La science du coup franc : analyse, modélisation et optimisation de la précision dans le football

HALIMI Akram

SCEI : 20092

INTRODUCTION

Figure-1: Coup franc indirect

Figure-2: Coup franc direct

Credit Photo - icon sport

Problématique

Comment la modélisation et l'analyse approfondie du coup franc dans le football peuvent-elles contribuer à perfectionner les compétences individuelles des joueurs ?

Sommaire

01. Etude et analyse

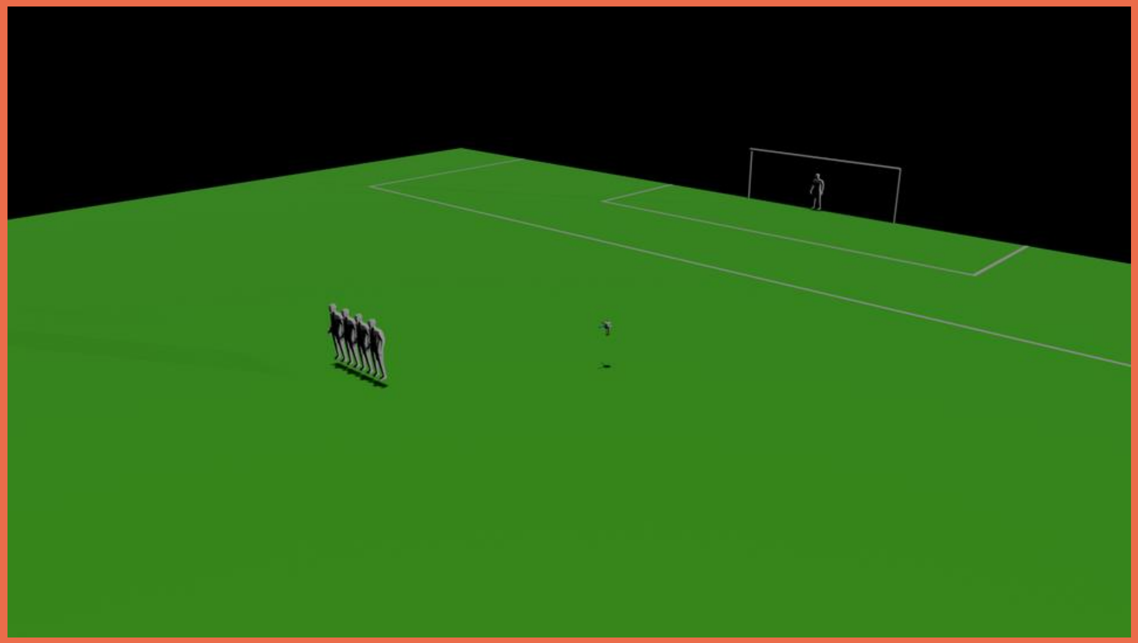


Figure-3 : Vue d'ensemble d'un tir

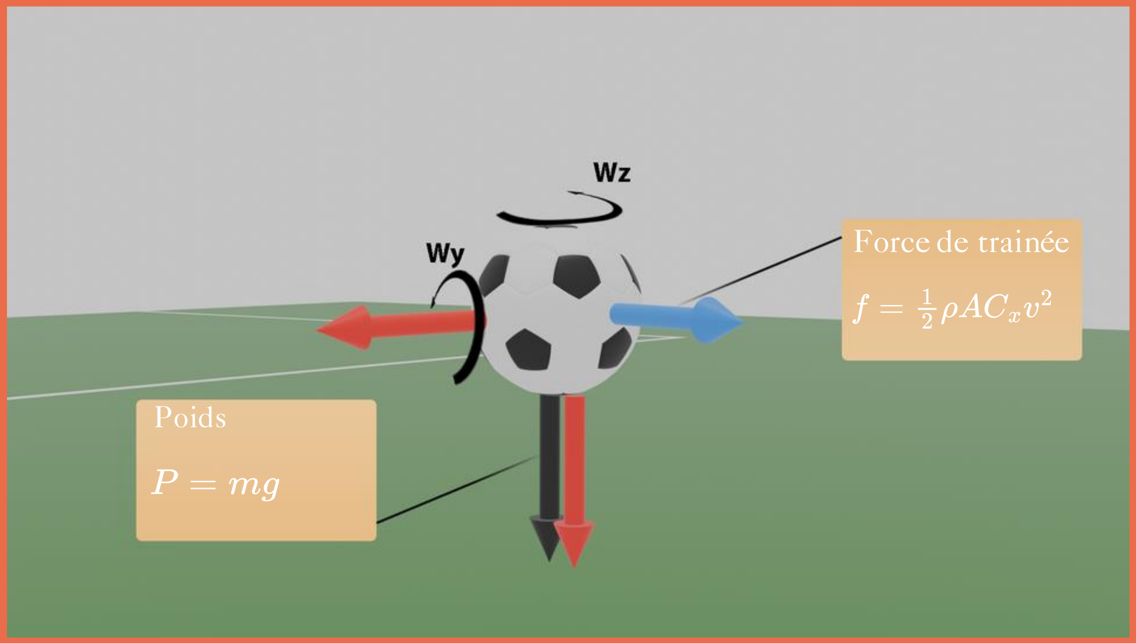


Figure-4 : Représentation des forces agissant sur la balle

Force de trainée

$$f = \frac{1}{2}\rho A C_x v^2$$

- Nombre de Reynolds :

$$R_e = \frac{\rho d v}{\eta}$$

d est le diamètre de la balle en m
v est la vitesse de la balle en $m \cdot s^{-1}$
η est la viscosité dynamique du fluide en $Pa \cdot s^{-1}$
ρ est la masse volumique du fluide en $kg \cdot m^{-3}$

Figure-5: sillage sans rotation
(https://fr.wikipedia.org/wiki/Effet_Magnus)

- Pour une balle de football :

$d = 0.11m, \quad \eta_{air} = 1.8 \cdot 10^{-5} Pa \cdot s^{-1}, \quad \rho_{air} = 1.2 kg \cdot m^{-3} \quad et \quad 1 < v < 50$

$$7.3 \cdot 10^3 < R_e < 3.7 \cdot 10^5$$

Force de trainée

$$\boxed{f = \frac{1}{2}\rho A C_x v^2}$$

- Pour $7.3 \cdot 10^3 < R_e < 3.7 \cdot 10^5$, l'écoulement est turbulent.
- Dans le cadre de notre étude, C_x demeure sensiblement constant

$\overrightarrow{f} = -\frac{1}{2}\rho A C_x v \overrightarrow{v}$

01. Etude et analyse

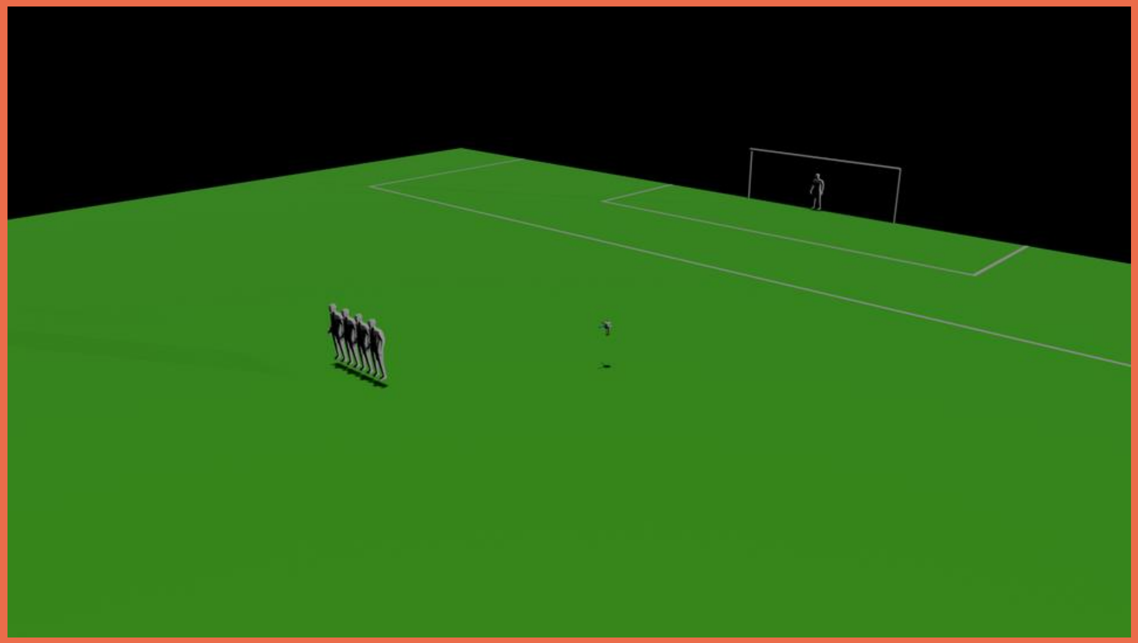


Figure-3 : Vue d'ensemble d'un tir

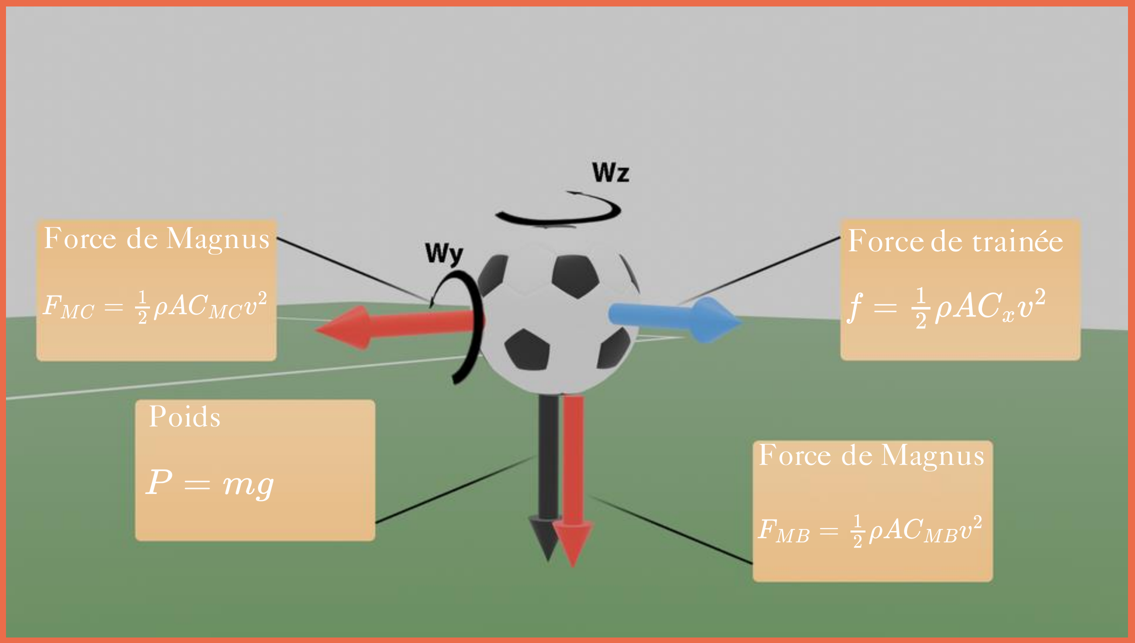


Figure-4 : Représentation des forces agissant sur la balle

Force de Magnus

Figure-8 : Sillage avec rotation
(https://fr.wikipedia.org/wiki/Effet_Magnus)

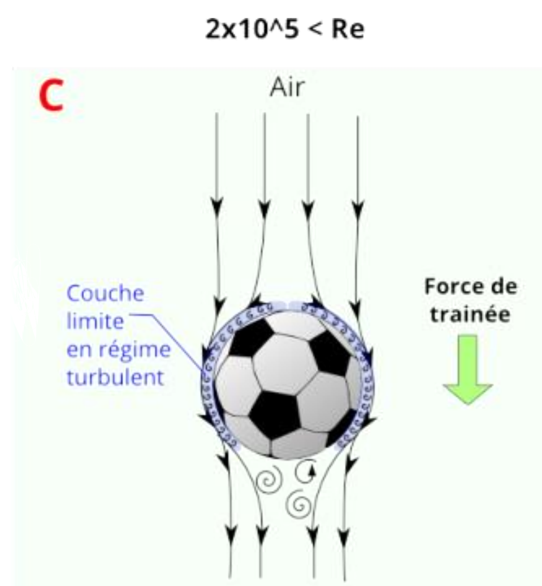


Figure-9 : Couche limite pour un écoulement turbulent

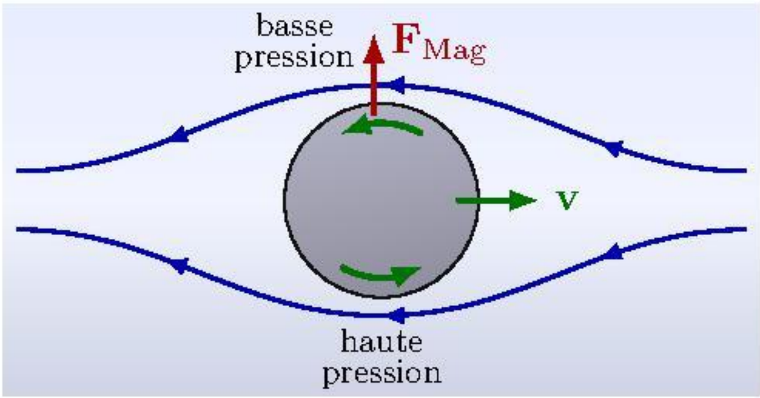


Figure-10 : Représentation de la force de Magnus
(https://tikz.net/fluid_dynamics_magnus/)

Force de Magnus

- Dans la littérature:

$$\overrightarrow{F_{mag}} = \alpha\overrightarrow{\omega} \wedge \overrightarrow{v}$$

- Nous utiliserons :

$$F_{MC} = \frac{1}{2}\rho A C_{MC} v^2 \quad et \quad F_{MB} = \frac{1}{2}\rho A C_{MB} v^2$$

$$avec \quad C_{MC} = \frac{1}{2+\frac{v}{R\omega_z}} \quad et \quad C_{MB} = \frac{1}{2+\frac{v}{R\omega_y}}$$

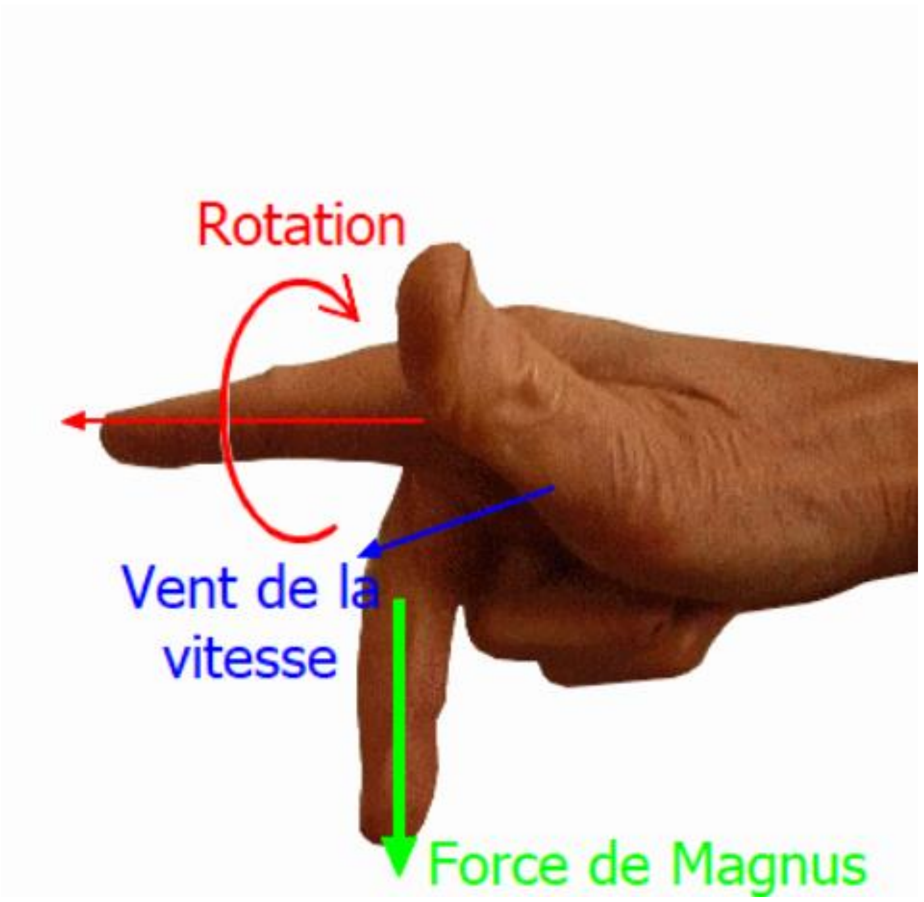
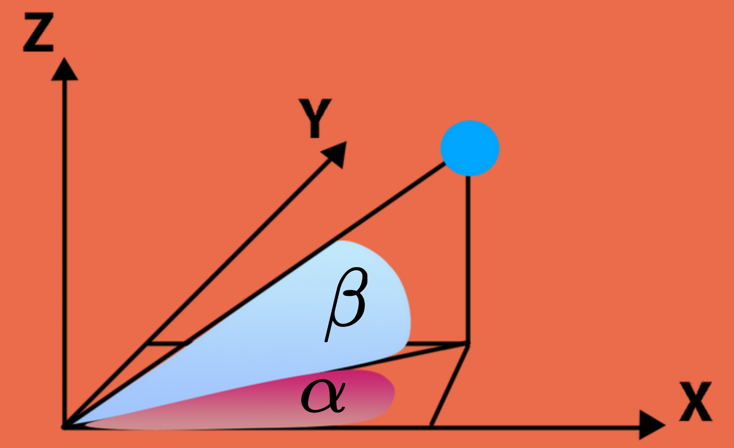


Figure-11 : Règle des trois doigts
(https://fr.wikipedia.org/wiki/Effet_Magnus)

02. Mise en équation



Système : {ballon}

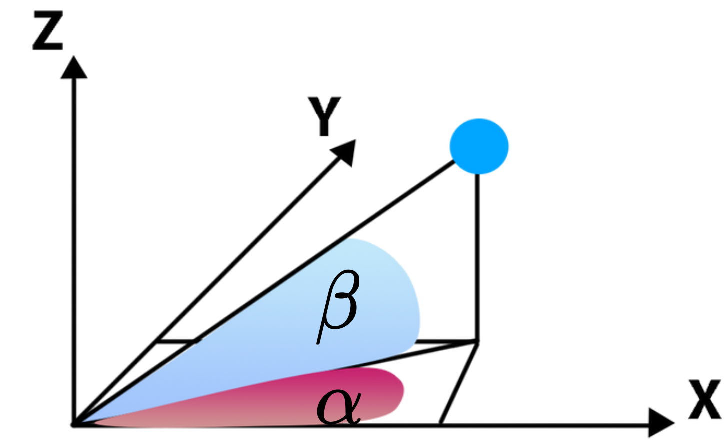
Référentiel : Terrestre supposé Galiléen

BAME : $\overrightarrow{P}, \overrightarrow{f}, \overrightarrow{F_{MC}}, \overrightarrow{F_{MB}}$

On pose $k = \frac{1}{2m}\rho A$

$$\begin{cases} v_x = vcos(\beta)cos(\alpha) \\ v_y = vcos(\beta)sin(\alpha) \\ v_z = vsin(\beta) \end{cases}$$

Mise en équation



- Principe fondamentale de la dynamique

$$m\overrightarrow{a} = \overrightarrow{P} + \overrightarrow{f} + \overrightarrow{F_{MC}} + \overrightarrow{F_{MB}}$$

$$\begin{cases} v_x = vcos(\beta)cos(\alpha) \\ v_y = vcos(\beta)sin(\alpha) \\ v_z = vsin(\beta) \end{cases}$$

- En projettant sur x :

$$ma_x = -fcos(\beta)cos(\alpha) + F_{MB}sin(\beta)cos(\alpha) - F_{ML}sin(\alpha)$$
$$a_x = -kC_xv^2cos(\beta)cos(\alpha) + kC_{MB}v^2sin(\beta)cos(\alpha) - kC_{MC}v^2sin(\alpha)$$
$$a_x = -kC_xvv_x + kC_{MB}vv_zcos(\alpha) - kC_{MC}v^2sin(\alpha)$$
$$a_x = -kv(C_xv_x - C_{MB}v_zcos(\alpha) + C_{MC}vsin(\alpha))$$

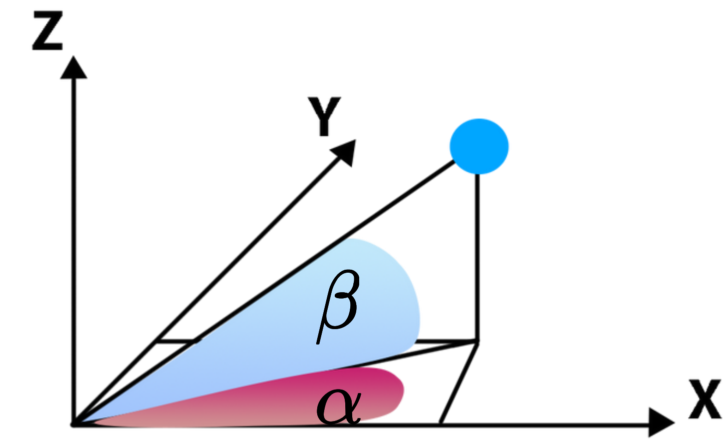
- En projettant sur y :

$$ma_y = -fcos(\beta)sin(\alpha) + F_{MB}sin(\beta)sin(\alpha) + F_{MC}cos(\alpha)$$
$$a_y = -kv(C_xv_y - C_{MB}v_zsin(\alpha) - C_{MC}vcos(\alpha))$$

- En projettant sur z :

$$ma_z = -fsin(\beta) - F_{MB}cos(\beta) - mg$$
$$a_z = -kv(C_xv_z + C_{MB}vcos(\beta)) - g$$

Mise en équation



- Système d'équation différentiel du second degré couplée:

$$\begin{cases} \ddot{x} = -kv(C_x\dot{x} + C_{MC}vsin(\alpha) - C_{MB}\dot{z}cos(\alpha)) \\ \ddot{y} = -kv(C_x\dot{y} - C_{MC}vcos(\alpha) - C_{MB}\dot{z}sin(\alpha)) \\ \ddot{z} = -kv(C_x\dot{z} + C_{MB}vcos(\beta)) - g \end{cases}$$

$$\begin{cases} v_x = vcos(\beta)cos(\alpha) \\ v_y = vcos(\beta)sin(\alpha) \\ v_z = vsin(\beta) \end{cases}$$

Modélisation informatique

- Dans notre cas, nous prendrons $C_x = 0.47$, donc on suppose que la balle est une sphère lisse.

- Le contact entre le pied et la balle sera supposée ponctuelle.

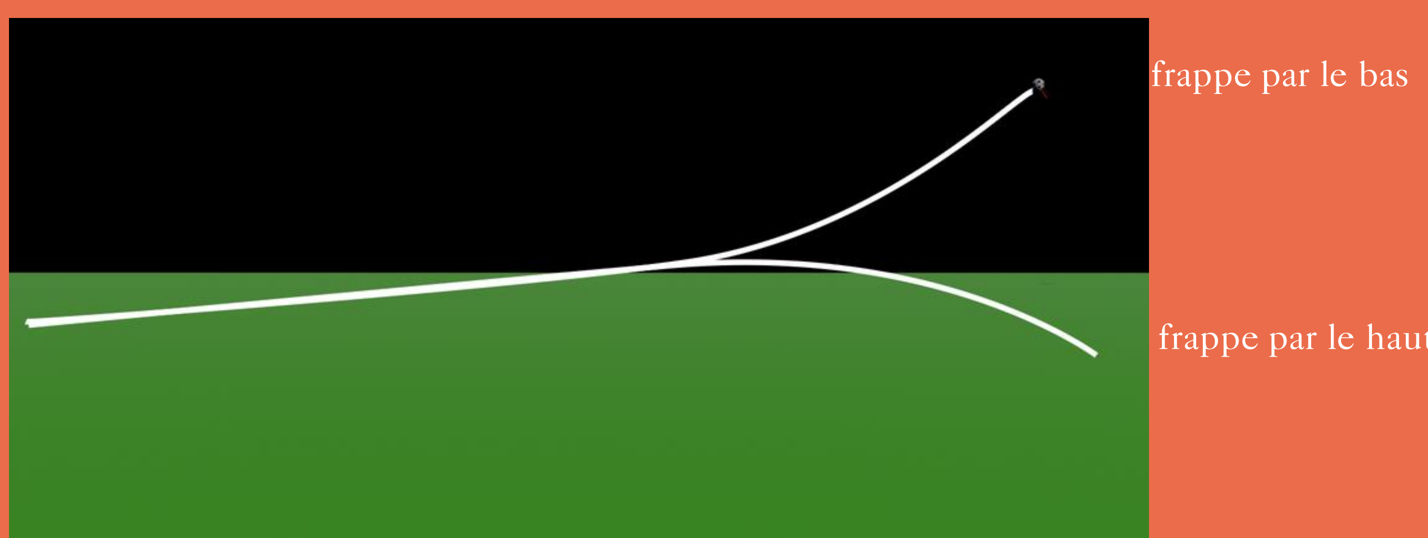


Figure-12 : Vue d'un tir en fonction du point d'impact

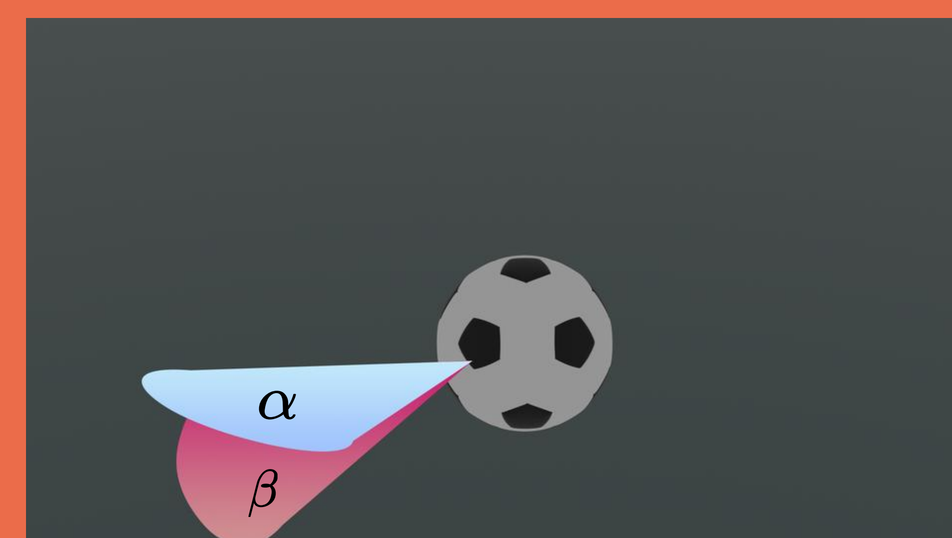
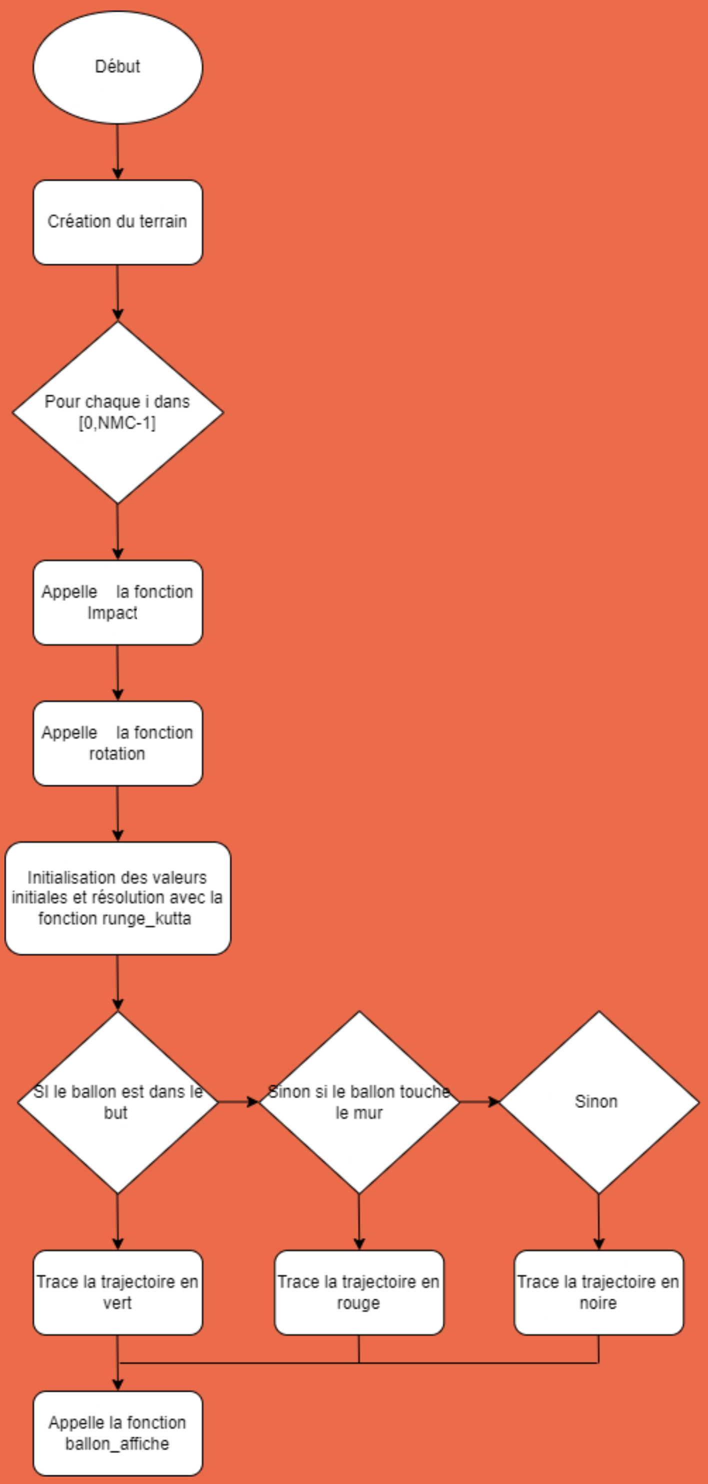


Figure-13 : Représentation des angles d'impact

Modélisation informatique

- Déroulé du programme informatique :
- But du programme : Calculer les coordonnées sur la balle pour un tir optimal.
- Affiche les différentes trajectoires et les points de tirs sur la balle.



Explication fonction impact

- Principe : Génère aléatoirement des nombres en suivant une distribution normale.

- Renvoie un pourcentage pour les composantes Y et Z, et les angles beta et alpha

Explication fonction rotation

- Principe : permet de calculer les composantes de rotation
- Calcul du moment cinétique :

$$
\begin{aligned}
J\vec{\omega} &= \overrightarrow{OM} \wedge m\vec{v} \\
&= \begin{bmatrix} a \\ b \\ c \end{bmatrix} \wedge m \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \\
&= \begin{bmatrix} bv_z - cv_y \\ cv_x - av_z \\ av_y - bv_x \end{bmatrix}
\end{aligned}
$$

- Or $\begin{cases} Fx = Fdtcos(\beta)cos(\alpha) \\ Fy = Fdtcos(\beta)sin(\alpha) \\ Fz = Fdtsin(\beta) \end{cases}$

Explication fonction rotation

- De plus, en utilisant que $m\frac{\Delta v}{\Delta t} = F$ (Formule de l'impulsion), on en déduit que :

$$\begin{cases} vx = Fx/m \\ vy = Fy/m \\ vz = Fz/m \end{cases}$$

- Pour une sphère, $J = \frac{2}{5}mr^2$:

$$\begin{cases} \omega_x = \frac{5}{2mr^2}(bF_z - cF_y) \\ \omega_y = \frac{5}{2mr^2}(-aF_z + cF_x) \\ \omega_z = \frac{5}{2mr^2}(aF_y - bF_x) \end{cases}$$

Résolution du système d'équation

Transformation

$$\begin{cases} \ddot{x} = -kv(C_x\dot{x} + C_{MC}vsin(\alpha) - C_{MB}\dot{z}cos(\alpha)) \\ \ddot{y} = -kv(C_x\dot{y} - C_{MC}vcos(\alpha) - C_{MB}\dot{z}sin(\alpha)) \\ \ddot{z} = -kv(C_x\dot{z} + C_{MB}vcos(\beta)) - g \end{cases}$$

$$\Leftrightarrow \quad \begin{cases} x_1 = \dot{x} \\ y_1 = \dot{y} \\ z_1 = \dot{z} \\ \dot{x_1} = -kv(C_x x_1 + C_{MC}vsin(\alpha) - C_{MB}z_1cos(\alpha)) \\ \dot{y_1} = -kv(C_x y_1 - C_{MC}vcos(\alpha) - C_{MB}z_1sin(\alpha)) \\ \dot{z_1} = -kv(C_x z_1 + C_{MB}vcos(\beta)) - g \end{cases}$$

Expérience 1

- Roberto Carlos : Tir à 35m avec une force de frappe de 16.72 N.s (formule de l'impulsion)

- Première série avec 1000 tirs : Nombre aléatoire suivant une distribution normale.

```
# Coordonnée Y de l'impact
randomY = 0.1*np.random.randn(1, 1)+0.16

# Coordonnée Z de l'impact
randomZ = 0.1*np.random.randn(1, 1)-0.150

# Angle alpha de la frappe
randomalpha = 0.1*np.random.randn(1, 1)-14

# Angle beta de la frappe
randombeta =  0.1*np.random.randn(1, 1)+7.5
```

Expérience 1

- Première série :

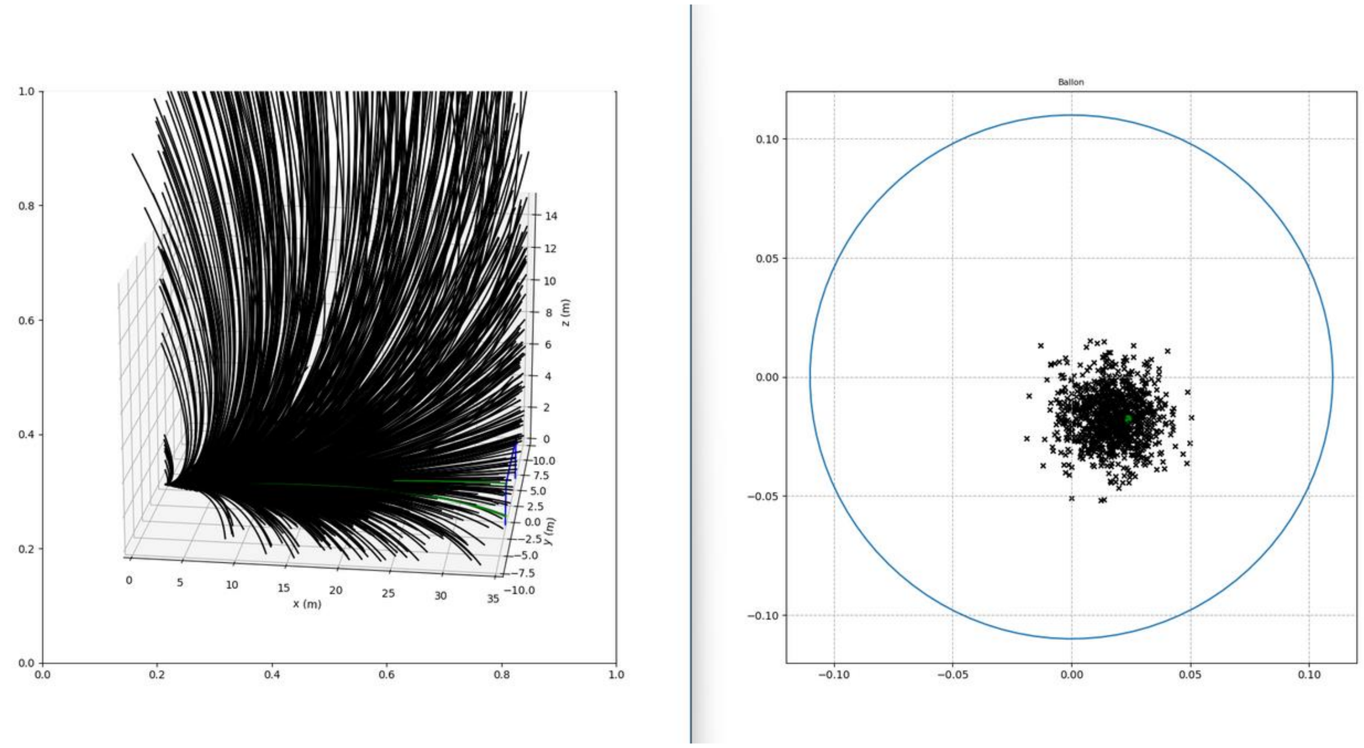


Figure-14 : Trajectoire et point d’impact pour la première série de 1000 tirs

- Résultat maximal pour les coordonnées et les angles :

```
[0.21305142946565703] [-0.16098345738681244]
[-13.923861918642833] [7.659788678511043]
```

Figure-15 : les valeurs maximal de (Y,Z) en pourcent et de (alpha,beta)

Expérience 1

- Seconde série :

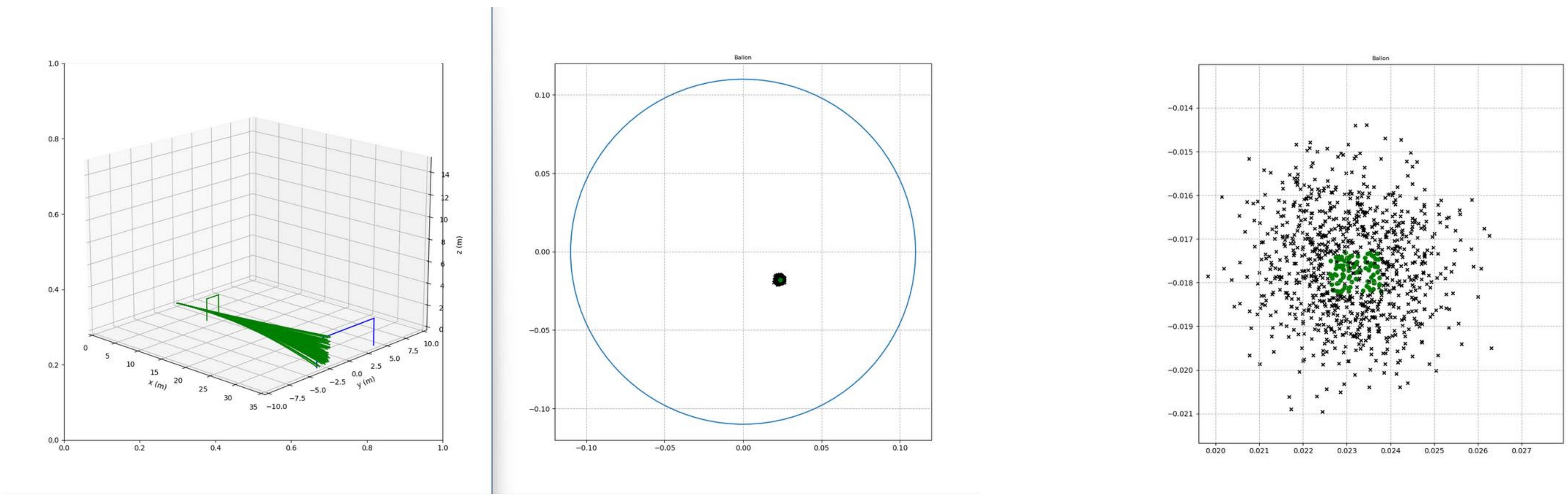


Figure-16 : Trajectoire et point d'impact pour la seconde série de 1000 tirs

Expérience 1

- Dernière série :

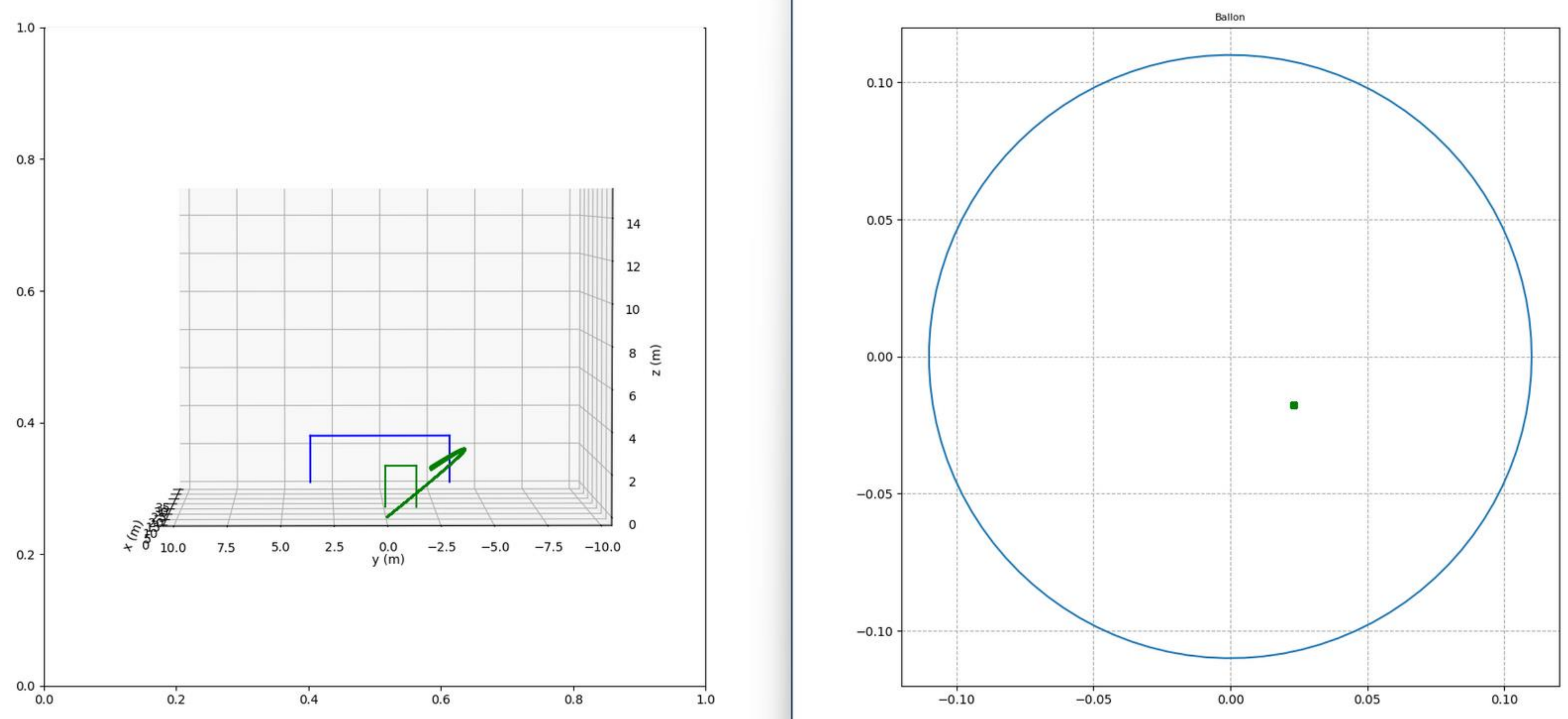


Figure-17 : Trajectoire et point d'impact pour la dernière série de 1000 tirs

Expérience 1

- Validation avec (Y=0.21,Z=-0.16) et (α=-14, β=7.66). On obtient

$v = 38.7 m \cdot s^{-1}$. On trouve, dans la littérature, un angle initial d'environ 12°et une vitesse initiale de $38.1 m \cdot s^{-1}$

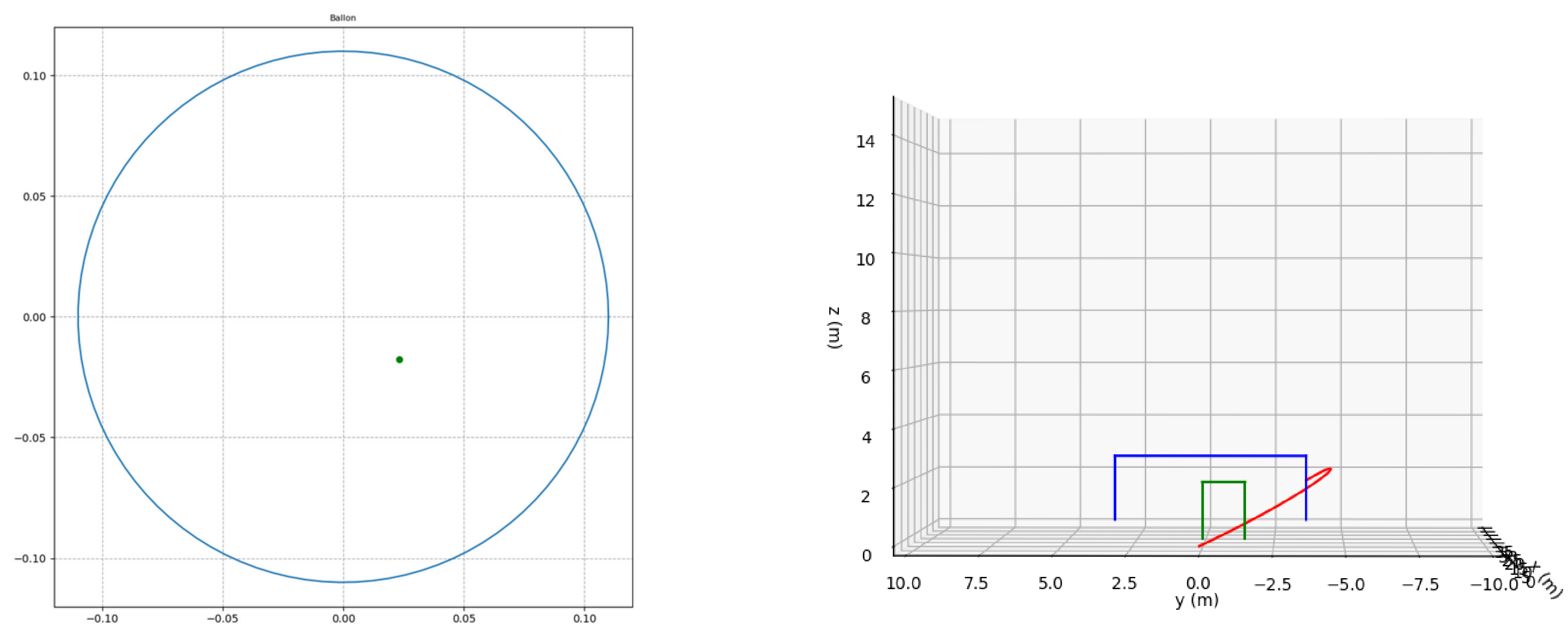


Figure-18 : Reproduction du tir de Roberto Carlos

Expérience 2

Figure-19 : Pointage avec le logiciel regressi

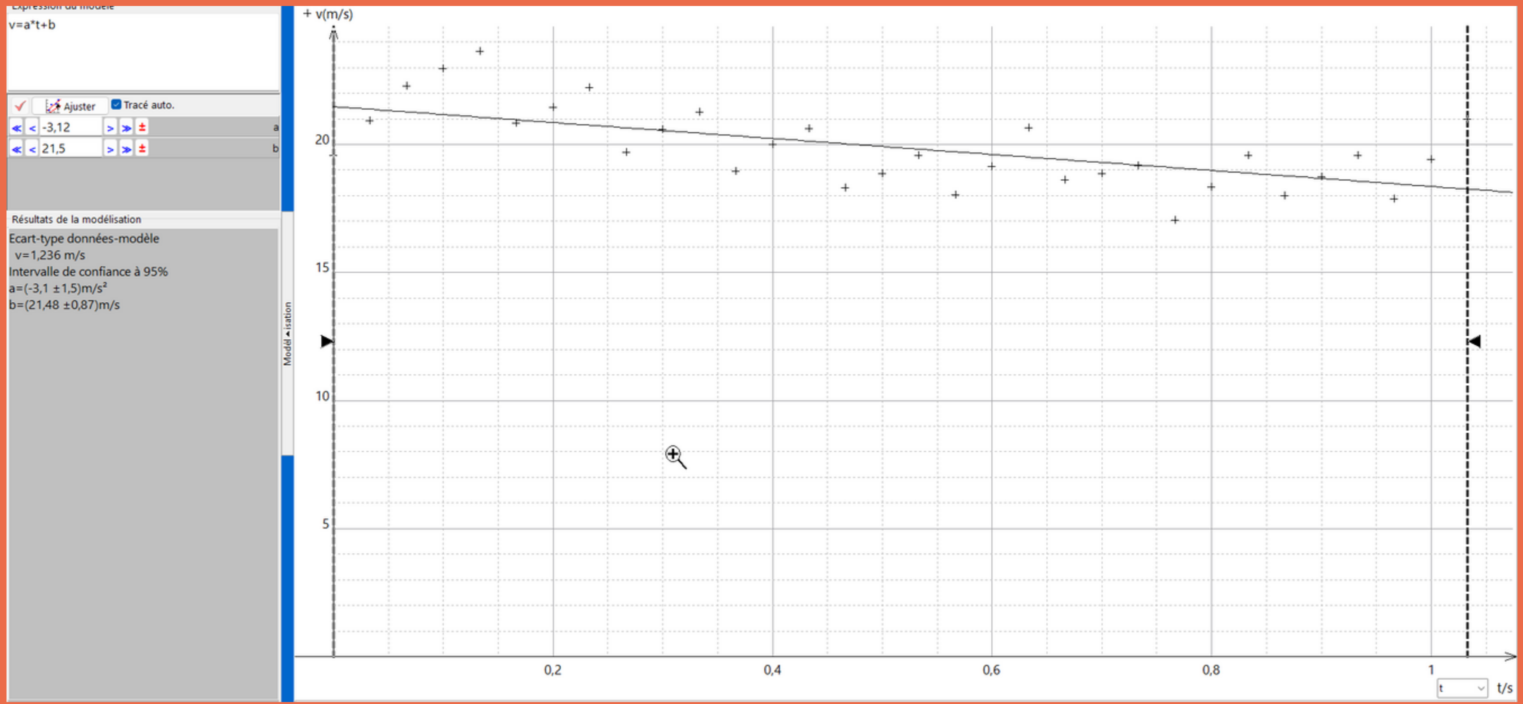


Figure-20 : Courbe de vitesse du tir

Expérience 2

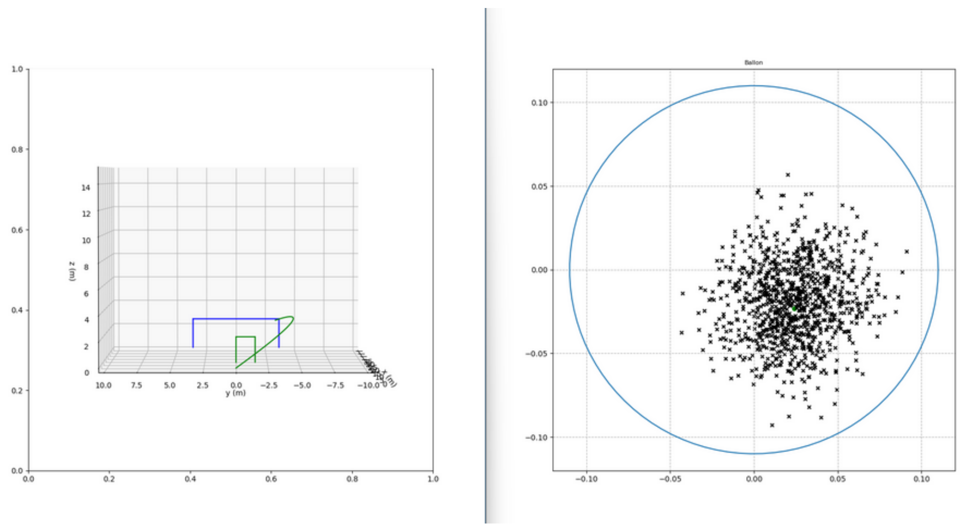


Figure-20 : Première série de tirs

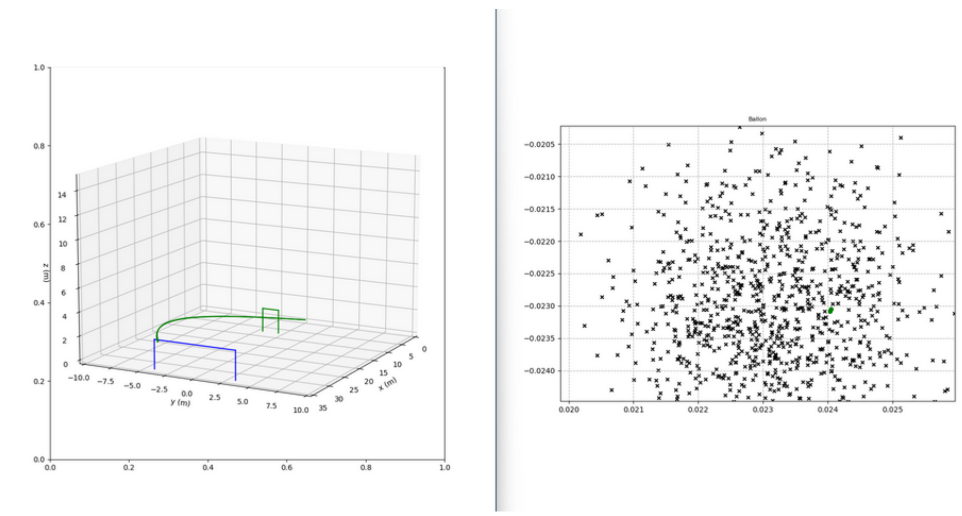


Figure-21 : Seconde série de tirs

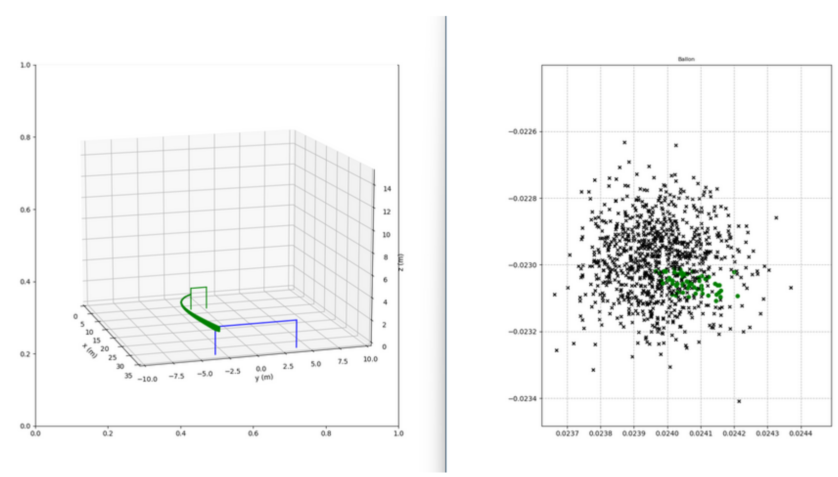


Figure-22 : Dernière série de tirs

Conclusion

- Léger écart avec les autres résultats sur internet pour le coup franc de Roberto Carlos
- Limites : Plusieurs phénomènes n'ont pas été prises en compte
  - La surface de la balle est supposée lisse
  - Les conditions météorologiques ne sont pas pris en compte
  - Difficultés à reproduire le tir lorsque celui-ci est trop précis

Figure-22 : Ballon de la WC10 (Jabulani)

Merci de votre attention

Script python

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


#############################
# Déclaration des variables #
#############################

Fdt = 16.72
Nmc=10
m = 0.44
r = 0.11
rho = 1.2
g = 9.81
A = (r**2) * np.pi
k = (rho * A) / (2 * m)
Cx = 0.47
t_0 = 0.0
t_end = 5.0
h = 0.001
distance=35
```

Script python

```python
def Impact():
    '''

    Prend aléatoire une valeur:
        Yc= coordonnée y du point d'impacte en pourcentage du rayon
        Zc=coordonnée z du point d'impacte en pourcentage du rayon
        alphaHit= angle du tir en degree, angle de x et y
        betaHit=angle du tir en degree, angle de x-y et r

    '''
    # Génération de valeurs aléatoires pour les paramètres de frappe en utilisant une distribution normale (loi normale)

    # Coordonnée Y de l'impact en pourcent
    randomY = 0.1*np.random.randn(1, 1)

    # Coordonnée Z de l'impact en pourcent
    randomZ = 0.1*np.random.randn(1, 1)

    # Angle alpha de la frappe
    randomalpha = 0.1*np.random.randn(1, 1)

    # Angle beta de la frappe
    randombeta =  0.1*np.random.randn(1, 1)

    # Extraction des valeurs scalaires à partir des tableaux générés par np.random.randn
    Yc = randomY[0][0]
    Zc = randomZ[0][0]
    alphaHit = randomalpha[0][0]
    betaHit = randombeta[0][0]

    return Yc, Zc, alphaHit, betaHit
```

Script python

```python
def rotation(Ypercent, Zpercent, r, m, Hitalpha, Hitbeta, Fdt):
    '''
    Cette fonction calcule les composantes de la vitesse, les angles de frappe et les composantes du vecteur de rotation
    d'un ballon en fonction des paramètres d'impact et des pourcentages de Y et Z.
    '''
    # Calcul des coordonnées en fonction des pourcentages et du rayon du ballon
    b = r * Ypercent
    c = r * Zpercent
    # Calcul de la coordonnée 'a' en utilisant l'equation d'une sphère
    a = -np.sqrt(r**2 - c**2 - b**2)

    # Conversion des angles de frappe de degrés en radians
    sinA = np.sin((Hitalpha * np.pi) / 180)
    sinB = np.sin((Hitbeta * np.pi) / 180)
    cosA = np.cos((Hitalpha * np.pi) / 180)
    cosB = np.cos((Hitbeta * np.pi) / 180)

    # Calcul des composantes de la force d'impact
    Fx = Fdt * cosB * cosA
    Fy = Fdt * cosB * sinA
    Fz = Fdt * sinB

    # Calcul des composantes du vecteur de rotation en utilisant les forces et les coordonnées
    Wx = 5 / (2 * m * (r**2)) * (b * Fz - c * Fy)
    Wy = 5 / (2 * m * (r**2)) * (-a * Fz + c * Fx)
    Wz = 5 / (2 * m * (r**2)) * (a * Fy - b * Fx)

    vx = Fx / m
    vy = Fy / m
    vz = Fz / m

    # Calcul des angles de la trajectoire de la balle en radians, puis conversion en degrés
    alpha = (atan(vy / vx) * 180) / np.pi
    beta = (atan(vz / (np.sqrt(vx**2 + vy**2))) * 180) / np.pi

    return vx, vy, vz, alpha, beta, Wx, Wy, Wz
```

Script python

```python
def runge_kutta_4th_order(f, y0, t_0, t_end, h, v, C_MC, C_MB, sinA, cosA, cosB,distance):
    '''
    Résolution de l'équation différentielle en utilisant la méthode de Runge-Kutta d'ordre 4.
    '''

    num_steps = int((t_end - t_0) / h) + 1

    t_values = np.linspace(t_0, t_end, num_steps)
    # Initialisation des valeurs de y
    y_values = np.zeros((num_steps, len(y0)))
    y_values[0] = y0

    # Boucle pour appliquer la méthode de Runge-Kutta à chaque pas de temps
    for i in range(1, num_steps):
        k1 = h * f(t_values[i - 1], y_values[i - 1], v, C_MC, C_MB, sinA, cosA, cosB)
        k2 = h * f(t_values[i - 1] + 0.5 * h, y_values[i - 1] + 0.5 * k1, v, C_MC, C_MB, sinA, cosA, cosB)
        k3 = h * f(t_values[i - 1] + 0.5 * h, y_values[i - 1] + 0.5 * k2, v, C_MC, C_MB, sinA, cosA, cosB)
        k4 = h * f(t_values[i - 1] + h, y_values[i - 1] + k3, v, C_MC, C_MB, sinA, cosA, cosB)

        y_values[i] = y_values[i - 1] + (k1 + 2 * k2 + 2 * k3 + k4) / 6

        # Vérifier si la balle sort
        if (y_values[i, 0] < 0) or (y_values[i, 2] < -10) or (y_values[i, 2] > 10) or y_values[i, 0] > distance:
            # Si la balle touche le mur, retourner les valeurs jusqu'à ce point
            return y_values[:i+1]

        # Vérifier si la balle touche le sol
        if y_values[i, 4] < 0:
            # Si la balle touche le sol, retourner les valeurs jusqu'à ce point
            return y_values[:i+1]

        # Vérifier si la balle entre dans le but
        if y_values[i, 0] == distance and (y_values[i, 4] > 2.44) and (-3.66 < y_values[i, 2] < -3.26):
            # Si la balle entre dans le but, retourner les valeurs jusqu'à ce point
            return y_values[:i+1]

        if y_values[i, 0] ==9.1 and ((y_values[i, 4] < 2.00) and (-1.5 < y_values[i, 2] < 0)):
            return y_values[:i+1]

    return y_values
```

```python
def f(t, Y,v,C_MC,C_MB,sinA,cosA,cosB):
    '''
    Equation différentielle

    '''


    x=Y[0]
    x1=Y[1]

    y=Y[2]
    y1=Y[3]

    z=Y[4]
    z1=Y[5]

    x_dot=x1
    y_dot=y1
    z_dot=z1

    x1_dot= -k*v*(Cx*x1+C_MC*v*sinA-C_MB*z1*cosA)
    y1_dot= -k*v*(Cx*y1-C_MC*v*cosA-C_MB*z1*sinA)
    z1_dot= -k*v*(Cx*z1+C_MB*v*cosB)-g

    return np.array([x_dot, x1_dot, y_dot, y1_dot, z_dot, z1_dot])
```

Script python

```python
def ballon_affiche(points,sol):
    '''
    affiche les points d'impact
    '''
    theta = np.linspace(0, 2*np.pi, 100)
    x1 = r*np.cos(theta)
    x2 = r*np.sin(theta)
    fig, ax = plt.subplots(1)


    ax.plot(x1, x2)
    ax.set_aspect(1)

    plt.xlim(-0.12,0.12)
    plt.ylim(-0.12,0.12)

    plt.grid(linestyle='--')

    for i in points:
        x=i[0]
        y=i[1]
        if not((x/r,y/r) in sol):
            plt.scatter(x, y, c = 'black',marker = 'x',s = 20)
        else:
            plt.scatter(x, y, c = 'green',marker = 'o',s = 30)


    plt.title('Ballon', fontsize=8)
```

```python
def cherche_max(lst):
    '''
    Trouve le maximum pour pouvoir en déduire un intervalle
    '''
    L_Y=[]
    L_Z=[]

    max_Y = lst[0][0]
    max_Z = lst[0][1]

    for num in lst:
        Y=num[0]
        Z=num[1]

        if Y > max_Y:
            max_Y = Y

        if Z > max_Z:
            max_Z = Z


    L_Y.append( max_Y)

    L_Z.append(max_Z)

    return L_Y,L_Z
```

Script python

```python
#######################
# CREATION DU TERRAIN #
#######################

fig, cx = plt.subplots()
cx = plt.axes(projection='3d')

# Définition des points pour la barre transversale du but
goal_z_p = np.linspace(2.44, 2.44, 50)  # Coordonnées z de la barre transversale (hauteur fixe à 2.44m)
goal_x_p = np.linspace(distance, distance, 50)      # Coordonnées x de la barre transversale (distance fixe)
goal_y_p = np.linspace(-3.66, 3.66, 50) # Coordonnées y de la barre transversale (largeur de -3.66m à 3.66m)

# Tracé de la barre transversale du but
cx.plot3D(goal_x_p, goal_y_p, goal_z_p, '-b', label='goal')

# Définition des points pour le poteau droit du but
goal_z_pr = np.linspace(0, 2.44, 50)
goal_x_pr = np.linspace(distance, distance, 50)
goal_y_pr = np.linspace(-3.66, -3.66, 50)

# Tracé du poteau droit du but
cx.plot3D(goal_x_pr, goal_y_pr, goal_z_pr, '-b')

# Définition des points pour le poteau gauche du but
goal_z_pl = np.linspace(0, 2.44, 50)
goal_x_pl = np.linspace(distance, distance, 50)
goal_y_pl = np.linspace(3.66, 3.66, 50)

# Tracé du poteau gauche du but
cx.plot3D(goal_x_pl, goal_y_pl, goal_z_pl, '-b')

# Définition des points pour le haut du mur de joueurs
wall_z_top = np.linspace(2.0, 2.0, 50)
wall_x_top = np.linspace(9.1, 9.1, 50)
wall_y_top = np.linspace(0, -1.5, 50)

# Tracé du haut du mur de joueurs
cx.plot3D(wall_x_top, wall_y_top, wall_z_top, '-g', label='wall of players')

# Définition des points pour le côté droit du mur de joueurs
wall_z_r = np.linspace(2.0, 0.0, 50)
wall_x_r = np.linspace(9.1, 9.1, 50)
wall_y_r = np.linspace(-1.5, -1.5, 50)

# Tracé du côté droit du mur de joueurs
cx.plot3D(wall_x_r, wall_y_r, wall_z_r, '-g')
```

```python
# Définition des points pour le côté gauche du mur de joueurs
wall_z_l = np.linspace(2.0, 0.0, 50)
wall_x_l = np.linspace(9.1, 9.1, 50)
wall_y_l = np.linspace(0.0, 0.0, 50)

# Tracé du côté gauche du mur de joueurs
cx.plot3D(wall_x_l, wall_y_l, wall_z_l, '-g')


cx.set_xlabel('x (m)')
cx.set_ylabel('y (m)')
cx.set_zlabel('z (m)')


cx.set_xlim(0, distance)
cx.set_ylim(-10, 10)
cx.set_zlim(0, 15)
```

Script python

```python
def main():
    sol=[]
    ang_sol=[]
    points=[]
    for i in range(Nmc):
        Yc, Zc, alphaHit, betaHit = Impact()
        vx, vy, vz, alpha, beta, Wx, Wy, Wz = rotation(Yc, Zc, 0.11, 0.41, alphaHit, betaHit, Fdt)
        v = np.sqrt(vx**2 + vy**2 + vz**2)
        angVy = Wy
        angVz = Wz
        angV = np.sqrt(Wx**2 + Wy**2 + Wz**2)
        C_MC = 1 / (2 + (v / (r * angVz)))
        C_MB = 1 / (2 + (v / (r * angVy)))

        sinA = np.sin((alpha * np.pi) / 180)
        sinB = np.sin((beta * np.pi) / 180)
        cosA = np.cos((alpha * np.pi) / 180)
        cosB = np.cos((beta * np.pi) / 180)

        y0 = np.array([0, vx, 0, vy, 0, vz])
        y_values = runge_kutta_4th_order(f, y0, t_0,t_end , h,v,C_MC,C_MB,sinA,cosA,cosB,distance)
        x = y_values[:, 0]
        y = y_values[:, 2]
        z = y_values[:, 4]

        if y_values[-1, 0]>=distance and 2<y_values[-1, 4] < 2.44 and (-3.66 < y_values[-1, 2] <3.66):
            cx.plot3D(x, y, z, color='green')
            sol.append((Yc,Zc))
            ang_sol.append((alphaHit,betaHit))

        elif y_values[-1, 0] ==9.1 and ((y_values[-1, 4] < 2.00) and (-1.5 < y_values[-1, 2] < 0)):
            cx.plot3D(x, y, z, color='red')

        else:
            cx.plot3D(x, y, z, color='black')

        points.append((r*Yc,r*Zc))

    ballon_affiche(points,sol)
    return sol,ang_sol

sol,ang_sol=main()
L_Y,L_Z=cherche_max(sol)
L_A,L_B=cherche_max(ang_sol)
print(L_Y,L_Z)
print(L_A,L_B)
plt.show()
```