La Lumière du Savoir

مدرسة نور المعرفة

Année 2022-2023 - Grand Oral de Spécialité

Question de mathématique

Les mathématiques peuvent-elles nous aider à marquer des buts ? (Étude de la trajectoire d'un ballon de football)

Auteur : Yacine DIDANE

Les équations du mouvement d'un ballon dépendent du contexte spécifique (conditions atmosphériques) et des forces qui agissent sur le ballon. Voici une formulation générale des équations du mouvement pour un ballon en mouvement sous l'influence de la gravité et de la force de traînée de l'air :

1. Équation de la force de gravité

$$F_g = m \cdot g \qquad (1)$$

où :

- F_g est la force gravitationnelle (N),
- m est la masse du ballon (kg),
- g est l'accélération due à la gravité ().

2. Équation de la force de traînée de l'air (résistance de l'air) :

$$Fd = 0.5 \cdot \rho \cdot A \cdot Cd \cdot v^2 \qquad (2)$$

où :

a. Fd est la force de traînée (N),
b. ρ est la densité de l'air (),
c. A est la surface de référence du ballon (),
d. Cd est le coefficient de traînée du ballon (),
e. v est la vitesse du ballon par rapport à l'air (m/s^2).

3. Équation de la force résultante :

$$F = m \times a \qquad (3)$$

où F est la force résultante (somme des forces), obtenue en soustrayant la force de traînée de la force gravitationnelle.

4. Équation du mouvement :

Elle est donnée par la loi fondamentale de la dynamique :

$$F = m \times a \qquad (4)$$

où :

- F est la force appliquée sur l'objet (N),
- m est la masse de l'objet (kg),
- a est l'accélération de l'objet (m/s^2).

La relation fondamentale de la dynamique correspond à la deuxième loi de Newton, qui établit la relation entre la somme des forces appliquée sur un objet, sa masse et son accélération. Cette loi peut être exprimée mathématiquement par une équation différentielle du deuxième ordre. L'équation différentielle correspondante est généralement formulée comme suit :

$$F = m \times \frac{d^2x}{dt^2}$$

En utilisant la définition de l'accélération comme la dérivée seconde de la position par rapport au temps ($a = d^2x/dt^2$).

Cette équation différentielle du deuxième ordre relie la force appliquée sur un objet à sa position dans le temps. En résolvant cette équation, on peut déterminer la trajectoire ou le mouvement de l'objet en fonction des conditions initiales et des forces qui lui sont appliquées.

5. Équation finale du mouvement :

Elle est obtenue en combinant les expression (3) et (4) par la loi fondamentale de la dynamique :

$$m \times a = Fg - Fd \quad (5)$$

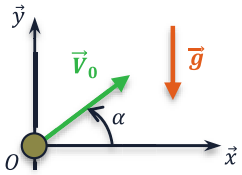
Nota :

a. L'équation (5) est une équation différentielle du second ordre. Il existe deux méthodes pour résoudre ce type d'équation :
   - Méthode de résolution analytique :
     Cette méthode consiste à trouver une solution analytique exacte de l'équation différentielle en utilisant des techniques mathématiques telles que la factorisation, la méthode des coefficients indéterminés, la méthode de variation des constantes, etc. Cependant, il n'est pas toujours possible de trouver une solution analytique pour toutes les équations différentielles du second ordre.
   - Méthode de résolution numérique :
     Lorsqu'il n'est pas possible de trouver une solution analytique, on peut recourir à des méthodes numériques pour résoudre les équations différentielles du second ordre. Ces méthodes impliquent la discrétisation de la variable indépendante (le temps) et l'utilisation d'algorithmes itératifs pour obtenir une solution approchée. Certaines des méthodes numériques les plus couramment utilisées incluent la méthode d'Euler, la méthode de Runge-Kutta, la méthode des différences finies, la méthode des éléments finis, etc.
     La méthode de résolution appropriée dépend de la nature spécifique de l'équation différentielle, de ses conditions initiales, des contraintes de temps et de précision, ainsi que des ressources informatiques disponibles.
     En fonction de la complexité de l'équation et des exigences de précision, il peut être nécessaire d'utiliser des logiciels spécialisés, tels que des bibliothèques de calcul numérique comme NumPy ou des logiciels de calcul symbolique comme SymPy, pour résoudre les équations différentielles du second ordre de manière efficace et précise.

b. Si on ne considère que la force de gravitation (on fait l'hypothèse -abusive- que Fd = 0), l'équation (5) peut être résolue de manière analytique. Les solutions ont été étudiées en cours de Physique et de Science de l'ingénieur (schéma Studyrama) :

Champ de pesanteur $\vec{g}$ uniforme

Mouvement d'un projectile de masse m ayant une vitesse initiale $\vec{V}_0$



2ème loi de Newton
$\vec{a} = \vec{g} \qquad a_x(t) = 0 \qquad a_y(t) = -g$
Primitive et conditions initiales
$V_x(t) = V_0 cos \propto \qquad V_y(t) = -gt + V_0 sin \propto$
Primitive et conditions initiales
$x(t) = V_0 \cos(\propto) t \qquad y(t) = -\frac{1}{2} g t^2 + V_0 \sin(\propto) t$
Energie cinétique
$E_c = \frac{1}{2} m V^2$

c. Dans toute sa généralité, l'équation (5) décrivant le mouvement du ballon doit être résolue de manière numérique. Pour résoudre numériquement cette équation, on va utiliser des méthodes d'intégration numérique, telles que la méthode d'Euler ou la méthode de Runge-Kutta (cf. paragraphes suivants).

d. Cependant, il convient de noter que la modélisation précise du mouvement d'un ballon peut nécessiter des considérations supplémentaires, telles que la pression de l'air à l'intérieur du ballon et les effets aérodynamiques spécifiques au ballon utilisé. Dans ces conditions, des modèles plus sophistiqués devront être développés, en intégrant des données expérimentales.

6. Résolution numérique de l'équation du mouvement avec la méthode d'Euler

La méthode d'Euler est une procédure numérique qui permet de résoudre de façon approximative des équations différentielles ordinaires du premier ordre avec condition initiale. Elle a le mérite d'être simple à comprendre et à programmer [3].

ALGORITHME D'EULER

La méthode présentée ici est dite méthode d'Euler explicite. Considérons l'équation différentielle ordinaire suivante

$$\begin{cases} \dot{\mathbf{y}} & = & f(t, \mathbf{y}(t)), \quad 0 \leq t \leq T \\ \mathbf{y}(0) & = & \mathbf{y}_0 \end{cases}$$

On peut intégrer cette équation comme suit :

$$\mathbf{y}(t_{n+1}) = \mathbf{y}(t_n) + \int_{t_n}^{t_{n+1}} f(t, \mathbf{y}(t)) \, \mathrm{d}t \tag{1}$$

La méthode d'Euler consiste à approcher l'intégrale par la méthode des rectangles à gauche :

$$\int_{t_n}^{t_{n+1}} f(t, \mathbf{y}) \, \mathrm{d}t \simeq h \times f(t_n, \mathbf{y}(t_n))$$

D'où le schéma itératif suivant

$$\begin{cases} \mathbf{y}_{n+1} & = & \mathbf{y}_n + h \, f(t_n, \mathbf{y}_n), \quad n = 0, 1, \ldots, N \\ \mathbf{y}_0 & = & \mathbf{y}(0) \end{cases} \tag{2}$$

où $\mathbf{y}_n$ désigne l'approximation numérique de $\mathbf{y}(t_n)$. La mise en œuvre est alors extrêmement simple :

♡ **Algorithme d'Euler**

- Initialisation du pas h et de la durée T.
- Initialisation des conditions initiales : $t = 0$ et $\mathbf{y} = \mathbf{y}(0)$.
- Tant que $t \leq T$ faire :
  - Calcul de $\mathbf{k}_1 = f(t, \mathbf{y})$.
  - $\mathbf{y} = \mathbf{y} + h\mathbf{k}_1$; $t = t + h$.
  - Enregistrement des données.

Le programme python de résolution par la méthode d'Euler est le suivant :

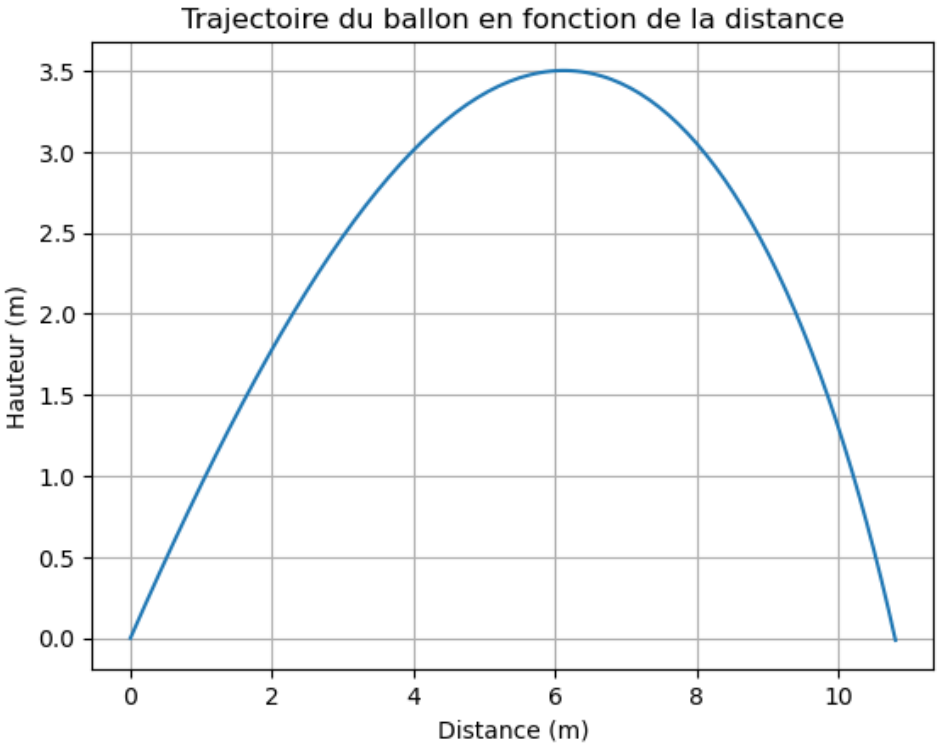
```
import numpy as np

import matplotlib.pyplot as plt

def calculate_trajectory(m, g, rho, A, Cd, v0, theta, dt, t_max):

   # Initialisation des listes de données

   t_list = [0]
   x_list = [0]
   y_list = [0]

   vx_list = [v0 * np.cos(theta)]
   vy_list = [v0 * np.sin(theta)]

   while (t_list[-1] < t_max) and (y_list[-1] >= 0.) :
```

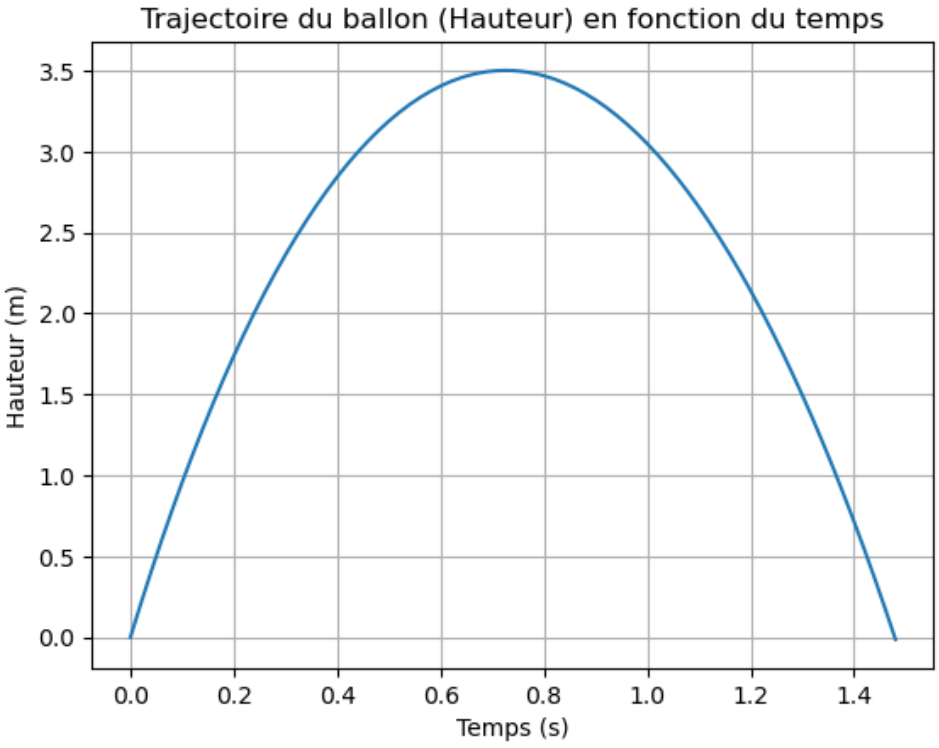
```
        # Calcul des forces

        Fg = m * g
        Fd = 0.5 * rho * A * Cd * np.sqrt(vx_list[-1]**2 + vy_list[-1]**2)**2

        # Calcul de l'accélération

        ax = -Fd * np.cos(theta) / m
        ay = -Fd * np.sin(theta) / m - g

        # Calcul des vitesses

        vx = vx_list[-1] + ax * dt
        vy = vy_list[-1] + ay * dt

        # Calcul des positions par la méthode d Euler

        x = x_list[-1] + vx * dt
        y = y_list[-1] + vy * dt

        # Ajout des données à la liste

        t_list.append(t_list[-1] + dt)
        x_list.append(x)
        y_list.append(y)

        vx_list.append(vx)
        vy_list.append(vy)

    return t_list, x_list, y_list


# Paramètres du ballon et du mouvement

m = 0.45  # Masse du ballon en kg
g = 9.81  # Accélération gravitationnelle en m/s^2
rho = 1.2  # Densité de l'air en kg/m^3
A = 0.1  # Surface de référence du ballon en m^2
Cd = 0.47  # Coefficient de traînée du ballon

v0 = 15  # Vitesse initiale en m/s
theta = np.pi / 4  # Angle de tir en radians

dt = 0.01  # Pas de temps en secondes
t_max = 10  # Temps total de simulation en secondes

# Calcul de la trajectoire

t, x, y = calculate_trajectory(m, g, rho, A, Cd, v0, theta, dt, t_max)

# Tracé de la trajectoire Hauteur = f(Distance)
```

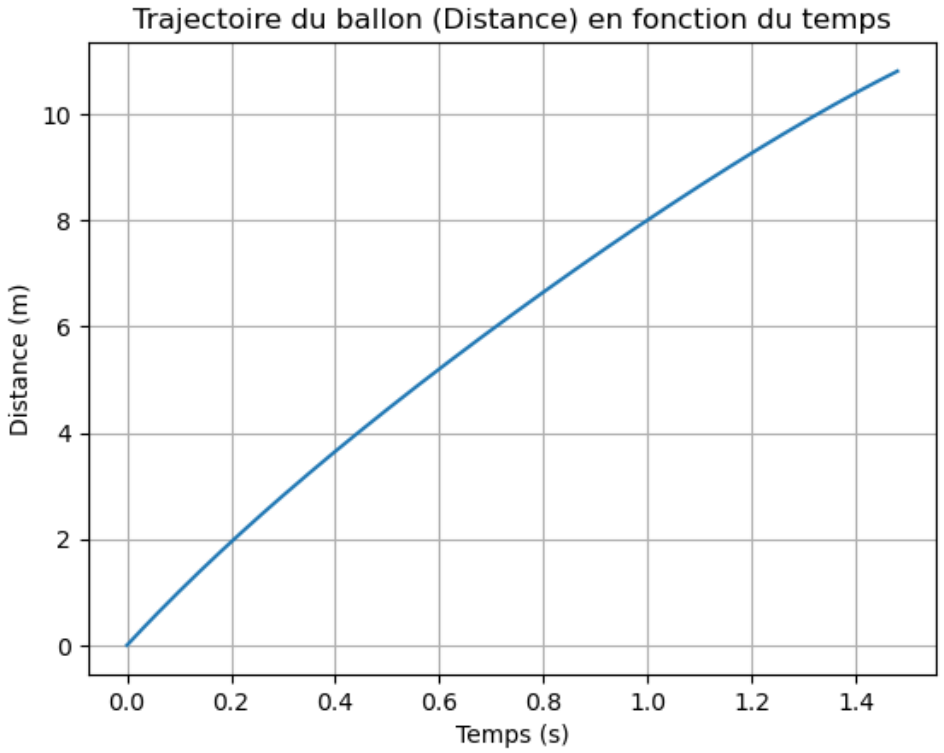
```
plt.plot(x, y)
plt.xlabel('Distance (m)')
plt.ylabel('Hauteur (m)')
plt.title('Trajectoire du ballon en fonction de la distance')
plt.grid(True)
plt.show()

# Tracé de la trajectoire Hauteur = f(Temps)

plt.plot(t, y)
plt.xlabel('Temps (s)')
plt.ylabel('Hauteur (m)')
plt.title('Trajectoire du ballon (Hauteur) en fonction du temps')
plt.grid(True)
plt.show()

# Tracé de la trajectoire Distance = f(Temps)

plt.plot(t, x)
plt.xlabel('Temps (s)')
plt.ylabel('Distance (m)')
plt.title('Trajectoire du ballon (Distance) en fonction du temps')
plt.grid(True)
plt.show()
```

Ce programme résout les équations du mouvement en utilisant la méthode d'Euler pour intégrer numériquement l'équations différentielle (5). Il calcule les coordonnées x et y du ballon à chaque pas de temps et trace ensuite la trajectoire du ballon en utilisant la bibliothèque matplotlib.

Il est possible d'ajuster les paramètres du ballon et du mouvement.


Trajectoire du ballon en fonction de la distance
Hauteur (m)
Distance (m)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
2
4
6
8
10


Trajectoire du ballon (Hauteur) en fonction du temps
Hauteur (m)
Temps (s)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4



7. Résolution numérique de l'équation du mouvement avec la méthode de Runge-Kutta

Les techniques de Runge-Kutta sont des schémas numériques à un pas qui permettent de résoudre les équations différentielles ordinaires. Elles font parties des méthodes les plus populaires de par leur facilité de mise en œuvre et leur précision. C'est Carle Runge et Martin Kutta qui, au début du XX^{e} siècle, ont inventé ces méthodes.

On trouvera dans la référence [3] une description détaillée de la méthode.

Deux algorithmes sont assez utilisés : ceux de Runge-Kutta d'ordre 2 et 4. On donne ci-dessous l'algorithme de la méthode

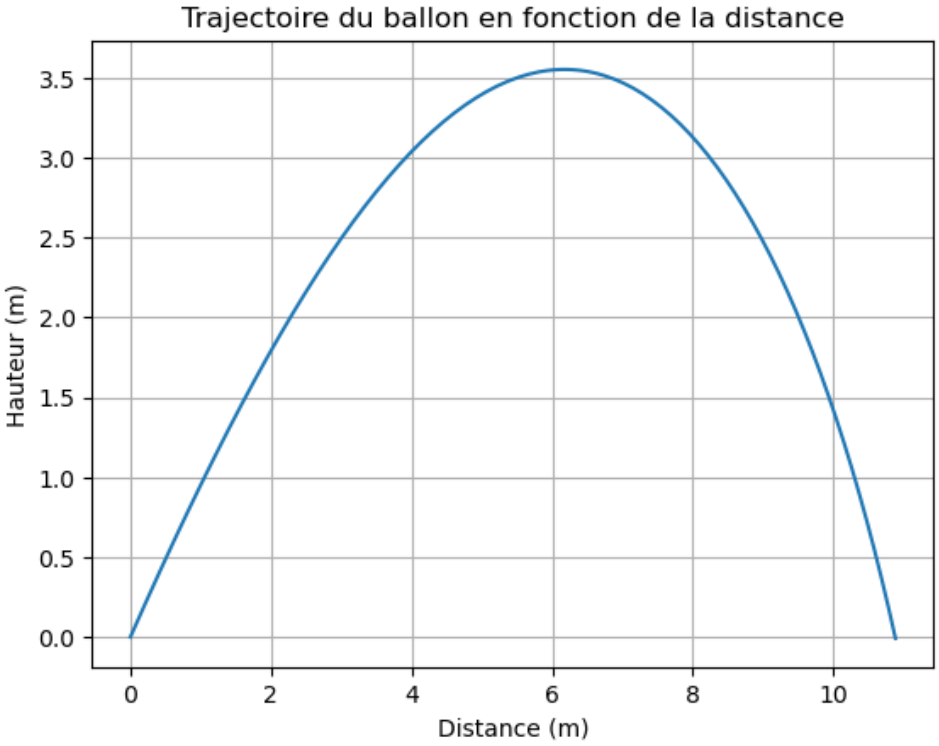
1. Initialisation du pas h, de la durée T.
2. Initialisation des conditions initiales : $t = 0$ et $\mathbf{y} = \mathbf{y}(0)$.
3. Définition de la fonction $f(t, y)$.
4. Tant que $t \leq T$ faire :
   a. Calcul de $k_1 = f(t, y)$.
   b. Calcul de $k_2 = f(t + h/2, y + hk_1/2)$.
   c. Calcul de $k_3 = f(t + h/2, y + hk_2/2)$.
   d. Calcul de $k_4 = f(t + h, y + hk_3)$.
   e. $y = y + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$; $t = t + h$.
   f. Enregistrement des données

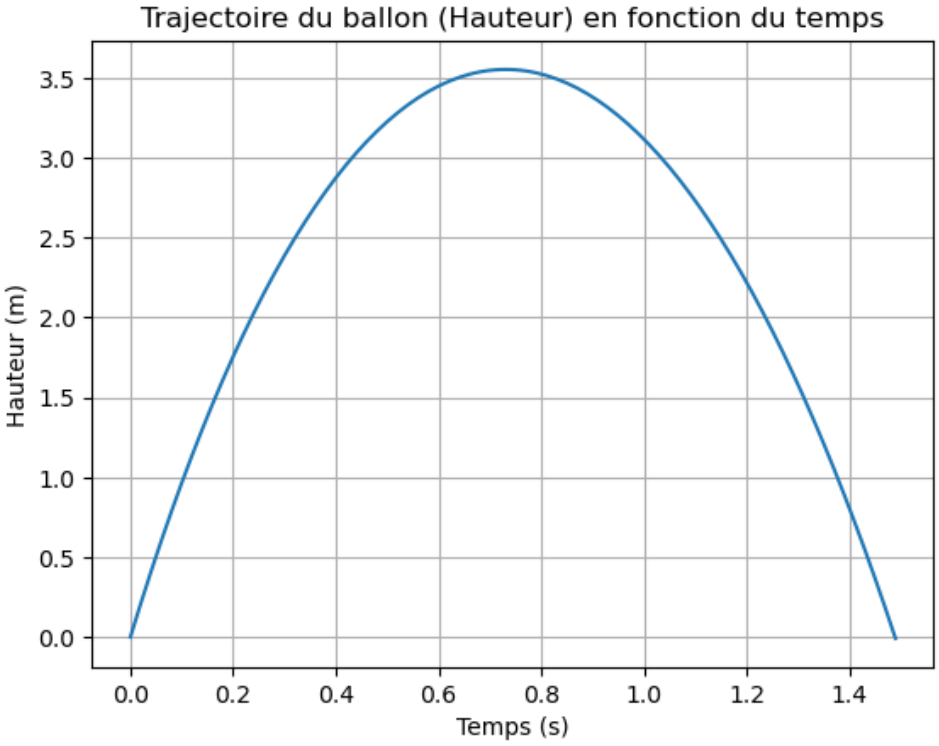
Le programme python de résolution par la méthode de Runge-Kutta est le suivant :

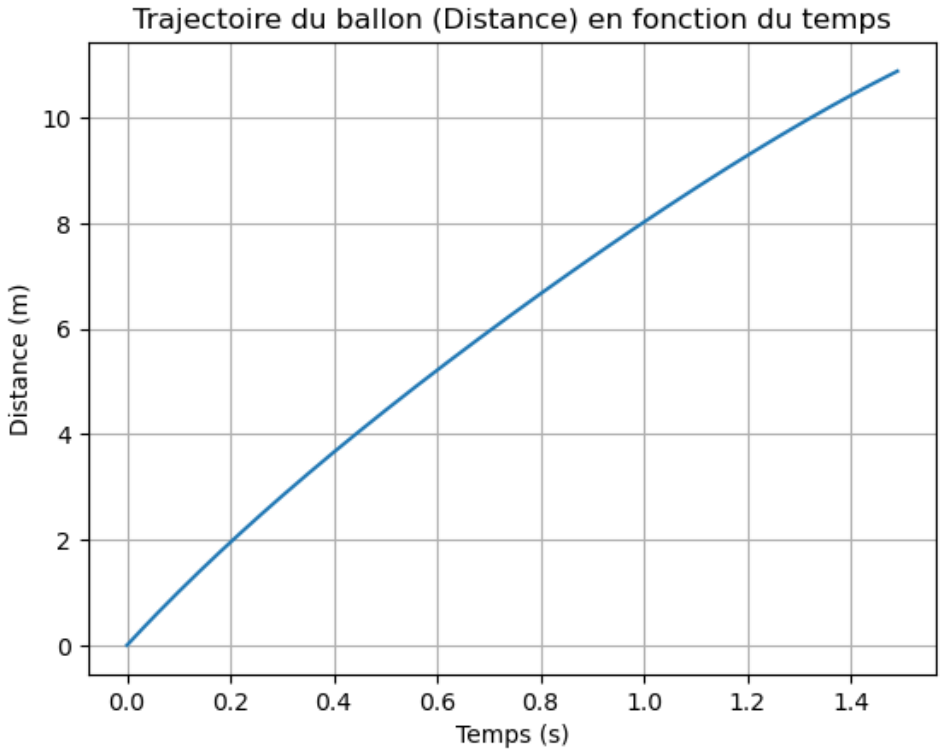
```
import numpy as np
import matplotlib.pyplot as plt

def calculate_trajectory(m, g, rho, A, Cd, v0, theta, dt, t_max):
    # Initialisation des listes de données
    t_list = [0]
    x_list = [0]
    y_list = [0]
    vx_list = [v0 * np.cos(theta)]
    vy_list = [v0 * np.sin(theta)]

    while (t_list[-1] < t_max) and (y_list[-1] >= 0.):

        # Calcul des forces
        Fg = m * g
        Fd = 0.5 * rho * A * Cd * np.sqrt(vx_list[-1]**2 + vy_list[-1]**2)**2

        # Calcul des accélérations
        ax = -Fd * np.cos(theta) / m
        ay = -Fd * np.sin(theta) / m - g

        # Méthode de Runge-Kutta d'ordre 4
        k1x = vx_list[-1] * dt
        k1y = vy_list[-1] * dt
        k1vx = ax * dt
        k1vy = ay * dt

        k2x = (vx_list[-1] + 0.5 * k1vx) * dt
        k2y = (vy_list[-1] + 0.5 * k1vy) * dt
        k2vx = ax * dt
        k2vy = ay * dt

        k3x = (vx_list[-1] + 0.5 * k2vx) * dt
        k3y = (vy_list[-1] + 0.5 * k2vy) * dt
        k3vx = ax * dt
        k3vy = ay * dt

        k4x = (vx_list[-1] + k3vx) * dt
        k4y = (vy_list[-1] + k3vy) * dt
        k4vx = ax * dt
        k4vy = ay * dt

        # Calcul des vitesses et des positions
        vx = vx_list[-1] + (1/6) * (k1vx + 2*k2vx + 2*k3vx + k4vx)
        vy = vy_list[-1] + (1/6) * (k1vy + 2*k2vy + 2*k3vy + k4vy)
        x = x_list[-1] + (1/6) * (k1x + 2*k2x + 2*k3x + k4x)
        y = y_list[-1] + (1/6) * (k1y + 2*k2y + 2*k3y + k4y)
```

```
        # Ajout des données à la liste
        t_list.append(t_list[-1] + dt)
        x_list.append(x)
        y_list.append(y)
        vx_list.append(vx)
        vy_list.append(vy)

    return t_list, x_list, y_list

# Paramètres du ballon et du mouvement

m = 0.45  # Masse du ballon en kg
g = 9.81  # Accélération gravitationnelle en m/s^2
rho = 1.2  # Densité de l'air en kg/m^3
A = 0.1  # Surface de référence du ballon en m^2
Cd = 0.47  # Coefficient de traînée du ballon

v0 = 15  # Vitesse initiale en m/s
theta = np.pi / 4  # Angle de tir en radians

dt = 0.01  # Pas de temps en secondes
t_max = 10  # Temps total de simulation en secondes


# Calcul de la trajectoire

t, x, y = calculate_trajectory(m, g, rho, A, Cd, v0, theta, dt, t_max)

# Tracé de la trajectoire Hauteur = f(Distance)

plt.plot(x, y)
plt.xlabel('Distance (m)')
plt.ylabel('Hauteur (m)')
plt.title('Trajectoire du ballon en fonction de la distance')
plt.grid(True)
plt.show()

# Tracé de la trajectoire Hauteur = f(Temps)

plt.plot(t, y)
plt.xlabel('Temps (s)')
plt.ylabel('Hauteur (m)')
plt.title('Trajectoire du ballon (Hauteur) en fonction du temps')
plt.grid(True)
plt.show()

# Tracé de la trajectoire Distance = f(Temps)

plt.plot(t, x)
plt.xlabel('Temps (s)')
plt.ylabel('Distance (m)')
plt.title('Trajectoire du ballon (Distance) en fonction du temps')
```

```
plt.grid(True)
plt.show()
```







Nota : Les différences sont minimes dans notre cas entre l'algorithme d'Euler et celui de Runge-Kutta d'ordre 4. On en déduit qu'ici, le schéma d'Euler est suffisant et bien convergé.

8. Modèle amélioré

Les équations présentées précédemment sont une approximation simplifiée du mouvement du ballon. Pour prendre en compte les effets complémentaires tels que la pression de l'air à l'intérieur du ballon et les effets aérodynamiques spécifiques, il est nécessaire d'utiliser des modèles plus avancés et détaillés.

Les effets complémentaires pris en compte sont :

1. La pression de l'air à l'intérieur du ballon : Pour prendre en compte la pression de l'air à l'intérieur du ballon, on va utiliser l'équation de l'état des gaz parfaits. Cela permet d'estimer la pression interne en fonction du volume et de la température à l'intérieur du ballon. Il sera également nécessaire de tenir compte des variations de pression dues à l'expansion ou à la compression de l'air à l'intérieur du ballon lorsque celui-ci se déplace.
2. Effets aérodynamiques spécifiques : Si on dispose de données sur les caractéristiques aérodynamiques spécifiques du ballon utilisé, telles que le coefficient de traînée (Cd) et le coefficient de portance (Cl), on peut les incorporer dans les calculs. On doit alors modifier l'équation de la force de traînée pour utiliser le Cd spécifique du ballon. De plus, on peut ajouter une force de portance en utilisant le Cl et l'angle d'attaque appropriés pour prendre en compte les effets aérodynamiques.

Il est important de noter que la modélisation précise de ces effets complémentaires est complexe et nécessite souvent des données expérimentales ou des simulations plus avancées. Les modèles simples présentés précédemment sont souvent utilisés comme une première approximation dans des situations où les effets complémentaires ne sont pas significatifs ou lorsque des données détaillées ne sont pas disponibles.

Si on dispose de données spécifiques sur le ballon que l'on souhaite modéliser, on peut les intégrer dans les calculs en adaptant les équations en conséquence. Cependant, cela peut nécessiter une connaissance approfondie des propriétés du ballon et des méthodes de modélisation appropriées.

Certainement ! Cependant, il est important de noter que la modélisation précise des effets complémentaires tels que la pression de l'air à l'intérieur du ballon et les effets aérodynamiques spécifiques nécessite des données expérimentales ou des simulations plus avancées. Par conséquent, le programme suivant vous donne une idée générale de la façon dont ces effets pourraient être pris en compte, mais il ne fournit pas une modélisation détaillée.

Voici un exemple de script Python qui incorpore la force de traînée et la force de portance pour prendre en compte les effets aérodynamiques :

```
import math
import numpy as np
import matplotlib.pyplot as plt

# Constantes et paramètres initiaux
g = 9.81 # m/s^2
rho = 1.2 # kg/m^3 - densité de l'air
diametre = 0.22 # m - diamètre du ballon
masse = 0.43 # kg - masse du ballon
rayon = diametre / 2 # rayon du ballon
section = math.pi * rayon**2 # section transversale du ballon
coeff_trainee = 0.47 # coefficient de traînée du ballon
coeff_portance = 0.2 # coefficient de portance du ballon
vitesse_initiale = 20.0 # m/s - vitesse initiale
angle_lancement_deg = 45.0 # degrés - angle de lancement initial
hauteur_initiale = 1.5 # m - hauteur initiale

# Convertir l'angle de lancement initial en radians
angle_lancement_rad = math.radians(angle_lancement_deg)

# Fonction pour calculer la force de traînée
def force_trainee(vitesse, coeff_trainee, section, densite_air):
    force = -0.5 * coeff_trainee * section * densite_air * vitesse**2
    return force

# Fonction pour calculer la force de portance
def force_portance(vitesse, angle_attaque, coeff_portance, section, densite_air):
    force = 0.5 * coeff_portance * section * densite_air * vitesse**2 * math.sin(angle_attaque)
    return force
```

```
# Fonction pour calculer les dérivées de la vitesse et de la position
def derivees(t, Y):
    vitesse = Y[1]
    position = Y[0]

    # Calculer les forces
    force_gravite = -masse * g
    force_trainee = force_trainee(vitesse, coeff_trainee, section, rho)
    angle_attaque = math.atan2(vitesse, max(0.1, Y[1]))
    force_portance = force_portance(vitesse, angle_attaque, coeff_portance, section, rho)

    # Calculer les dérivées
    derivee_position = vitesse
    derivee_vitesse = (force_gravite + force_trainee + force_portance) / masse

    return np.array([derivee_position, derivee_vitesse])

# Conditions initiales
Y0 = np.array([hauteur_initiale, vitesse_initiale * math.cos(angle_lancement_rad),
            0.0, vitesse_initiale * math.sin(angle_lancement_rad)])

# Discrétisation temporelle
dt = 0.01 # s
t = np.arange(0.0, 10.0, dt)

# Résolution avec la méthode de Runge-Kutta
Y = np.zeros((len(t), len(Y0)))
Y[0,:] = Y0
for i in range(len(t)-1):
    k1 = dt * derivees(t[i], Y[i,:])
    k2 = dt * derivees(t[i] + 0.5*dt, Y[i,:] + 0.5*k1)
    k3 = dt * derivees(t[i] + 0.5*dt, Y[i,:] + 0.5*k2)
    k4 = dt * derivees(t[i] + dt, Y[i,:] + k3)
    Y[i+1,:] = Y[i,:] + (1/6) * (k1 + 2*k2 + 2*k3 + k4)

# Tracé de la trajectoire
x = Y[:,0]
y = Y[:,2]
plt.plot(x, y)
plt.xlabel('Distance (m)')
plt.ylabel('Hauteur (m)')
plt.title('Trajectoire du ballon avec effets aérodynamiques')
plt.grid(True)
plt.show()
```

Ce programme utilise la méthode de Runge-Kutta d'ordre 4 pour résoudre les équations du mouvement en tenant compte de la force de traînée et de la force de portance. Les fonctions `force_trainee` et `force_portance` calculent respectivement la force de traînée et la force de portance en fonction de la vitesse, de l'angle d'attaque, des coefficients de traînée et de portance, de la section

transversale du ballon et de la densité de l'air. Ces forces sont ensuite utilisées dans la fonction `derivees` pour calculer les dérivées de la vitesse et de la position.

Le programme trace ensuite la trajectoire du ballon en utilisant les coordonnées x et y calculées lors de la résolution.

Il est possible d'ajuster les paramètres du ballon selon ses besoins et d'ajouter d'autres effets aérodynamiques spécifiques au ballon en modifiant les fonctions `force_trainee` et `force_portance` en conséquence.

9. Effet Magnus

L'effet Magnus est un phénomène physique qui se produit lorsque l'air en mouvement interagit avec un objet en rotation, tel qu'un ballon de football. Il crée une force de portance qui influence la trajectoire du ballon.

L'effet Magnus est principalement dû à deux forces : la force de portance et la force de traînée. La force de portance est la force perpendiculaire au mouvement relatif de l'air par rapport à l'objet en rotation, tandis que la force de traînée est la force opposée au mouvement relatif.

Les équations de l'effet Magnus sont les suivantes :

1. Force de portance :

La force de portance (Fp) exercée sur le ballon est donnée par l'équation suivante :

$$Fp = 0.5 \cdot \rho \cdot A \cdot Cm \cdot (v \cdot \omega)$$

où :

- ρ est la masse volumique de l'air,
- A est la section transversale du ballon,
- Cm est le coefficient de portance du ballon, qui dépend de la rugosité de la surface et du taux de rotation du ballon,
- v est la vitesse du ballon par rapport à l'air,
- ω est la vitesse de rotation du ballon.

2. Force de traînée :

La force de traînée (Fd) exercée sur le ballon est donnée par l'équation suivante :

$$Fd = 0.5 \cdot \rho \cdot A \cdot Cd \cdot (v^2)$$

où :

- ρ est la masse volumique de l'air,
- A est la section transversale du ballon,
- Cd est le coefficient de traînée du ballon, qui dépend de la forme du ballon et de la rugosité de sa surface,
- v est la vitesse du ballon par rapport à l'air.

Ces équations prennent en compte l'effet de la masse volumique de l'air, la surface du ballon, les coefficients de portance et de traînée, ainsi que les vitesses du ballon et de rotation. La force de portance agit perpendiculairement à la direction du mouvement relatif de l'air, tandis que la force de traînée s'oppose au mouvement.

En utilisant ces équations dans le programme Python suivant, il est possible d'estimer la trajectoire du ballon en tenant compte de l'effet Magnus.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat May 20 07:53:23 2023


@author: b89637
"""


import math
import matplotlib.pyplot as plt


# Constantes

g = 9.81 # Accélération due à la gravité (m/s^2)
rho = 1.2 # Masse volumique de l'air (kg/m^3)
A = math.pi * (0.11 / 2)**2 # Section transversale du ballon (m^2)
C_d = 0.47 # Coefficient de traînée du ballon (adimensionnel)
C_m = 0.25 # Coefficient de portance du ballon (adimensionnel)
m = 0.43 # Masse du ballon (kg)
dt = 0.01 # Pas de temps (s)

# Conditions initiales

v0 = 30 # Vitesse initiale (m/s)
theta = math.radians(30) # Angle de tir initial (radians)
omega = 10 # Vitesse de rotation du ballon (rad/s)
```

```
# Fonctions pour le calcul de l'effet Magnus

def magnus_force(v, omega):

return 0.5 * rho * A * C_m * (v * omega)

def drag_force(v):

return 0.5 * rho * A * C_d * (v**2)

def calculate_trajectory(v0, theta, omega, dt):

# Conversion des conditions initiales en composantes x et y

vx0 = v0 * math.cos(theta)
vy0 = v0 * math.sin(theta)

# Variables pour la boucle de simulation

x = 0.0
y = 0.0
vx = vx0
vy = vy0

# Listes pour enregistrer les coordonnées (x, y) à chaque pas de temps

x_values = [x]
y_values = [y]
dt_values = [0.0]

while y >= 0.0:

# Calcul des forces

magnus = magnus_force(vx, omega)
drag = drag_force(vx)
fx = -drag
fy = -m * g + magnus

# Mise à jour des vitesses et des positions

ax = fx / m
ay = fy / m
vx += ax * dt
vy += ay * dt
x += vx * dt
y += vy * dt

# Ajout des coordonnées à la liste

x_values.append(x)
y_values.append(y)
```

```
dt_values.append(dt_values[-1]+dt)

return dt_values, x_values, y_values

# Appel de la fonction de calcul de trajectoire

t_traj, x_traj, y_traj = calculate_trajectory(v0, theta, omega, dt)

# Affichage des résultats

for i in range(len(x_traj)):

print(f"Temps: {i * dt:.2f}s, Position: ({x_traj[i]:.2f}m, {y_traj[i]:.2f}m)")

# Tracé de la trajectoire Hauteur = f(Distance)

plt.plot(x_traj, y_traj)
plt.xlabel('Distance (m)')
plt.ylabel('Hauteur (m)')
plt.title('Trajectoire du ballon avec effets aérodynamiques en fonction de la distance')
plt.grid(True)
plt.show()

# Tracé de la trajectoire Hauteur = f(Temps)

plt.plot(t_traj, y_traj)
plt.xlabel('Temps (s)')
plt.ylabel('Hauteur (m)')
plt.title('Trajectoire du ballon (Hauteur) avec effets aérodynamiques en fonction du temps')
plt.grid(True)
plt.show()

# Tracé de la trajectoire Distance = f(Temps)

plt.plot(t_traj, x_traj)
plt.xlabel('Temps (s)')
plt.ylabel('Distance (m)')
plt.title('Trajectoire du ballon (Distance) avec effets aérodynamiques en fonction du temps')
plt.grid(True)
plt.show()
```

10. Conclusions

Comme dans d'autres sports, la science et les techniques ont permis d'améliorer les performances sportives. Les Mathématiques, et la Physique, peuvent nous aider à modéliser la trajectoire d'un ballon et de mieux comprendre les différents effets qui interviennent.
Mais c'est surtout le sens du jeu, les qualités athlétiques et l'entrainement qui permettent de marquer des buts mythiques qui feront exploser de joie les supporters et parfois même tout un pays !

11. Références

[1] https://openai.com/product/gpt-4
[2] https://tpe-ballondefootball.jimdofree.com/
[3] https://femto-physique.fr/analyse-numerique/euler.php
[4] https://www.studyrama.com/revision-examen/bac/fiches-de-revision-du-bac/serie-generale/terminale/sciences-de-l-ingenieur