit
                X.append(larg)
                Y.append(haut)
            else:                       #coin bas droite
                X.append(larg)
                Y.append(0)
        #face inférieur dans la zone delta
        elif intersection_segment_droite(a, b, 0, 0, [0, 0], [delta, 0], X[-1], Y[-1], vx, vy) or intersection_segment_droite(a, b, 0, 0, [larg - delta, 0], [larg, 0], X[-1], Y[-1], vx, vy):
            X.append(-b / a)
            Y.append(0)
        #face supérieur dans la zone delta
        elif intersection_segment_droite(a, b, 0, haut, [0, haut], [delta, haut], X[-1], Y[-1], vx, vy) or intersection_segment_droite(a, b, 0, haut, [larg - delta, haut], [larg, haut], X[-1], Y[-1], vx, vy):
            X.append((haut - b) / a)
            Y.append(haut)
        #face de gauche dans la zone delta
        elif intersection_segmentVerticale_droite(a, b, 0, [0, 0], [0, delta], vx, vy) or intersection_segmentVerticale_droite(a, b, 0, [0, haut - delta], [0, haut], vx, vy):
            X.append(0)
            Y.append(b)
        #face de droite dans la zone delta
        else:
            X.append(larg)
            Y.append(a * larg + b)
        FLAG = 1
    return vx, vy, FLAG
```

On peut donc exécuter le programme et on obtient :

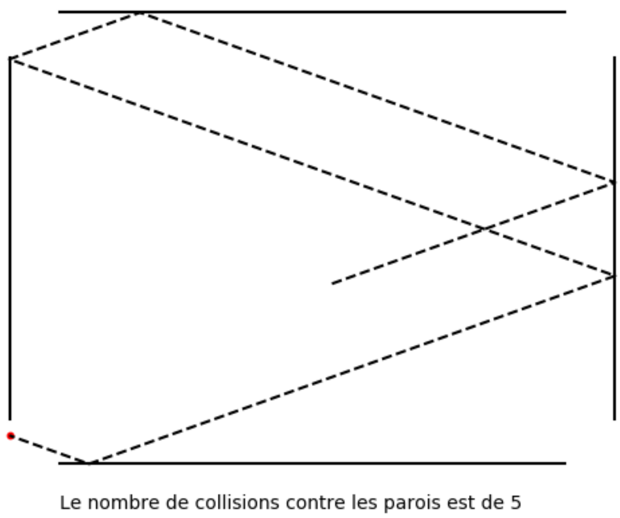


Figure 8: Movement de la particule dans le billard

2.3 Problème : Boîte avec cercle au centre

2.3.1 Explication du problème

Pour ce problème, nous reprenons la géométrie du problème I et nous ajoutons un cercle au centre de la boîte figure 9.

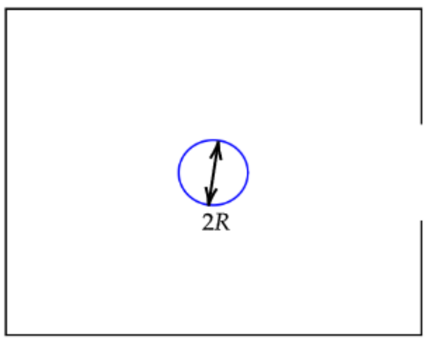


Figure 9: Géométrie du problème III

Dans un premier temps, nous avons chercher les points d'intersections entre la trajectoire de la particule et le cercle. On note (x_c, y_c) les coordonnées du cercle :

$$\begin{cases} R^2 &= (x - x_c)^2 + (y - y_c)^2 \\ y &= ax + b \end{cases} \iff \begin{cases} 0 &= Ax^2 + Bx + C \\ y &= ax + b \end{cases}$$

avec $A = (a^2 + 1) \quad B = 2(ab - ay_c - x_c) \quad C = x_c^2 + y_c^2 + b^2 - 2by_c - R^2 \quad \Delta = B^2 - 4AC$

On obtient donc deux couples de solutions :

$$\mathcal{S} = \left\{ \left(\frac{-B - \sqrt{\Delta}}{2A}, a\frac{-B - \sqrt{\Delta}}{2A} + b \right), \left(\frac{-B + \sqrt{\Delta}}{2A}, a\frac{-B + \sqrt{\Delta}}{2A} + b \right) \right\}$$

Il suffit d'éliminer une des deux solutions en évaluant les distances à la particule :

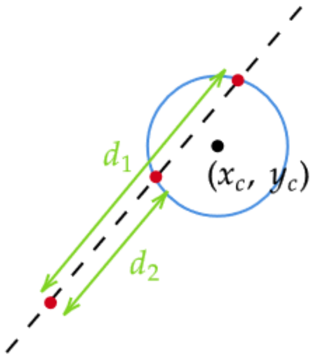


FIGURE 10: Tracé de la situation

Après avoir détecté les points d'intersections, nous devons déterminer les nouvelles composantes après le rebond

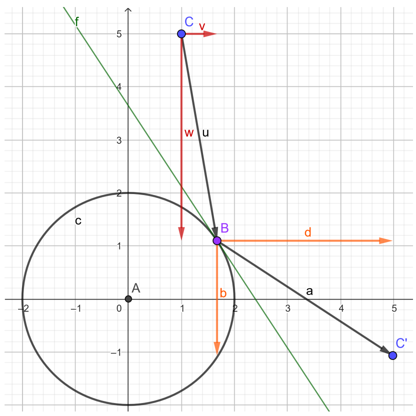


FIGURE 11: Tracé de la situation

Pour ce faire, nous utilisons une formule pour calculer la réflextion par rapport à la droite normal au cercle :

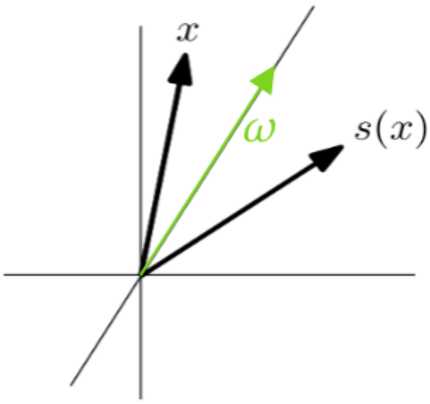


FIGURE 12: Réflexion

$$\boxed{s(x) = 2\langle x \mid \omega \rangle - x}$$

En effet, on peut retrouver cette formule via une construction graphique, figure 13.

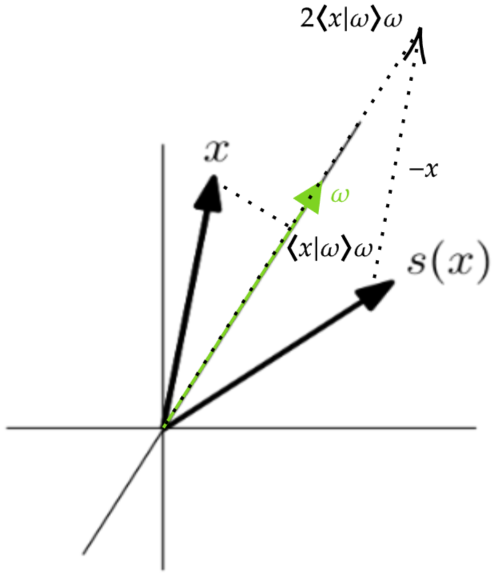


FIGURE 13: Démonstration graphique de la formule de réflexion

Après avoir trouvé le vecteur symétrique, on peut facilement trouver l'équation diriger par ce vecteur. En effet, $s(x) = (-p, m)$ dirige la droite d'équation $y = -\frac{m}{p}x + \frac{c}{m}$ avec c une constante que l'on détermine avec le point d'intersection entre la trajectoire et le cercle. On peut donc poser $a = -\frac{m}{p}$ et $b = \frac{c}{m}$ qui sont respectivement, le coefficient directeur et l'ordonnée à l'origine de la trajectoire de la particule après un rebond avec le cercle.

En utilisant le faite que $||v||^2 = v_x^2 + v_y^2 = cts$ et $a = \frac{v_y}{v_x}$, on obtient :

$$v_x = \frac{||v||}{\sqrt{1+a^2}} \qquad v_y = a\frac{||v||}{\sqrt{1+a^2}}$$

2.3.2 Implémentation informatique

On commence par implémenter des fonctions qui nous servirons pour le programme finale qui calcule la symétrie du vecteur vitesse après un rebond :

```python
def prod_scal(a, b):
    """prend en argument deux vecteurs et retourne le preoduit scalaire"""
    return a[0] * b[0] + a[1] * b[1]

def mult_scalaire_vec(alpha, x):
    """renvoie la multiplication d'un scalaire par un vecteur"""
    return [alpha * x[0], alpha * x[1]]

def norme(x):
    """norme le vecteur a"""
    return sqrt((x[0])**2 + (x[1])**2)
```

Puis, nous implémentons deux fonctions qui nous permet de déterminer le point de contacte entre le cercle et la trajectoire de la particule :

```python
def detect(xc, yc, R, X, Y, vx, vy):
    """prend en argument origine du cercle, le rayon, les listes de coordonnées et les
    composantes de vitesse et return True s'il y a intersection entre le cercle et la
    trajectoire"""
    alpha = vx ** 2 + vy ** 2
    b = 2 * (vx * (X[-1] - xc) + vy * (Y[-1] - yc))
    c = xc ** 2 + yc ** 2 + X[-1] ** 2 + Y[-1] ** 2 - 2 * (xc * X[-1] + yc * Y[-1]) - R ** 2
    discriminant = b ** 2 - 4 * alpha * c
    t1 = (- b + sqrt(abs(discriminant))) / (2 * alpha)
    t2 = (- b - sqrt(abs(discriminant))) / (2 * alpha)
    if (discriminant < 0):
        return False
```

```
    elif t1<=0 and t2<=0:
        return False
    else :
        return True
```

Enfin, nous pouvons construire le programme symétrie qui nous permet de trouver le vecteur symétrique de la vitesse par rapport à un vecteur normal au cercle :

```
def vec_normal_cercle(xc, yc, R, X, Y, vx, vy):
    """prend en argument origine du cercle, le rayon, les listes de coordonnées
    et les composantes de vitesse et retourne le vecteur normal à la droite tangante au cercle du point d'intersection"""
    intersection_circle_halfline(xc, yc, R, X, Y, vx, vy)
    vec_directeur = [ X[-1] - xc, Y[-1] - yc]
    return [(1 / norme(vec_directeur)) * vec_directeur[0] , (1 / norme(vec_directeur)) * vec_directeur[1]]

def symetrique(xc, yc, R, X, Y, vx, vy):
    """prend en argument l'origine du cercle, le rayon, les listes de coordonnées et les composantes de vitesse
    et retourne le symétrique du vecteur incident """
    omega = vec_normal_cercle(xc, yc, R, X, Y, vx, vy)
    moins_v = [X[-2] - X[-1],Y[-2] - Y[-1]]
    prod = prod_scal(moins_v, omega)
    res =mult_scalaire_vec(2 * prod, omega)
    return [res[0] - moins_v[0], res[1] - moins_v[1]]
```

Pour finir, nous pouvons réutiliser ces fonctions pour construire la fonction after_rebond qui nous permet de changer les composantes de la vitesses après un rebond :

```
def after_rebond(vec, vx, vy):
    """retourne vx_rebond et vy_rebond en fonction du coeff directeur et de l'ordonné à l'origine de la droite"""
    V=(vx**2+vy**2)**(1/2)
    if vec[0] == 0 :
        return 1,np.copysign(V, vec[1])
    else :
        a= vec[1] / vec[0]
        V = sqrt(vx**2 + vy**2)
        vx = np.copysign(V * sqrt(1 / (1 + a**2)), vec[0])
        vy = np.copysign(V * sqrt(1 / (1 + (1/ a**2))), vec[1])
        return vx,vy
```

Il nous suffit d'adapter la fonction trajectoire avec la nouvelle fonction, on obtient :

```
def trajectoire(larg, haut, ouv, X, Y, vx, vy):
    """prend en arguments, les dimensions de la boîte, la liste des positions de la particule, la liste des cercles et les composantes de la vitesse les paramètres du cercle et return le nombre de choc entre la particule et les parois et deux listes qui permettent d'avoir toutes les positions de la particule pendant la simulation"""
    FLAG = 0
    count = 0
    while FLAG != 1:
            while detect(xc, yc, R, X, Y, vx, vy) != False:     #detecte si il y intersection avec des cercles
                    vec = symetrique(xc,yc,R,X, Y, vx, vy)
                    vx, vy = after_rebond(vec, vx, vy)
                    X[-1]= X[-1] + R_part * vx
                    Y[-1]= Y[-1] +R_part * vy
            vx, vy, FLAG = trouver_intersec_composante(larg, haut, ouv, X, Y, vx, vy, FLAG)
            count += 1
    return X, Y
```

La fonction trouver_intersec_composante étant la même que précédemment.
On obtient donc après exécution du programme :

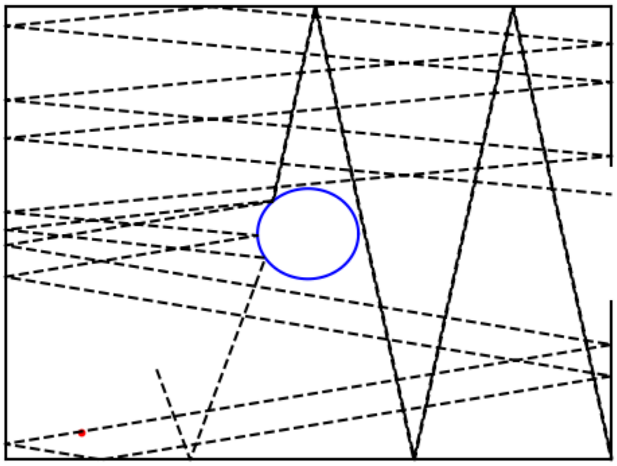


FIGURE 14: Mouvement de la particule dans la boîte

2.4 Problème : Boîte contenant plusieurs cercles

Pour ce problème, on utilise les mêmes programmes que précédemment en les adaptant :

```

def detect_general(X, Y, C, vx, vy):
    """fonction qui prend en arguments X,Y , C (les différents cercles) et les composantes vx,
    vy et elle retourne soit l'indice avec lequel il y a intersection ou non"""
    n=len(C)
    L=np.ones(n)
    indices = [i for i in range(len(L))]
    for i in range(len(C)):
        L[i]=sqrt((C[i][0]-X[-1])**2+(C[i][1]-Y[-1])**2)        #calcule de la distance avec
    les cercles et le point précédent
    indice_triee= [i for _, i in sorted(zip(L, indices))]
    j=0
    for j in indice_triee :
        if detect(C[j][0], C[j][1],C[j][2], X, Y, vx, vy):
            return j    #donne l'indice du cercle de la liste,
                        #le premier cercle a pour paramètres [0,0,0] pour faciliter la
    fonction trajectoire
```

```

def trajectoire(larg, haut, ouv, X, Y, vx, vy, C):
    """prend en arguments, les dimensions de la boîte, la liste des positions de la particule,
    la liste des cercles et les composantes de la vitesse les paramètres du cercle et return
    le nombre de choc entre la particule et les parois et deux listes qui permettent d'avoir
    toutes les positions de la particule pendant la simulation"""
    FLAG = 0
    count = 0
    while FLAG != 1:
            while detect_general(X, Y, C , vx, vy):      #detecte si il y intersection avec des
     cercles
                    k = detect_general(X, Y, C , vx, vy)
                    vec = symetrique(C[k][0],C[k][1],C[k][2],X, Y, vx, vy)
                    vx,vy = after_rebond(vec, vx, vy)
                    X[-1]=X[-1] + R_part*vx
                    Y[-1]=Y[-1] +R_part*vy
            vx, vy, FLAG = trouver_intersec_composante(larg, haut, ouv, X, Y, vx, vy, FLAG)
            count += 1
```

Après exécution, on obtient :

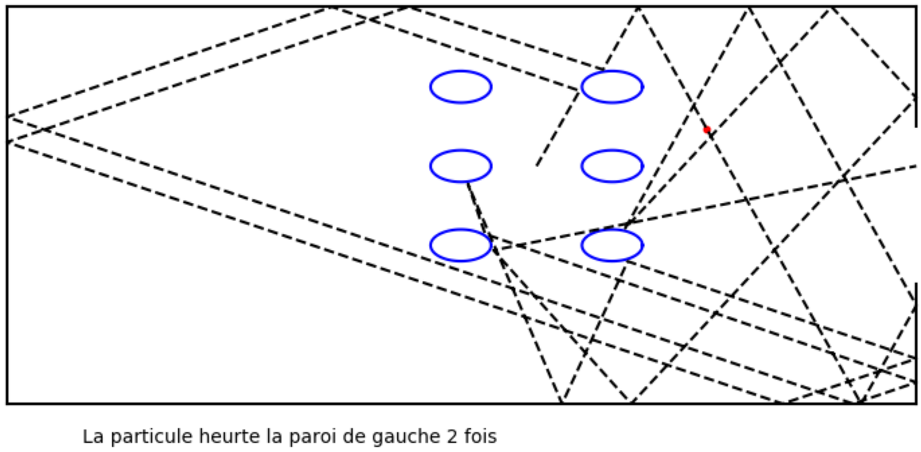


FIGURE 15: Mouvement de la particule dans la boîte

3 Interface graphique

Pour l'interface graphique nous avons utilisé le module matplotlib.animation.

Nous avons utilisé les mêmes programmes pour l'animation de la boîte. En fonction du cahier des charge nous avons adapté les programmes.

Pour créer la boîte, les cercles et l'ouverture. Nous avons utilisé ces programmes :

```
def ligne(X, Y, **kwargs):
    """fonction qui crée les lignes de la boite"""
    plt.plot([X[0],X[1]],[Y[0],Y[1]], **kwargs)

def creer_boite( larg, haut, ouv):
    """fonction qui crée la boite"""
    ligne([0, larg], [0, 0], color='black')
    ligne([0, 0], [0, haut], color='black')
    ligne([0, larg], [haut, haut], color='black')
    ligne([larg, larg], [0, (haut - ouv) / 2], color='black')
    ligne([larg, larg], [(haut + ouv) / 2, haut], color='black')
    plt.axis("off")
    plt.title("MOUVEMENT D'UNE PARTICULE DANS UNE BOITE")

def plot_circle_halfline(xc, yc, R):
    """permet de créer le cercle"""
    # Génération des points du cercle
    t = np.linspace(0, 2 * np.pi, 100)
    x = xc + R * np.cos(t)
    y = yc + R * np.sin(t)
    # Affichage du cercle
    plt.plot(x, y, 'b-')
    plt.axis("off")
```

Pour l'animation de la boule :

```
X,Y = trajectoire(larg, haut, ouv, X, Y, vx, vy) #actualise les liste X,Y avec les points de
    collisions

def distance_points(X,Y):
    """fonction qui prends en arguments X et Y et qui retourne une liste des distances entre
    les points"""
    D = []
    for i in range (len(X) - 1):
        D.append(sqrt((X[i+1] - X[i]) ** 2 + (Y[i+1] - Y[i]) ** 2))
    return D

```

```python
D = distance_points(X,Y)
D = np.array(D)
D = np.round(D).astype(int)      #nécessité d'avoir des entiers

def liste_anim(X,Y):
    """fonction qui prend en arguments X,Y c'est à dire les différents points de collisions et
    calcule des points intermédiaires pour l'animation """
    longueur = len(X)
    X_tmp, Y_tmp = np.copy(X), np.copy(Y)    #on copie les listes X,Y sinon on utilisera des noouveaux points de la liste
    for i in range(longueur - 1):
        index = longueur - i - 2
        for j in range(D[index] - 1, 0, -1):
            X_int = X_tmp[index] + j * (X_tmp[index + 1] - X_tmp[index]) / D[index]
            Y_int = Y_tmp[index] + j * (Y_tmp[index + 1] - Y_tmp[index]) / D[index]
            X.insert(index + 1, X_int)
            Y.insert(index + 1, Y_int)

liste_anim(X,Y)                          #actualise les listes X,Y

def creation_fig(larg,haut,ouv,xc,yc,R):
    creer_boite(larg,haut,ouv)       #on utilise la fonction creer_boite pour faire le cadre
    plot_circle_halfline(xc, yc, R)  #permet de tracer les différents cercles

def frames(X):
    """fonction qui prend en argument la liste X actualisée et retourne une liste"""
    T = []
    for i in range (len(X)):
        T.append(i)
    return T

Tmps = frames(X) #la longueur de cette liste nous donnera le nombre de frames
fig, ax = plt.subplots()
scat = ax.scatter(X, Y, c='r',s=100*R_part)

def animate(i):
    """"fonction qui permet de mettre en place les points sur la figure, fonctionne avec FuncAnimation """
    scat.set_offsets((X[i], Y[i]))

creation_fig(larg,haut,ouv, xc, yc, R)

ani = animation.FuncAnimation(fig, animate, frames=len(Tmps), interval=40, repeat=False)
plt.axis('equal')
plt.show()
```

Pour le dernier problème, nous avons adapter cette derrière commande pour actualiser les différents cercles.