
LIVRABLE 3
PROJET FAST & FURIOUS
M ÜG 0104

SOMMAIRE

I. Contexte

Dom Toretto a été mis au défi par Owen Shawn de remporter une course appelée le « Double Loop Dare », le vainqueur repartira avec comme prix une Mazda RX-7.

Cette course se déroulera sur un circuit divisé en plusieurs parties :

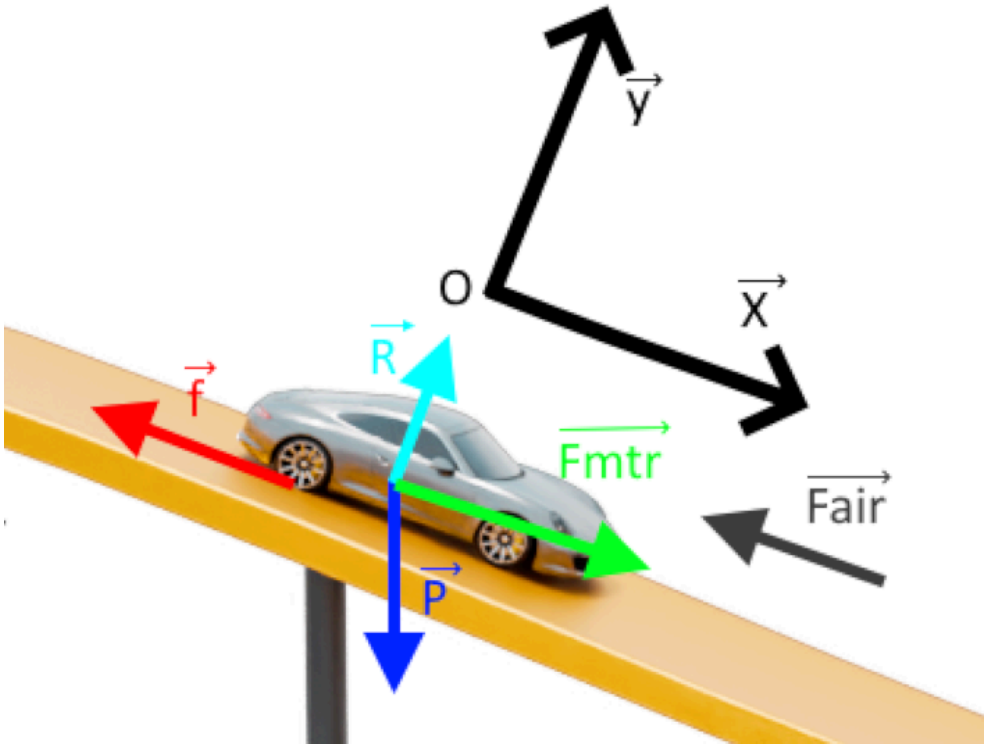
Nous aurons 3 surfaces planes horizontales , une surface plane inclinée, un looping et un ravin divisant deux des surfaces planes horizontales.

Dans ce troisième livrable, il nous a été demandé de résoudre les équation différentielle du mouvement de la voiture sur les différentes parties du circuit et nous verrons cela à travers les différentes étapes ci-dessous :

- La résolution des équations du mouvement pour chaque étape à l'aide d'un code python paramétrable

II. PENTE

Le rappel du schéma et des équations du mouvement (livrable 2)



forces:

$$\vec{f} \Rightarrow force\ de\ frottement\ avec\ le\ sol$$

$$\vec{R} \Rightarrow normale\ R$$

$$\vec{P} \Rightarrow poid$$

$$\vec{Fmtr} \Rightarrow force\ motrice\ du\ moteur\ de\ la\ voiture$$

$$\vec{Fair} \Rightarrow force\ de\ frottement\ avec\ l'air$$

sur l'axe (x)

$$m\frac{dv}{dt} = mg.sin(\alpha) + m.am - \mu.mg.cos(\alpha) - k.v^2$$

sur l'axe (y)

$$ma_y = -m.g.cos(\alpha) + R = 0$$

Le calcul de la vitesse en bas de la pente

un code python s'occupe de la résolution de l'équation différentielle ci dessous:

$$am \Rightarrow Accélération\ moyenne$$

$\alpha \Rightarrow angle\ de\ la\ pente$

$$\frac{dv}{dt} = g.sin(\alpha) + am - \mu.g.cos(\alpha) - \frac{k.v^2}{m}$$

en utilisant la méthode d'euler on obtient deux graphs celui de la vitesse v(t) et celui de la position x(t) on peut donc déterminer à partir de cela la vitesse de sortie de pente qui est la valeur de v(t) lors de x(t) = 31 m (longueur de la pente)

Voiture	Vitesse de sortie (Pente)
Modèle 1 : Dodge Charger R/T, 1970	17.07m/s
Modèle 2 : Toyota Supra Mark IV, 1994	16.92m/s
Modèle 3 : Chevrolet Yenko Camaro 1969	17.43m/s
Modèle 4 : Mazda RX-7 FD	17.28m/s
Modèle 5 : Nissan Skyline GTR-R34, 1999	18.29m/s
Modèle 6 : Mitsubishi Lancer Evolution VII	16.91m/s

code (pente):

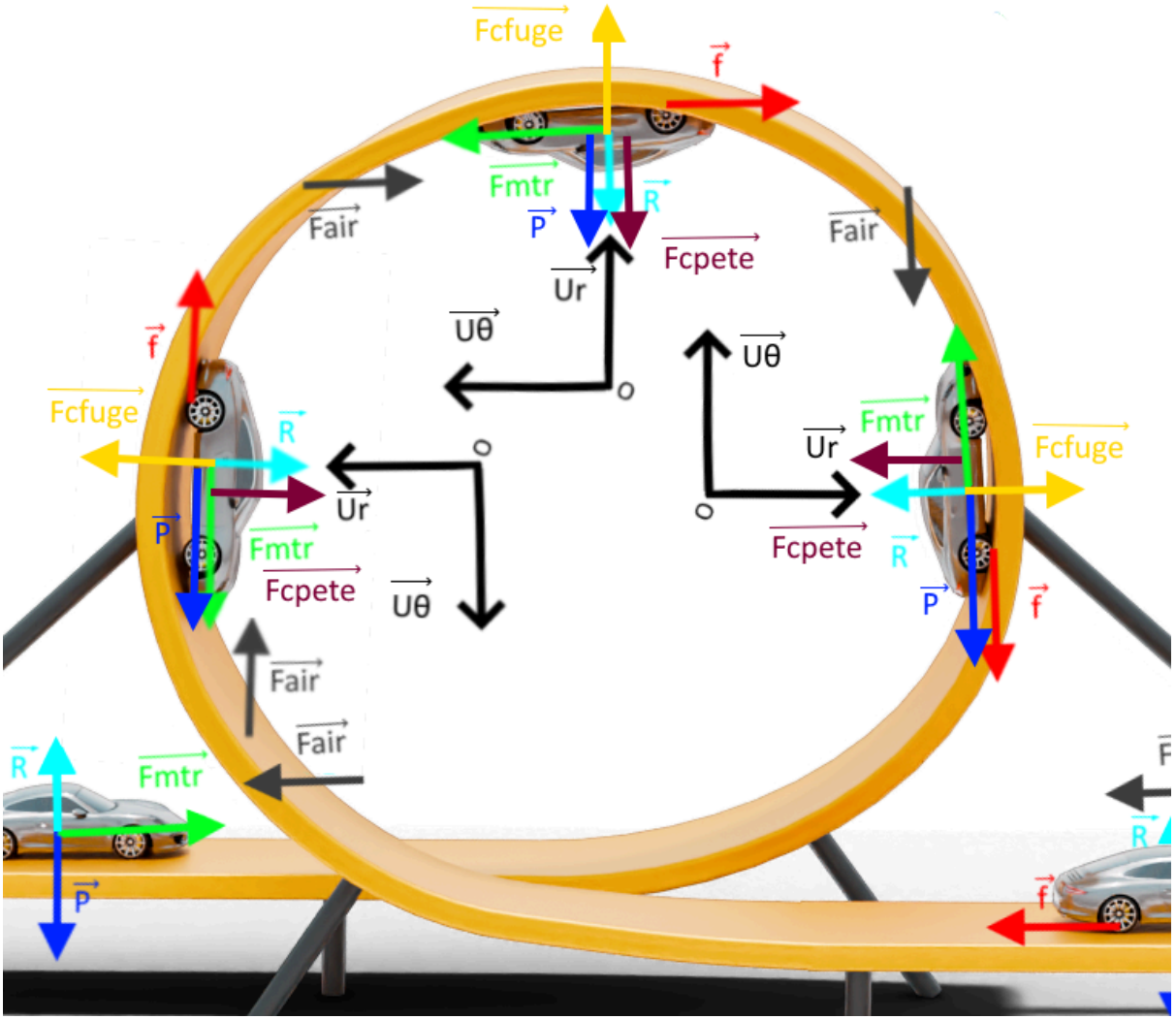
```
import numpy as np
from scipy.integrate import odeint
import matplotlib

matplotlib.use('TkAgg')  # Use 'TkAgg' or 'Qt5Agg' as a backend
import matplotlib.pyplot as plt
import Loop


def RunModule(Masse, Largeur, Hauteur, Longueur, Cx, Cz, mu,
AccMoyenne,nos_active, nos_location, aileron_active):
    g = 9.806
    ro = 1.292
    boost = 1
    if nos_active == True and nos_location == "la pente":
        boost = 1.3

    A = Largeur * Hauteur
    anglePente = np.radians(3.7)

    V0 = 0
    X0 = 0
    y0 = [V0, X0]

    def SetDiffEq(y, t, g, ro, Masse, A, Cx, mu, AccMoyenne,
anglePente):
        v = y[0]
        x = y[1]
        dv_dt = (g * np.sin(anglePente) + AccMoyenne*boost
                 - mu * g * np.cos(anglePente)
                 - (0.5 * Cx * ro * A * v ** 2) / Masse)
        dx_dt = v
```

```
        return [dv_dt, dx_dt]

    t = np.linspace(0, 5, 5000)

    solution = odeint(SetDiffEq, y0, t, args=(g, ro, Masse, A, Cx,
mu, AccMoyenne, anglePente))

    v = solution[:, 0]
    x = solution[:, 1]

    threshold = 31
    idx_stop = np.argmax(x >= threshold)
    t_stop = t[idx_stop]
    v_stop = v[idx_stop]
    x_stop = x[idx_stop]

    temps_31m = t_stop

    print(f"Vitesse de la voiture à x = {x_stop:.2f} m :
{v_stop:.2f} m/s")
    print(f"Temps pour atteindre x = {x_stop:.2f} m :
{temps_31m:.2f} s")

    Loop.RunModule(Masse, Largeur, Hauteur, Longueur, Cx, Cz, mu,
AccMoyenne, v_stop, temps_31m,nos_active, nos_location,
aileron_active)

    plt.figure(1, figsize=(12, 6))
    plt.subplot(2, 1, 1)
    plt.plot(t[:idx_stop + 1], v[:idx_stop + 1], label='Vitesse
(m/s)', color='blue')
    plt.xlabel('Temps (s)')
    plt.ylabel('Vitesse (m/s)')
    plt.title('Évolution de la vitesse')
    plt.grid()
    plt.legend()

    plt.subplot(2, 1, 2)
    plt.plot(t[:idx_stop + 1], x[:idx_stop + 1], label='Position
(m)', color='red')
    plt.xlabel('Temps (s)')
    plt.ylabel('Position (m)')
    plt.title('Évolution de la position')
    plt.grid()
    plt.legend()

    plt.figtext(0.5, 0.01,
```

```
                f"Vitesse de sortie : {v_stop:.2f} m/s | Temps pour
atteindre 31m : {temps_31m:.2f} secondes",
                fontsize=10, ha="center", va="center")

    plt.gcf().canvas.manager.set_window_title("Simulation Pente")
    plt.tight_layout()
    plt.show(block=False)
```

III. LOOPING

Le rappel du schéma et des équations du mouvement (livrable 2)



forces:

$\vec{f} \Rightarrow force\ de\ frottement\ avec\ le\ sol$

$\vec{R} \Rightarrow normale\ R$

$\vec{P} \Rightarrow poid$

$$\vec{Fmtr} \Rightarrow force\ motrice\ du\ moteur\ de\ la\ voiture$$

$$\vec{Fair} \Rightarrow force\ de\ frottement\ avec\ l'air$$

$$\vec{Fcfuge} \Rightarrow force\ centrifuge$$

$$\vec{Fcpete} \Rightarrow force\ centripète$$

projection sur $\vec{ur}$:

$$-r.\dot{\theta}^{2} = g.cos(\theta) - \frac{R}{m}$$

projection sur $\vec{u\theta}$:

$$m(r.\ddot{\theta}) = -m.g.sin(\theta) + m.am - \mu.R - k.v^{2}$$

$$m(r.\ddot{\theta}) = -m.g(\mu.cos(\theta) + sin(\theta)) - \dot{\theta}^{2}.(\mu.m.r + k.r^{2}) + m.am$$

Le tracé de la vitesse de la voiture au cours du temps

un code python s'occupe de la résolution de l'équation différentielle ci dessous:

$$\ddot{\theta} = \frac{-m.g(\mu.cos(\theta)+sin(\theta))-\dot{\theta}^{2}.(\mu.m.r+k.r^{2})+m.am}{m.r}$$

$$\frac{d\omega}{dt} = \frac{-m.g(\mu.cos(\theta)+sin(\theta))-\dot{\theta}^{2}.(\mu.m.r+k.r^{2})+m.am}{m.r}$$

grâce à la méthode d'euler on obtient le graph de la vitesse angulaire de la voiture au cours du temps $\omega(t)$

calculation de la vitesse minimale a l'entré du looping pour faire le tour sans tomber

la vitesse minimale d'entré dans le looping est calculé par le code python en testant plusieur valeurs de vitesse et vérifier à chaque fois si $\theta(t)$ atteint 2π (tour complet du looping)

Voiture	Vitesse min (Looping)
Modèle 1 : Dodge Charger R/T, 1970	9.647m/s
Modèle 2 : Toyota Supra Mark IV, 1994	9.836m/s
Modèle 3 : Chevrolet Yenko Camaro 1969	9.238m/s
Modèle 4 : Mazda RX-7 FD	9.433m/s
Modèle 5 : Nissan Skyline GTR-R34, 1999	8.139m/s
Modèle 6 : Mitsubishi Lancer Evolution VII	9.842m/s

calculation de la vitesse de sortie du looping

la vitesse de sortie est la valeur de vitesse angulaire $\omega(t)$ a l'instant ou $\theta(t) = 2\pi$ (tour complet donc la voiture sort du looping)

Voiture	Vitesse de sortie (Looping)
Modèle 1 : Dodge Charger R/T, 1970	19.428m/s
Modèle 2 : Toyota Supra Mark IV, 1994	19.311m/s
Modèle 3 : Chevrolet Yenko Camaro 1969	19.731m/s
Modèle 4 : Mazda RX-7 FD	19.606m/s
Modèle 5 : Nissan Skyline GTR-R34, 1999	20.476m/s
Modèle 6 : Mitsubishi Lancer Evolution VII	19.294

comparaison de la vitesse minimale pour franchir le looping avec la vitesse de sortie de la pente

Voiture	Vitesse de sortie (Pente)		Vitesse min (Looping)
Modèle 1 : Dodge Charger R/T, 1970	17.07m/s	>	9.647m/s
Modèle 2 : Toyota Supra Mark IV, 1994	16.92m/s	>	9.836m/s
Modèle 3 : Chevrolet Yenko Camaro 1969	17.43m/s	>	9.238m/s
Modèle 4 : Mazda RX-7 FD	17.28m/s	>	9.433m/s
Modèle 5 : Nissan Skyline GTR-R34, 1999	18.29m/s	>	8.139m/s
Modèle 6 : Mitsubishi Lancer Evolution VII	16.91m/s	>	9.842m/s

possibilité de passer le looping pour chaque voiture

toute les voitures ont des vitesses de sortie "pente" plus grandes que leur vitesse min pour passer le looping donc:

Voiture	possibilité de passer le looping
Modèle 1 : Dodge Charger R/T, 1970	POSSIBLE
Modèle 2 : Toyota Supra Mark IV, 1994	POSSIBLE
Modèle 3 : Chevrolet Yenko Camaro 1969	POSSIBLE
Modèle 4 : Mazda RX-7 FD	POSSIBLE
Modèle 5 : Nissan Skyline GTR-R34, 1999	POSSIBLE
Modèle 6 : Mitsubishi Lancer Evolution VII	POSSIBLE

code (Looping)

```
import numpy as np
import matplotlib
import Ravin
import tkinter as tk
from tkinter import messagebox, ttk
matplotlib.use('TkAgg')  # Use 'TkAgg' or 'Qt5Agg' as a backend
import matplotlib.pyplot as plt
from scipy.integrate import odeint

def RunModule(Masse, Largeur, Hauteur, Longueur, Cx, Cz, mu,
AccMoyenne, V0,T,nos_active, nos_location, aileron_active):
    # Constantes physiques
```

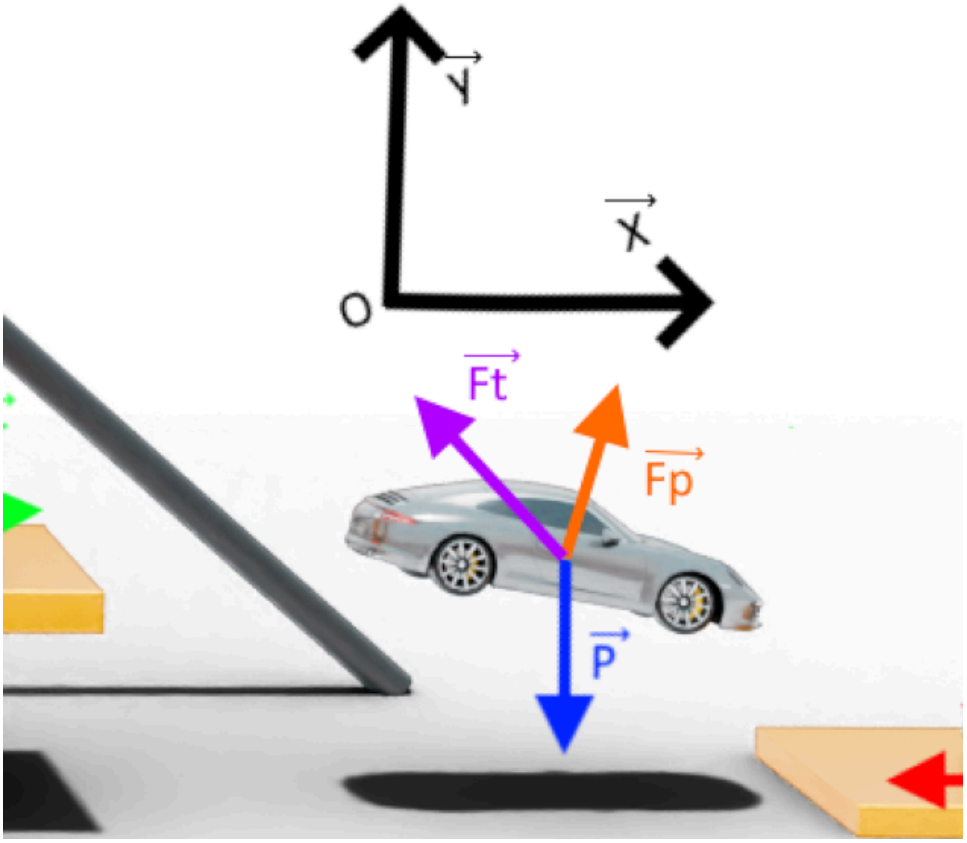
```
    g = 9.806  # Gravité (m/s^2)
    ro = 1.292  # Masse volumique de l'air (kg/m^3)
    drag_area = Largeur * Hauteur  # Surface frontale (m^2)
    boost = 1
    if nos_active == True and nos_location == "le looping":
        boost = 1.3
    # Paramètres du circuit
    rayon_looping = 6  # Rayon du looping (m)
    X0 = 0  # Position initiale
    omega0 = V0 / rayon_looping  # Vitesse angulaire initiale
(rad/s)
    y0 = [0, omega0]  # Conditions initiales [position, vitesse
angulaire]

    # Equation différentielle
    def equation(y, t):
        theta, omega = y  # theta = y[0], omega = y[1]
        calcul = (-Masse * g * (np.sin(theta) + mu * np.cos(theta))
                  - omega ** 2 * (mu * Masse * rayon_looping + 0.5
* Cx * ro * drag_area * rayon_looping ** 2)
                  + AccMoyenne *boost * Masse) / (Masse *
rayon_looping)
        return [omega, calcul]  # [theta_prime, theta_double_prime]

    # Vérification si la voiture complète le looping
    def completes_loop(V0):
        omega0 = V0 / rayon_looping
        sol = odeint(equation, [0, omega0], np.linspace(0, 10,
1000))
        return np.any(sol[:, 0] >= 2 * np.pi)  # Vérifie si theta
atteint 2*pi

    # Recherche binaire pour trouver Vmin
    def find_vmin():
        V_low, V_high = 0, 50  # Bornes initiales de recherche
        tol = 1e-2  # Tolérance pour la convergence
        while V_high - V_low > tol:
            V_mid = (V_low + V_high) / 2
            if completes_loop(V_mid):
                V_high = V_mid
            else:
                V_low = V_mid
        return (V_low + V_high) / 2

    # Calcul de Vmin
    Vmin = find_vmin()

    # Vecteur temps
```

```
    t = np.linspace(0, 5, 1000)
    sol = odeint(equation, y0, t)

    # Détection de l'angle theta = 2*pi
    i = 0
    while i < len(sol) and sol[i][0] < 2 * np.pi:
        i += 1

    # Vitesse finale de la voiture après le looping
    v_looping_af = sol[i - 1][1] * rayon_looping
    print("Vitesse minimale requise : {:.3f} m/s".format(Vmin))
    print("Vitesse initiale : {:.3f} m/s | Vitesse finale : {:.3f}
m/s".format(V0, v_looping_af))
    print("Le temps final du looping est de : {:.3f} s".format(t[i
- 1]))

    # Pass the rounded value to the function

    if V0>= Vmin:
        Ravin.RunModule(Masse, Largeur, Hauteur, Longueur, Cx, Cz,
mu, AccMoyenne, round(v_looping_af, 3),t[i - 1] + T,nos_active,
nos_location, aileron_active)
        # Tracer des résultats
        plt.figure(2, figsize=(10, 6))
        plt.plot(t[:i], sol[:i, 1] * rayon_looping, "olive")
        plt.title("Vitesse de la voiture avec frottements en
fonction du temps")
        plt.xlabel("Temps (s)")
        plt.ylabel("Vitesse (m/s)")
        plt.grid()
        plt.tight_layout()

        # Ajout des informations dans le graphique
        plt.figtext(
            0.5, 0.01,
            "Vitesse minimale requise : {:.3f} m/s | Vitesse
initiale : {:.3f} m/s | Vitesse finale : {:.3f} m/s | Temps final
: {:.3f} s".format(
                Vmin, V0, v_looping_af, t[i - 1]
            ),
            fontsize=10, ha="center", va="center"
        )
        plt.gcf().canvas.manager.set_window_title("Simulation
Looping")
        plt.show(block=False)
    else:
```

```
        messagebox.showerror("Erreur d'Entrée",f"La voiture n'a pas
pu franchir le looping en raison d'une V0 insuffisante : {V0:.2f}
m/s < {Vmin:.2f} m/s")
```

IV- LE SAUT DU RAVIN

Le rappel du schéma et des équations du mouvement (livrable 2)



forces:

$$\vec{P} \Rightarrow poid$$

$$\vec{Ft} \Rightarrow force\ de\ traînée$$

$$\vec{Fp} \Rightarrow force\ de\ portance$$

sur l'axe (x)

$$ma_x = -\frac{1}{2}\rho . C_x . S . v^2$$

$$\frac{dvx}{dt} = \frac{-\frac{1}{2}\rho . C_x . S . v^2}{m}$$

sur l'axe (y)

$$ma_y = -\ m . g\ + \frac{1}{2}\rho . C_z . S . v^2$$

$$\frac{dvy}{dt} = -\ g\ + \frac{\frac{1}{2}\rho.C_z.S.v^2}{m}$$

Le tracé de la trajectoire de la voiture dans le ravin

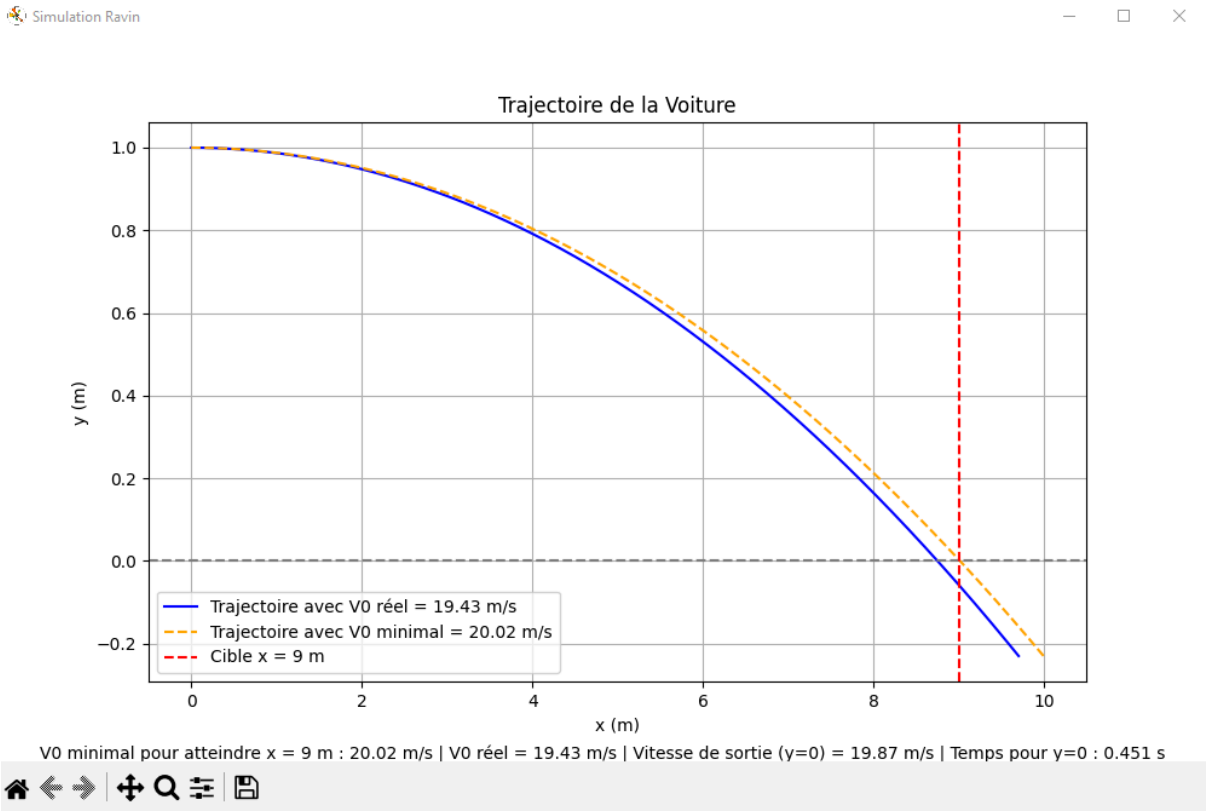
un code python s'occupe de la résolution de l'équation différentielle ci dessous:

$$\frac{dvx}{dt} = \frac{-\frac{1}{2}\rho.C_x.S.v^2}{m}$$

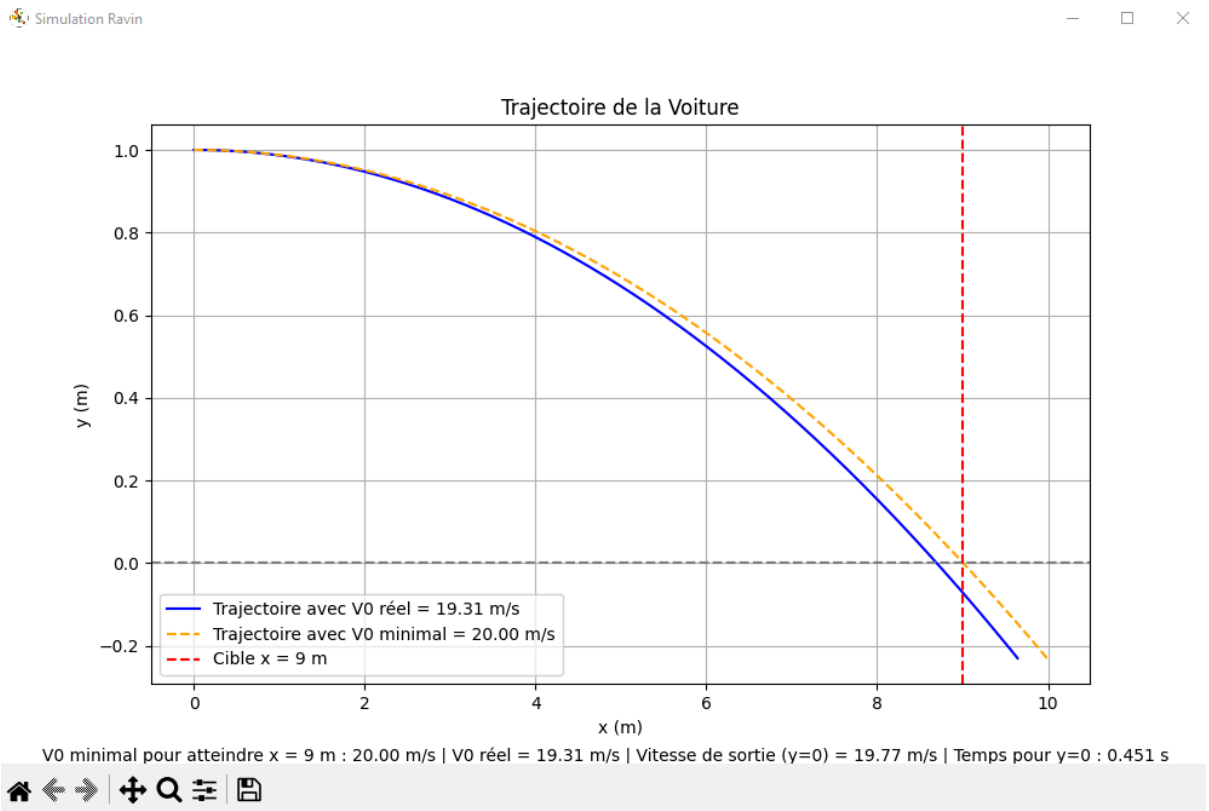
$$\frac{dvy}{dt} = -\ g\ + \frac{\frac{1}{2}\rho.C_z.S.v^2}{m}$$

pour obtenir le graph y(x)

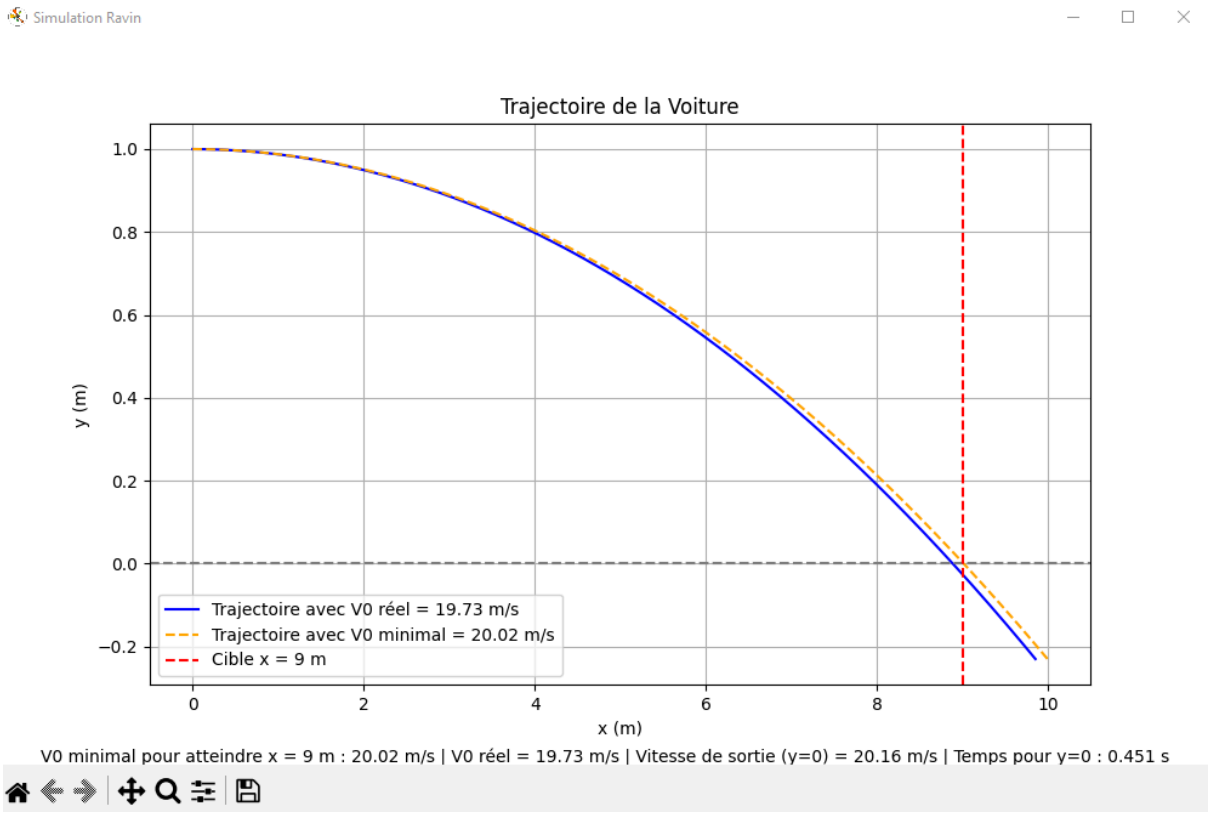
Modèle 1 : Dodge Charger R/T, 1970



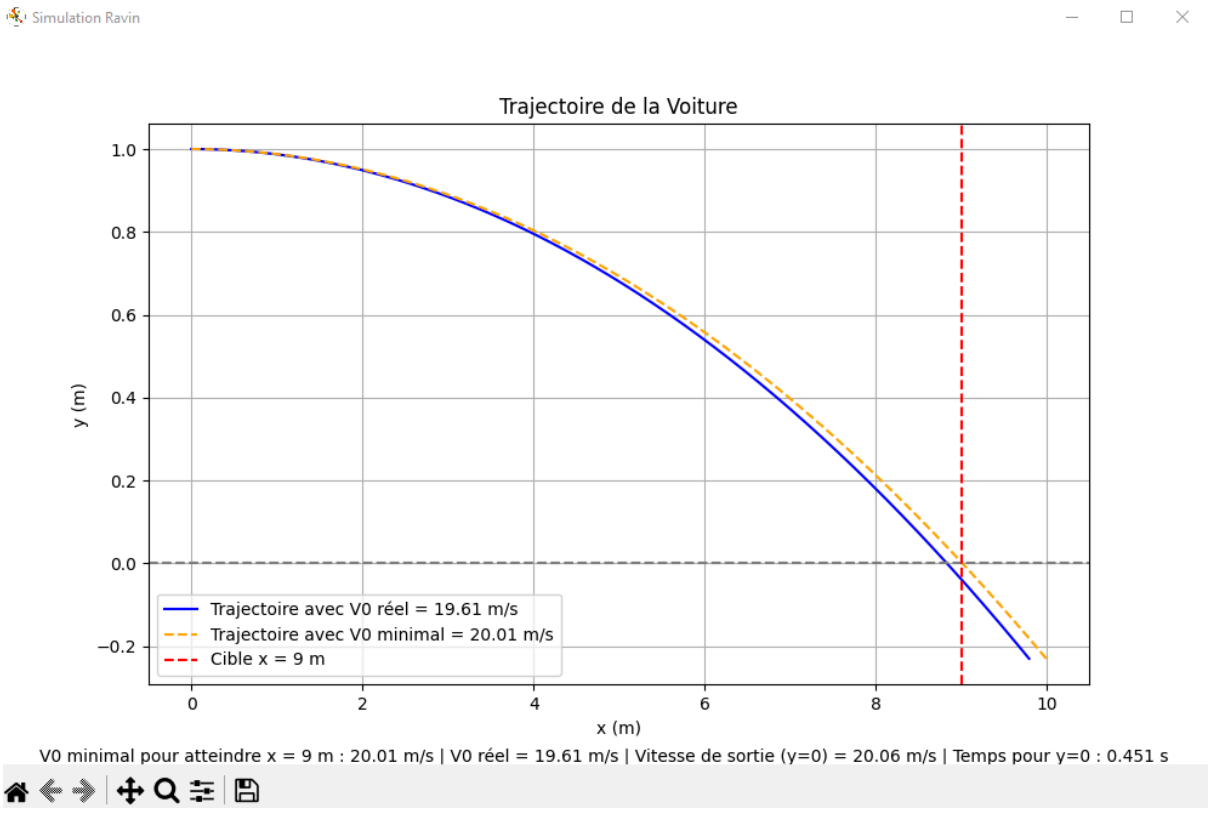
Modèle 2 : Toyota Supra Mark IV, 1994



Modèle 3 : Chevrolet Yenko Camaro 1969

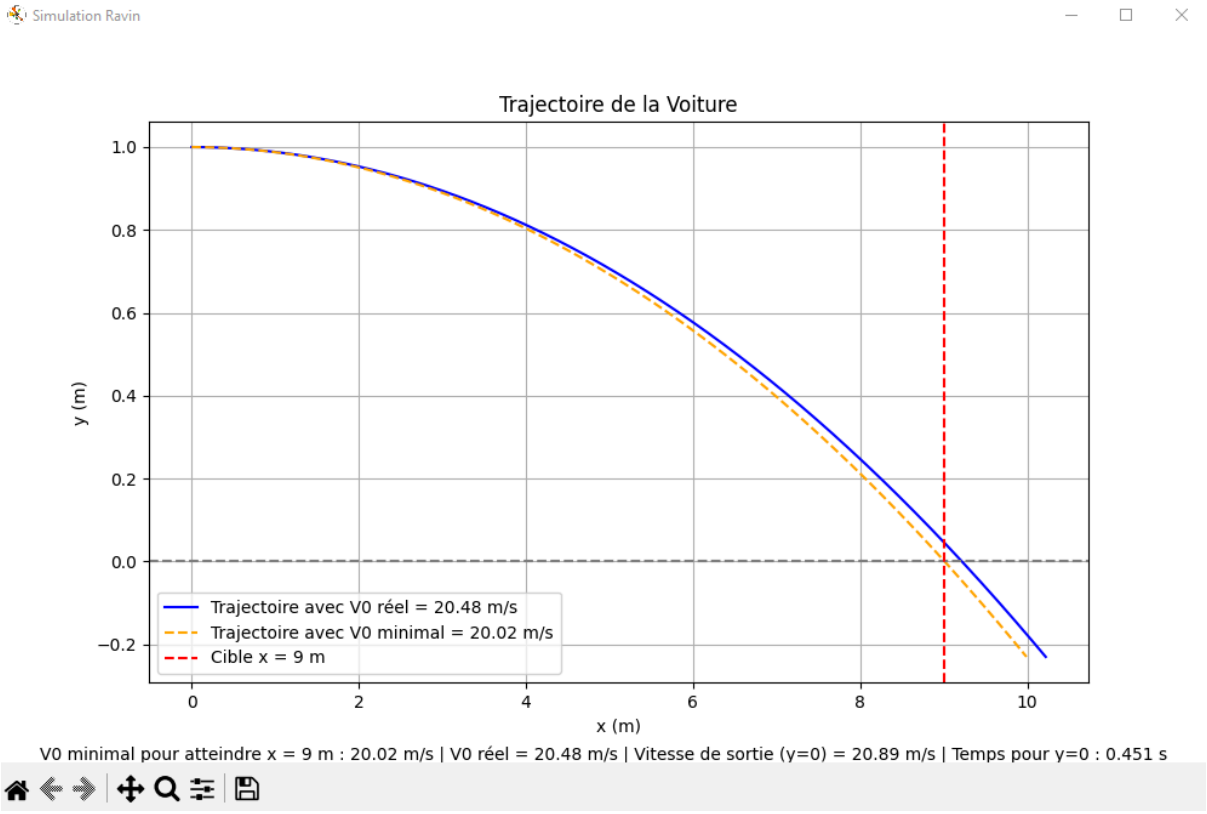


Modèle 4 : Mazda RX-7 FD

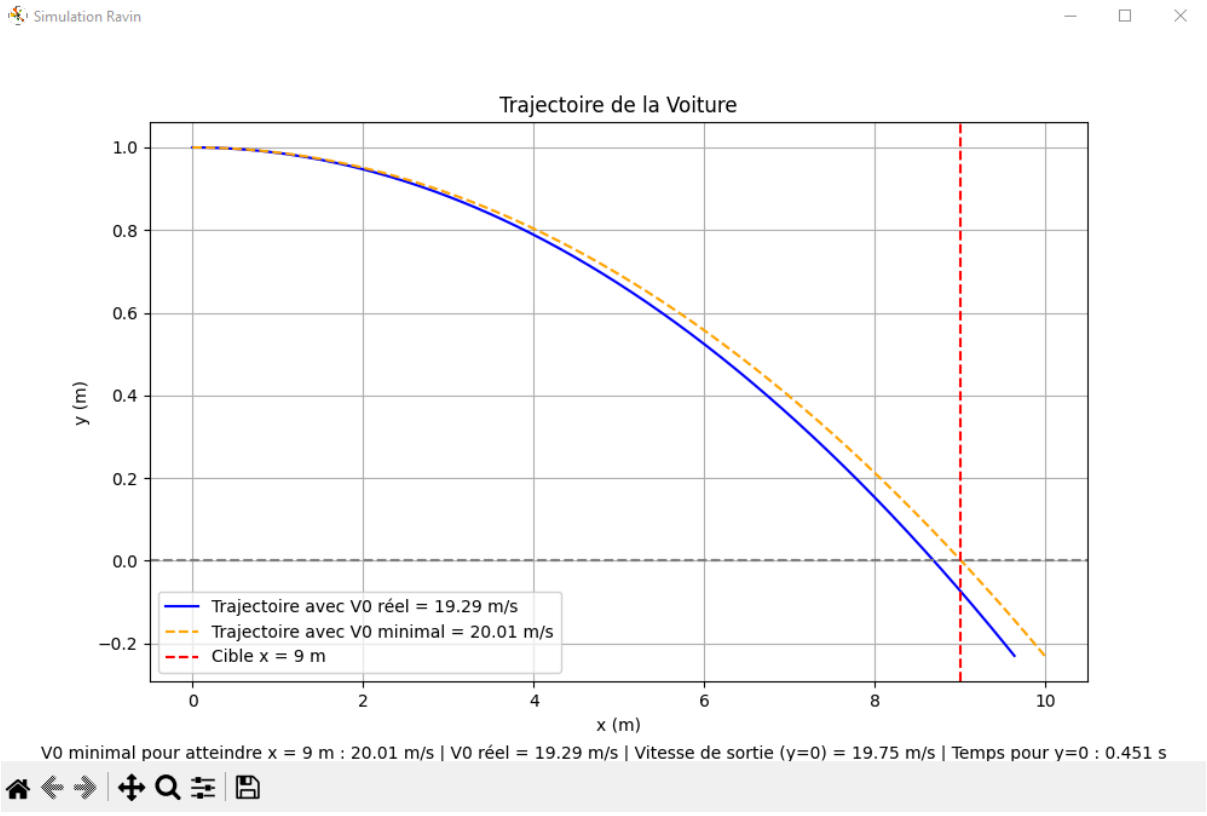


La valeur de la vitesse minimale à avoir pour franchir le ravin et la comparaison avec la vitesse de sortie du looping

Modèle 5 : Nissan Skyline GTR-R34, 1999



Modèle 6 : Mitsubishi Lancer Evolution VII



vitesse min pour passer le ravin

il faut déterminer la valeur de vitesse v0 qui résulte en x(t) = 9m(longueur du ravin) dans l'instant où y(t) = 0 (la voiture touche le sol)

Voiture	Vitesse min (Ravin)
Modèle 1 : Dodge Charger R/T, 1970	20.02m/s
Modèle 2 : Toyota Supra Mark IV, 1994	20m/s
Modèle 3 : Chevrolet Yenko Camaro 1969	20.02m/s
Modèle 4 : Mazda RX-7 FD	20.01m/s
Modèle 5 : Nissan Skyline GTR-R34, 1999	20.02m/s
Modèle 6 : Mitsubishi Lancer Evolution VII	20.01m/s

comparaison avec la vitesse de sortie du looping

Voiture	Vitesse de sortie (Looping)		Vitesse min (Ravin)
Modèle 1 : Dodge Charger R/T, 1970	19.428m/s	<	20.02m/s
Modèle 2 : Toyota Supra Mark IV, 1994	19.311m/s	<	20m/s
Modèle 3 : Chevrolet Yenko Camaro 1969	19.731m/s	<	20.02m/s
Modèle 4 : Mazda RX-7 FD	19.606m/s	<	20.01m/s
Modèle 5 : Nissan Skyline GTR-R34, 1999	20.476m/s	>	20.02m/s
Modèle 6 : Mitsubishi Lancer Evolution VII	19.294	<	20.01m/s

Une conclusion sur la possibilité de passer le ravin

La Nissan Skyline est la seule voiture capable de passer le ravin.

Voiture	possibilité de passer le ravin
Modèle 1 : Dodge Charger R/T, 1970	PAS POSSIBLE
Modèle 2 : Toyota Supra Mark IV, 1994	PAS POSSIBLE
Modèle 3 : Chevrolet Yenko Camaro 1969	PAS POSSIBLE
Modèle 4 : Mazda RX-7 FD	PAS POSSIBLE
Modèle 5 : Nissan Skyline GTR-R34, 1999	POSSIBLE
Modèle 6 : Mitsubishi Lancer Evolution VII	PAS POSSIBLE

code (Ravin)

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
import FinDePiste
from tkinter import messagebox, ttk
# Variable globale pour stocker vx lorsque y = 0
final_vx_at_y0 = 0.0

def Run Module(m, Largeur, Hauteur, Longueur, Cx, Cz, mu, Acc
Moyenne, V0,T,nos_active, nos_location, aileron_active):
    # Constantes
    rho = 1.225  # Densité de l'air (kg/m^3)

    A = Largeur * Hauteur
    g = 9.81  # Gravité (m/s^2)
    SurfacePortBonus = 0

    CzBonus = 1
    CxReduit = 1
    if aileron_active == True:

        SurfacePortBonus = 0.8
        CzBonus = 1.1
        CxReduit = 1 - 0.05

    S = Longueur * Largeur + SurfacePortBonus  # Aire de la section
transversale (m^2)
    # Équations du mouvement
    def equations(state, t):
        x, vx, y, vy = state
        v = np.sqrt(vx ** 2 + vy ** 2)  # Amplitude de la vitesse

        # Accélérations avec traînée et portance
        ax = -0.5 * rho * Cx*CxReduit * A * v * vx / (m)
        ay = -g + 0.5 * rho * Cz*CzBonus * S * v * vy / (m)
```

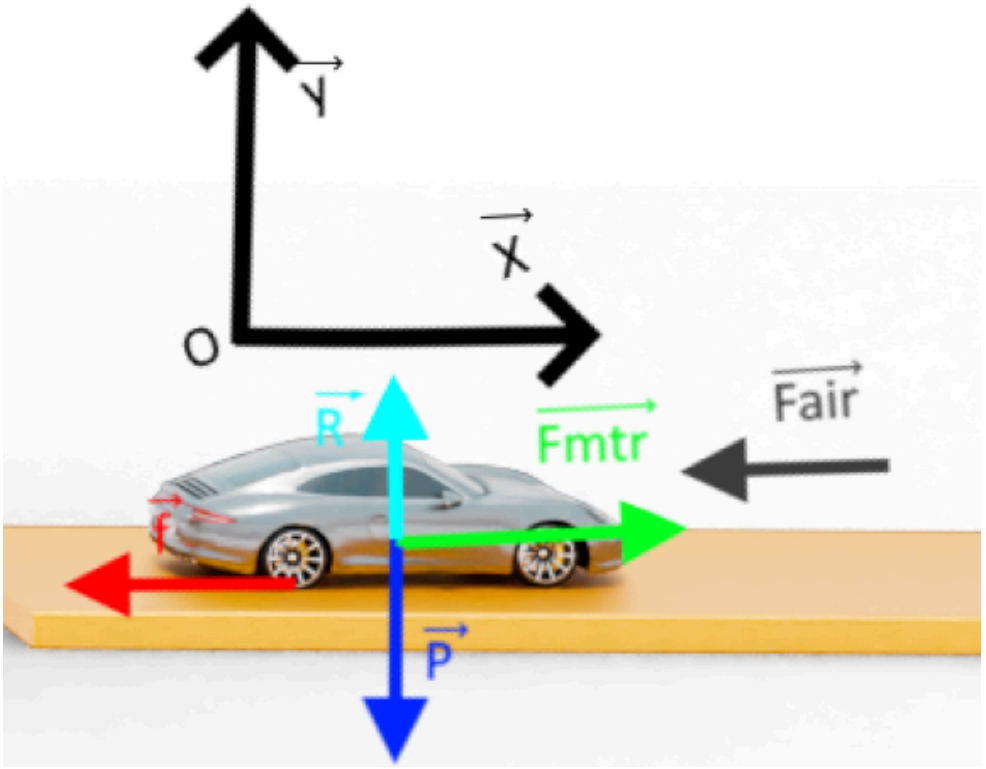
```
        return [vx, ax, vy, ay]

    # Fonction pour résoudre la trajectoire
    def solve_trajectory(v0_x):
        # Conditions initiales : x0, v0_x, y0, v0_y
        initial_conditions = [0, v0_x, 1, 0]

        # Intervalle de temps
        t = np.linspace(0, 0.5, 1000)  # Résolution jusqu'à 0,5
secondes

        # Résolution des EDO
        solution = odeint(equations, initial_conditions, t)

        x = solution[:, 0]  # Position horizontale
        y = solution[:, 2]  # Position verticale

        return x, y, t

    # Fonction pour trouver la vitesse initiale minimale v0_x pour
atteindre x=9 lorsque y=0
    def find_min_v0(target_x):
        def condition(v0_x):
            x, y, _ = solve_trajectory(v0_x)
            # Trouver où y croise 0 et retourner la valeur de x
correspondante
            for i in range(len(y) - 1):
                if y[i] > 0 and y[i + 1] <= 0:  # Détecter un
croisement
                    return x[i] - target_x
            return -target_x  # Si y n'atteint jamais 0, retourner
une erreur élevée

        # Méthode de bissection pour trouver la v0_x minimale
        v0_low = 1.0  # Limite inférieure
        v0_high = 50.0  # Limite supérieure
        tol = 1e-2  # Tolérance pour la précision

        while v0_high - v0_low > tol:
            v0_mid = (v0_low + v0_high) / 2
            if condition(v0_mid) >= 0:  # Dépasse ou atteint la cible
                v0_high = v0_mid
            else:
                v0_low = v0_mid

        return v0_high

    # Fonction pour trouver la vitesse à y = 0
```

```
    def get_exit_velocity(y, vx, vy):
        global final_vx_at_y0  # Utiliser la variable globale
        for i in range(len(y) - 1):
            if y[i] > 0 and y[i + 1] <= 0:  # Détecter le croisement
y = 0
                final_vx_at_y0 = vx[i]  # Stocker la composante
horizontale v_x
                v_exit = np.sqrt(vx[i]**2 + vy[i]**2)  # Vitesse
totale à y = 0
                return v_exit
        return 0.0  # Si aucun croisement n'est trouvé

    # Trouver la vitesse horizontale initiale minimale
    target_x = 9  # Position cible en x
    minimal_v0 = find_min_v0(target_x)

    # Résoudre la trajectoire en utilisant la vitesse initiale donnée
(V0 réel)
    x_given, y_given, t = solve_trajectory(V0)

    # Calculer les composantes de la vitesse pour la trajectoire
donnée
    vx_given = np.gradient(x_given, t)
    vy_given = np.gradient(y_given, t)

    # Calculer la vitesse de sortie à y = 0
    exit_velocity = get_exit_velocity(y_given, vx_given, vy_given)

    # Affichage de la composante v_x lorsque y=0
    print(f"Composante horizontale finale v_x lorsque y=0 :
{final_vx_at_y0:.2f} m/s")

    # Résoudre la trajectoire en utilisant la vitesse minimale v0
pour comparaison (optionnel)
    x_min, y_min, _ = solve_trajectory(minimal_v0)

    if V0 >= minimal_v0:


        # Trouver le temps où y atteint 0
        time_at_y0 = None
        for i in range(len(y_given) - 1):
            if y_given[i] > 0 and y_given[i + 1] <= 0:  # Croisement
de y = 0
                # Interpolation pour calculer le temps précis
                time_at_y0 = t[i] + (0 - y_given[i]) * (t[i + 1] -
t[i]) / (y_given[i + 1] - y_given[i])
                break  # On s'arrête au premier croisement
```

```
        # Tracer la trajectoire résultante pour la vitesse donnée V0
        FinDePiste.RunModule(m, Largeur, Hauteur, Longueur, Cx, Cz,
mu, AccMoyenne, exit_velocity, T + time_at_y0,nos_active,
nos_location, aileron_active)
        plt.figure(3, figsize=(10, 6))
        plt.plot(x_given, y_given, label=f'Trajectoire avec V0 réel =
{V0:.2f} m/s', color='blue')
        plt.plot(x_min, y_min, label=f'Trajectoire avec V0 minimal =
{minimal_v0:.2f} m/s', color='orange',
                 linestyle='--')

        plt.axhline(0, color='gray', linestyle='--')  # Ligne du sol
        plt.axvline(target_x, color='red', linestyle='--',
label='Cible x = 9 m')

        plt.title('Trajectoire de la Voiture')
        plt.xlabel('x (m)')
        plt.ylabel('y (m)')
        plt.legend()
        plt.grid()

        # Ajouter les informations calculées en bas du graphique
        plt.figtext(0.5, 0.01,
                    f"V0 minimal pour atteindre x = {target_x} m :
{minimal_v0:.2f} m/s | "
                    f"V0 réel = {V0:.2f} m/s | Vitesse de sortie
(y=0) = {exit_velocity:.2f} m/s | "
                    f"Temps pour y=0 : {time_at_y0:.3f} s",
                    fontsize=10, ha="center", va="center")

        plt.gcf().canvas.manager.set_window_title("Simulation Ravin")
        plt.show(block=False)
    else:
        messagebox.showerror("Erreur d'Entrée",f"La voiture n'a pas
pu franchir le ravin en raison d'une V0 insuffisante : {V0:.2f} m/s
< {minimal_v0:.2f} m/s")
```

V. FIN DE PISTE

Le rappel du schéma et des équations du mouvement (livrable 2)



forces:

$$\vec{f} \Rightarrow force\ de\ frottement\ avec\ le\ sol$$

$$\vec{R} \Rightarrow normale\ R$$

$$\vec{P} \Rightarrow poid$$

$$\vec{Fmtr} \Rightarrow force\ motrice\ du\ moteur\ de\ la\ voiture$$

$$\vec{Fair} \Rightarrow force\ de\ frottement\ avec\ l'air$$

sur l'axe (x)

$$\vec{ma}_x = \vec{f} + \vec{Fair} + \vec{Fmtr}$$

$$ma_x = Fmtr - f - Fair$$

$$ma_x = m.am - \mu.R - k.v^2$$

$$\frac{dvx}{dt} = \frac{m.am - \mu.R - k.v^2}{m}$$

l'équation différentielle est traité par le code python

Le temps total pour parcourir le circuit

cette valeur est déterminée en prenant l'instant où x(t) = 10m (longueur de la fin de piste) et puisque la **Nissan Skyline GTR-R34, 1999** est la seule voiture a pouvoir passer le ravin elle est la seul a pouvoir théoriquement terminer la course entièrement

Nissan Skyline GTR-R34, 1999: Temps de complétion du circuit: 6.65s
code (fin de piste)

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp
import tkinter as tk
from tkinter import messagebox, ttk

def RunModule(m, Largeur, Hauteur, Longueur, Cx, Cz, mu, am,
V0,T,nos_active, nos_location, aileron_active):
    # Paramètres physiques
    R = m * 9.81  # Force normale (Poids, N) avec g = 9.81 m/s²
    ro = 1.292  # Masse volumique de l'air (kg/m³)
    k = 0.5 * Cx * Hauteur * Largeur * ro  # Coefficient de traînée
aérodynamique (kg/m)
    boost = 1
    if nos_active == True and nos_location == "la fin de piste":
        boost = 1.3
    # Équation différentielle : dv/dt et dx/dt
    def equations(t, state):
        x, v = state
        dxdt = v
        dvdt = (m * am*boost - mu * R - k * v ** 2) / m
        return [dxdt, dvdt]

    # Fonction d'événement pour arrêter l'intégration lorsque x
atteint 10m
    def stop_at_10m(t, state):
        x, v = state
        return x - 10

    stop_at_10m.terminal = True
    stop_at_10m.direction = 1

    # Conditions initiales
    x0 = 0
    initial_state = [x0, V0]

    # Vecteur temps
    t_span = (0, 50)
    t_eval = np.linspace(0, 50, 1000)  # Plus de points pour
améliorer la précision

    # Résolution avec solve_ivp
```

```
    solution = solve_ivp(equations, t_span, initial_state,
t_eval=t_eval, events=stop_at_10m, rtol=1e-8, atol=1e-10)

    # Extraire les résultats
    x = solution.y[0]
    v = solution.y[1]
    t = solution.t

    # Interpolation pour trouver exactement x = 10
    if solution.status == 1:  # Arrêt par événement
        # Prendre les deux derniers points avant et après x = 10
        x_before, x_after = x[-2], x[-1]
        t_before, t_after = t[-2], t[-1]
        v_before, v_after = v[-2], v[-1]

        # Interpolation linéaire pour le temps et la vitesse à x = 10
        t_10 = t_before + (10 - x_before) * (t_after - t_before) /
(x_after - x_before)
        v_10 = v_before + (10 - x_before) * (v_after - v_before) /
(x_after - x_before)
    else:
        t_10, v_10 = None, None

    # Tracer les résultats
    plt.figure(4,figsize=(10, 6))
    plt.subplot(2, 1, 1)
    plt.plot(t, v, label="Vitesse v_x(t)", color='b')
    plt.title("Vitesse v_x(t) en fonction du temps")
    plt.xlabel("Temps (s)")
    plt.ylabel("Vitesse (m/s)")
    plt.grid(True)

    plt.subplot(2, 1, 2)
    plt.plot(t, x, label="Position x(t)", color='r')
    plt.title("Position x(t) en fonction du temps")
    plt.xlabel("Temps (s)")
    plt.ylabel("Position (m)")
    plt.grid(True)

    # Ajouter la vitesse et le temps exacts à x = 10 m
    if t_10 is not None:
        plt.figtext(0.5, 0.01,
                    f"Vitesse finale à 10m : {v_10:.2f} m/s | Temps
pour atteindre 10m : {t_10:.2f} s",
                    fontsize=10, ha="center", va="center")
    else:
        plt.figtext(0.5, 0.01, "L'événement x = 10m n'a pas été
détecté", fontsize=10, ha="center", va="center")
```

```
    plt.tight_layout()

    plt.gcf().canvas.manager.set_window_title("Simulation Fin de
piste")
    plt.show(block=False)

    success_window = tk.Toplevel(tk._default_root)
    success_window.title("Succès")
    success_window.geometry("300x100")  # Adjust size as needed
    tk.Label(success_window, text=f"Course terminée en : {T +
t_10:.2f} secondes").pack(pady=20)
    tk.Button(success_window, text="OK",
command=success_window.destroy).pack()
```

VI. AMÉLIORATIONS

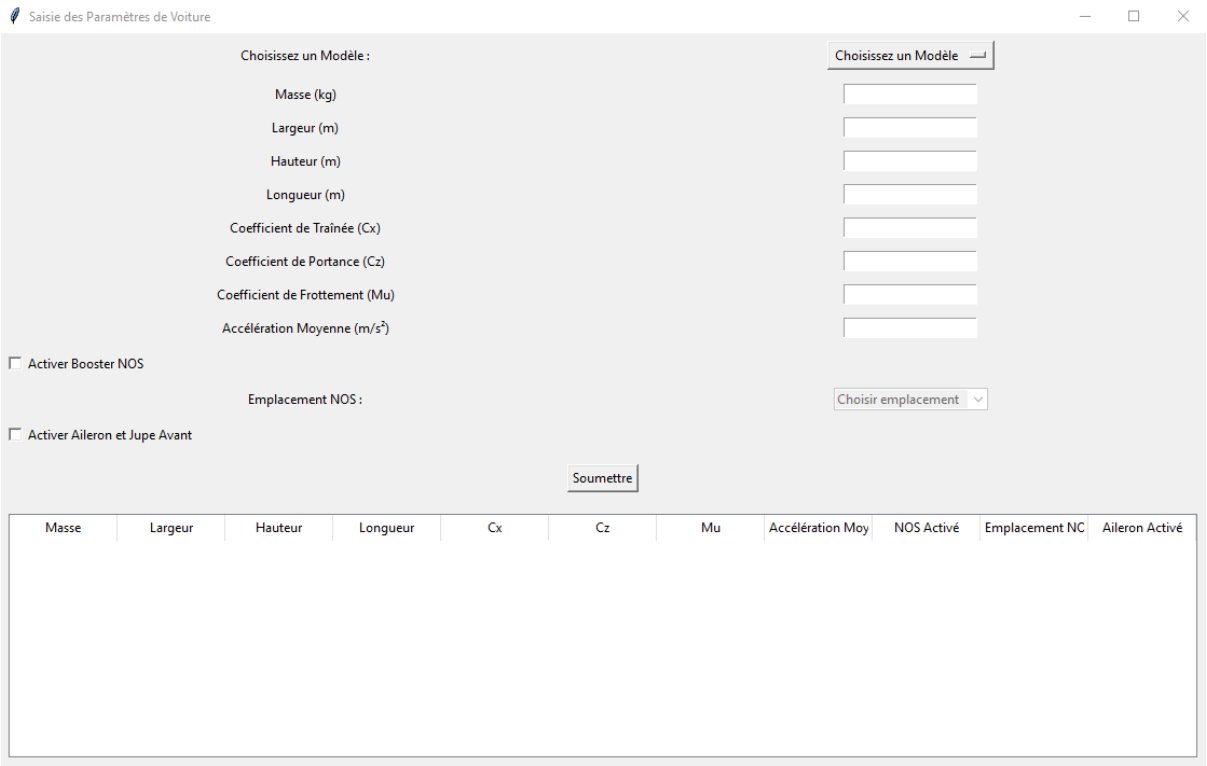
les testes sur notre programme python on démontré qu'il est possible de finir la course encore plus rapidement en utilisant un Booster NOS les meilleurs temps ont été obtenus lors de son utilisation dans la partie pente:

Voiture	temps pour finir la course
Modèle 1 : Dodge Charger R/T, 1970	6.37s
Modèle 2 : Toyota Supra Mark IV, 1994	Ravin-non-validé
Modèle 3 : Chevrolet Yenko Camaro 1969	6.24s
Modèle 4 : Mazda RX-7 FD	6.29s
Modèle 5 : Nissan Skyline GTR-R34, 1999	5.94s
Modèle 6 : Mitsubishi Lancer Evolution VII	Ravin-non-validé

(Un booster permet de gagner 30% de l'accélération moyenne de la voiture sur une portion de circuit uniquement (pente, looping ou fin de piste).
et la NISSAN GTR skyline est toujour la plus rapide donc c'est elle qu'on choisit

pour tester les voitures vous mêmes:
Programme complet de simulation: Download

additionnel:
code de l’interface du programme:




```python
import tkinter as tk
from tkinter import messagebox, ttk

import Pente

PRESETS = {
    "Modèle 1 : Dodge Charger R/T, 1970": {"mass": 1760, "width":
1.95, "height": 1.35, "length": 5.28, "Cx": 0.38,
                                          "Cz": 0.3, "mu": 0.1,
"a_avg": 5.1},
    "Modèle 2 : Toyota Supra Mark IV, 1994": {"mass": 1615, "width":
1.81, "height": 1.27, "length": 4.51, "Cx": 0.29,
                                            "Cz": 0.3, "mu": 0.1,
"a_avg": 5},
    "Modèle 3 : Chevrolet Yenko Camaro 1969": {"mass": 1498, "width":
1.88, "height": 1.30, "length": 4.72, "Cx": 0.35,
                                             "Cz": 0.3, "mu": 0.1,
"a_avg": 5.3},
    "Modèle 4 : Mazda RX-7 FD": {"mass": 1385, "width": 1.75,
"height": 1.23, "length": 4.3, "Cx": 0.28, "Cz": 0.3,
                                  "mu": 0.1, "a_avg": 5.2},
    "Modèle 5 : Nissan Skyline GTR-R34, 1999": {"mass": 1540,
"width": 1.79, "height": 1.36, "length": 4.6, "Cx": 0.34,
                                             "Cz": 0.3, "mu": 0.1,
"a_avg": 5.8},
    "Modèle 6 : Mitsubishi Lancer Evolution VII": {"mass": 1600,
"width": 1.81, "height": 1.48, "length": 4.51,
```

```
                                                    "Cx": 0.28, "Cz":
0.3, "mu": 0.1, "a_avg": 5}
}


def load_preset(*args):
    preset_name = preset_var.get()
    if preset_name in PRESETS:
        preset = PRESETS[preset_name]
        mass_entry.delete(0, tk.END)
        mass_entry.insert(0, preset["mass"])
        width_entry.delete(0, tk.END)
        width_entry.insert(0, preset["width"])
        height_entry.delete(0, tk.END)
        height_entry.insert(0, preset["height"])
        length_entry.delete(0, tk.END)
        length_entry.insert(0, preset["length"])
        cx_entry.delete(0, tk.END)
        cx_entry.insert(0, preset["Cx"])
        cz_entry.delete(0, tk.END)
        cz_entry.insert(0, preset["Cz"])
        mu_entry.delete(0, tk.END)
        mu_entry.insert(0, preset["mu"])
        a_avg_entry.delete(0, tk.END)
        a_avg_entry.insert(0, preset["a_avg"])


def submit_parameters():
    try:
        mass = float(mass_entry.get())
        width = float(width_entry.get())
        height = float(height_entry.get())
        length = float(length_entry.get())
        cx = float(cx_entry.get())
        cz = float(cz_entry.get())
        mu = float(mu_entry.get())
        a_avg = float(a_avg_entry.get())

        # Retrieve NOS and Aileron options
        nos_active = nos_var.get()
        nos_location = nos_location_var.get() if nos_active else "Non
activé"
        aileron_active = aileron_var.get()
        massAileron = 0
        if aileron_active == True:
            massAileron = 45
        # Call RunModule with all required arguments
```

```
        Pente.RunModule(mass+ massAileron, width, height, length, cx,
cz, mu, a_avg, nos_active, nos_location, aileron_active)

        # Update the table
        tree.insert("", "end", values=(mass, width, height, length,
cx, cz, mu, a_avg, nos_active, nos_location, aileron_active))

    except ValueError:
        messagebox.showerror("Erreur de saisie", "Veuillez entrer des
valeurs numériques valides pour tous les champs.")


root = tk.Tk()
root.title("Saisie des Paramètres de Voiture")

# Variables
preset_var = tk.StringVar(value="Choisissez un Modèle")
nos_var = tk.BooleanVar(value=False)
nos_location_var = tk.StringVar(value="Choisir emplacement")
aileron_var = tk.BooleanVar(value=False)

# Widgets de base
preset_label = tk.Label(root, text="Choisissez un Modèle :")
preset_label.grid(row=0, column=0, padx=5, pady=5)
preset_dropdown = tk.OptionMenu(root, preset_var, *PRESETS.keys(),
command=load_preset)
preset_dropdown.grid(row=0, column=1, padx=5, pady=5)

fields = [("Masse (kg)", 1), ("Largeur (m)", 2), ("Hauteur (m)", 3),
("Longueur (m)", 4),
          ("Coefficient de Traînée (Cx)", 5), ("Coefficient de
Portance (Cz)", 6),
          ("Coefficient de Frottement (Mu)", 7), ("Accélération
Moyenne (m/s²)", 8)]
entries = []

for label, row in fields:
    tk.Label(root, text=label).grid(row=row, column=0, padx=5,
pady=5)
    entry = tk.Entry(root)
    entry.grid(row=row, column=1, padx=5, pady=5)
    entries.append(entry)

mass_entry, width_entry, height_entry, length_entry, cx_entry,
cz_entry, mu_entry, a_avg_entry = entries

# Options personnalisables
```

```
nos_check = tk.Checkbutton(root, text="Activer Booster NOS",
variable=nos_var)
nos_check.grid(row=9, column=0, sticky="w", padx=5, pady=5)

nos_location_label = tk.Label(root, text="Emplacement NOS :")
nos_location_label.grid(row=10, column=0, padx=5, pady=5)
nos_location_menu = ttk.Combobox(root,
textvariable=nos_location_var, state="disabled",
                                 values=["la pente", "le looping",
"le ravin", "la fin de piste"])
nos_location_menu.grid(row=10, column=1, padx=5, pady=5)


def toggle_nos_location(*args):
   state = "readonly" if nos_var.get() else "disabled"
   nos_location_menu.config(state=state)


nos_var.trace_add("write", toggle_nos_location)

aileron_check = tk.Checkbutton(root, text="Activer Aileron et Jupe
Avant", variable=aileron_var)
aileron_check.grid(row=11, column=0, sticky="w", padx=5, pady=5)

# Bouton soumettre
submit_button = tk.Button(root, text="Soumettre",
command=submit_parameters)
submit_button.grid(row=12, column=0, columnspan=2, pady=10)

# Tableau
table_frame = tk.Frame(root)
table_frame.grid(row=13, column=0, columnspan=2, padx=10, pady=10)

columns = (
"Masse", "Largeur", "Hauteur", "Longueur", "Cx", "Cz", "Mu",
"Accélération Moyenne", "NOS Activé", "Emplacement NOS",
"Aileron Activé")
tree = ttk.Treeview(table_frame, columns=columns, show="headings")

for col in columns:
   tree.heading(col, text=col)
   tree.column(col, width=100, anchor="center")

tree.pack()

root.mainloop()
import tkinter as tk
from tkinter import messagebox, ttk
```

```
import Pente

PRESETS = {
   "Modèle 1 : Dodge Charger R/T, 1970": {"mass": 1760, "width":
1.95, "height": 1.35, "length": 5.28, "Cx": 0.38,
                                          "Cz": 0.3, "mu": 0.1,
"a_avg": 5.1},
   "Modèle 2 : Toyota Supra Mark IV, 1994": {"mass": 1615, "width":
1.81, "height": 1.27, "length": 4.51, "Cx": 0.29,
                                            "Cz": 0.3, "mu": 0.1,
"a_avg": 5},
   "Modèle 3 : Chevrolet Yenko Camaro 1969": {"mass": 1498, "width":
1.88, "height": 1.30, "length": 4.72, "Cx": 0.35,
                                             "Cz": 0.3, "mu": 0.1,
"a_avg": 5.3},
   "Modèle 4 : Mazda RX-7 FD": {"mass": 1385, "width": 1.75,
"height": 1.23, "length": 4.3, "Cx": 0.28, "Cz": 0.3,
                                  "mu": 0.1, "a_avg": 5.2},
   "Modèle 5 : Nissan Skyline GTR-R34, 1999": {"mass": 1540,
"width": 1.79, "height": 1.36, "length": 4.6, "Cx": 0.34,
                                             "Cz": 0.3, "mu": 0.1,
"a_avg": 5.8},
   "Modèle 6 : Mitsubishi Lancer Evolution VII": {"mass": 1600,
"width": 1.81, "height": 1.48, "length": 4.51,
                                                "Cx": 0.28, "Cz":
0.3, "mu": 0.1, "a_avg": 5}
}


def load_preset(*args):
   preset_name = preset_var.get()
   if preset_name in PRESETS:
       preset = PRESETS[preset_name]
       mass_entry.delete(0, tk.END)
       mass_entry.insert(0, preset["mass"])
       width_entry.delete(0, tk.END)
       width_entry.insert(0, preset["width"])
       height_entry.delete(0, tk.END)
       height_entry.insert(0, preset["height"])
       length_entry.delete(0, tk.END)
       length_entry.insert(0, preset["length"])
       cx_entry.delete(0, tk.END)
       cx_entry.insert(0, preset["Cx"])
       cz_entry.delete(0, tk.END)
       cz_entry.insert(0, preset["Cz"])
       mu_entry.delete(0, tk.END)
       mu_entry.insert(0, preset["mu"])
```

```
        a_avg_entry.delete(0, tk.END)
        a_avg_entry.insert(0, preset["a_avg"])


def submit_parameters():
    try:
        mass = float(mass_entry.get())
        width = float(width_entry.get())
        height = float(height_entry.get())
        length = float(length_entry.get())
        cx = float(cx_entry.get())
        cz = float(cz_entry.get())
        mu = float(mu_entry.get())
        a_avg = float(a_avg_entry.get())

        # Retrieve NOS and Aileron options
        nos_active = nos_var.get()
        nos_location = nos_location_var.get() if nos_active else "Non
activé"
        aileron_active = aileron_var.get()
        massAileron = 0
        if aileron_active == True:
            massAileron = 45
        # Call RunModule with all required arguments
        Pente.RunModule(mass+ massAileron, width, height, length, cx,
cz, mu, a_avg, nos_active, nos_location, aileron_active)

        # Update the table
        tree.insert("", "end", values=(mass, width, height, length,
cx, cz, mu, a_avg, nos_active, nos_location, aileron_active))

    except ValueError:
        messagebox.showerror("Erreur de saisie", "Veuillez entrer des
valeurs numériques valides pour tous les champs.")


root = tk.Tk()
root.title("Saisie des Paramètres de Voiture")

# Variables
preset_var = tk.StringVar(value="Choisissez un Modèle")
nos_var = tk.BooleanVar(value=False)
nos_location_var = tk.StringVar(value="Choisir emplacement")
aileron_var = tk.BooleanVar(value=False)

# Widgets de base
preset_label = tk.Label(root, text="Choisissez un Modèle :")
preset_label.grid(row=0, column=0, padx=5, pady=5)
```

```
preset_dropdown = tk.OptionMenu(root, preset_var, *PRESETS.keys(),
command=load_preset)
preset_dropdown.grid(row=0, column=1, padx=5, pady=5)

fields = [("Masse (kg)", 1), ("Largeur (m)", 2), ("Hauteur (m)", 3),
("Longueur (m)", 4),
          ("Coefficient de Traînée (Cx)", 5), ("Coefficient de
Portance (Cz)", 6),
          ("Coefficient de Frottement (Mu)", 7), ("Accélération
Moyenne (m/s²)", 8)]
entries = []

for label, row in fields:
    tk.Label(root, text=label).grid(row=row, column=0, padx=5,
pady=5)
    entry = tk.Entry(root)
    entry.grid(row=row, column=1, padx=5, pady=5)
    entries.append(entry)

mass_entry, width_entry, height_entry, length_entry, cx_entry,
cz_entry, mu_entry, a_avg_entry = entries

# Options personnalisables
nos_check = tk.Checkbutton(root, text="Activer Booster NOS",
variable=nos_var)
nos_check.grid(row=9, column=0, sticky="w", padx=5, pady=5)

nos_location_label = tk.Label(root, text="Emplacement NOS :")
nos_location_label.grid(row=10, column=0, padx=5, pady=5)
nos_location_menu = ttk.Combobox(root,
textvariable=nos_location_var, state="disabled",
                                 values=["la pente", "le looping",
"le ravin", "la fin de piste"])
nos_location_menu.grid(row=10, column=1, padx=5, pady=5)


def toggle_nos_location(*args):
    state = "readonly" if nos_var.get() else "disabled"
    nos_location_menu.config(state=state)


nos_var.trace_add("write", toggle_nos_location)

aileron_check = tk.Checkbutton(root, text="Activer Aileron et Jupe
Avant", variable=aileron_var)
aileron_check.grid(row=11, column=0, sticky="w", padx=5, pady=5)

# Bouton soumettre
```

```
submit_button = tk.Button(root, text="Soumettre",
command=submit_parameters)
submit_button.grid(row=12, column=0, columnspan=2, pady=10)

# Tableau
table_frame = tk.Frame(root)
table_frame.grid(row=13, column=0, columnspan=2, padx=10, pady=10)

columns = (
"Masse", "Largeur", "Hauteur", "Longueur", "Cx", "Cz", "Mu",
"Accélération Moyenne", "NOS Activé", "Emplacement NOS",
"Aileron Activé")
tree = ttk.Treeview(table_frame, columns=columns, show="headings")

for col in columns:
    tree.heading(col, text=col)
    tree.column(col, width=100, anchor="center")

tree.pack()

root.mainloop()
```

VII. CONCLUSION

nous savons maintenant quel voiture choisir pour la course de manière à assurer aucun dégât et la complétion en un temp record

FIN DU DOCUMENT