Modélisation mathématique du traffic routier: quel est l'impact de la réduction de vitesse sur la fréquence des bouchons

Introduction

Modèles statiques

Utilisés pour la prévision de la demande de transport

Modèles dynamiques

Modèles macroscopiques

- Issus d'une analogie hydrodynamique de l'écoulement des véhicules
- De l'ordre de la dizaine ou centaine de mètres en espace et de la minute en temps
- Modèles du 1er ordre (LWR), du 2nd ordre, cellulaire, cinétique, …

Modèles microscopiques

- Décrivent les comportements individuels de chaque véhicule (à travers EDO couplées)
- Pour un trafic suffisamment dense: vitesse (ou accélération) du véhicule (i) conditionnée par celles du véhicule $(\mathrm{i}-1)$ (modèles de poursuite)
- Modèles à distance de sécurité, de stimulus-réponse, à vitesse optimale …

Le modèle microscopique de second ordre de Chandler et al. (1958)

$$M\frac{\mathrm{d}}{\mathrm{d}t}u_i(t) = \lambda[u_{i-1}(t-\tau) - u_i(t-\tau)] \quad \forall t > \tau, i = 2, \dots, N$$

En posant $\alpha = \frac{\lambda}{M}$ et $\dot{x}_i(t) = u_i(t)$, on obtient:

$$\ddot{x}_i(t) = \alpha\,[\dot{x}_{i-1}(t-\tau) - \dot{x}_i(t-\tau)]$$

- <u>Résolution numérique par la méthode d'Euler</u>

On introduit un pas numérique fixe Δt tel que : $0 < \Delta t < \tau$ et $\exists k \epsilon \mathbb{N}$: $\tau = k\Delta t$.

On définit par U_i^n une solution numérique approchée de la solution de notre équation de modèle : $U_i^n \approx u_i(n\Delta t)$

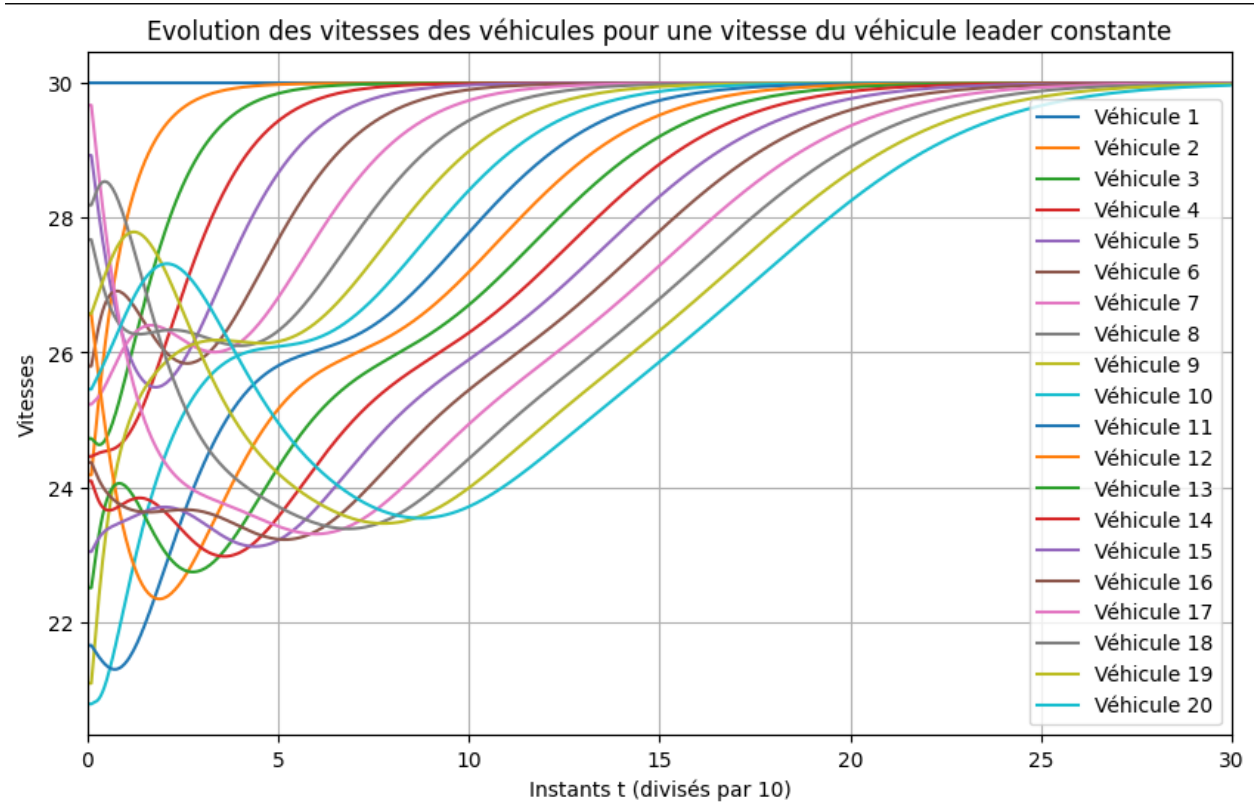
On peut alors approximer $\frac{\mathrm{d}}{\mathrm{d}t}u_i$ par par $\frac{U_i^{n+1} - U_i^n}{\Delta t}$ et on obtient le schéma numérique sur les vitesses :

$$U_i^{n+1} = U_i^n + \alpha\Delta t\left(U_{\mathrm{i}-1}^{n-k} - U_i^{n-k}\right)$$

Mise en place d’un programme pour une vitesse de véhicule leader constante

```python
# Création de la liste des vitesses U_1 du véhicule 1

U_1 = [30 for i in range(len(t))]
dic_U = {1: U_1}


# Création des listes des vitesses U_i des véhicules suivants

U_prec = U_1
for i in range(2, N_voitures + 1):

    U_i = [np.random.uniform(U_init_min, U_init_max)]
    U_i = U_i * 3
    for n in range(3,len(t)):
        U_i.append(U_i[n-1] + alpha * pas * (U_prec[n-1-k] - U_i[n-1-k]))

    U_prec = U_i
    dic_U[i] = U_i

df_U = pd.DataFrame(data = dic_U)
df_U.columns = ["U_"+str(j) for j in range(1, N_voitures + 1)]
```

```python
# Représentation des vitesses des véhicules au cours du temps pour une vitesse du véhicule leader constante

plt.figure(figsize=(10, 6))
for i in range(1, N_voitures + 1):
    plt.plot(t, dic_U[i], label=f"Véhicule {i}")
    plt.xlim(0,30)

plt.title("Evolution des vitesses des véhicules pour une vitesse du véhicule leader constante")
plt.xlabel("Instants t (divisés par 10)")
plt.ylabel("Vitesses")
plt.legend()
plt.grid()
plt.show()
```

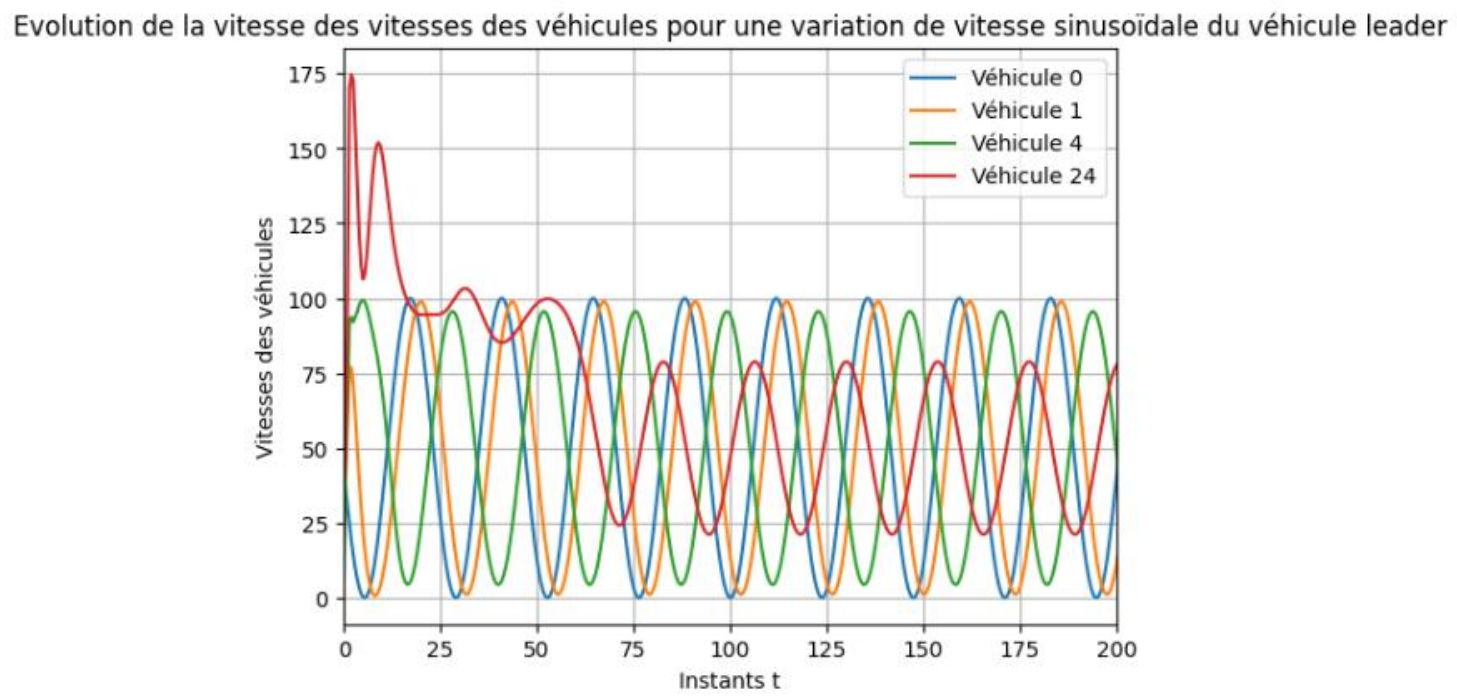


Mise en place d’un programme pour une vitesse de véhicule leader variant sinusoïdalement

```python
import numpy as np
import random
import pandas as pd
import matplotlib.pyplot as plt


N_voitures = 25
alpha = 0.37
pas = 0.5
tau = 1.5
k = int(tau/pas)

t = np.arange(0,300,pas) # Instants t
```

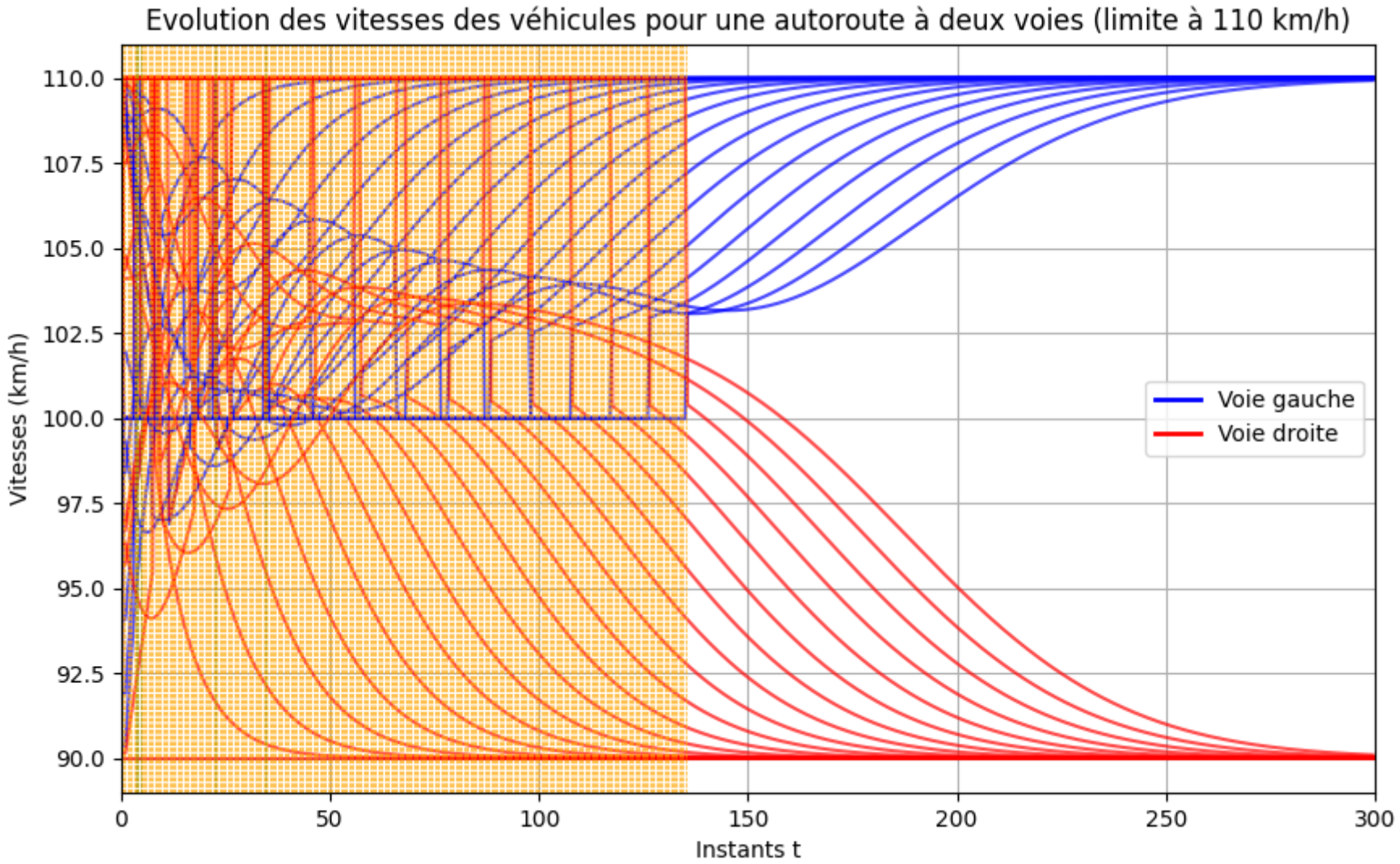
```python
U = np.zeros((N_voitures,len(t)))

for i in range(N_voitures):
        for n in range(4):
            if i ==0:
                U[i][n] = 50*np.sin((N_voitures)*(n+1))+50
            else:
                U[i][n] = random.uniform(N_voitures,N_voitures+i)*random.uniform(n,n+1)

for n in range(4,len(t)):
    U[0][n] = 50*np.sin((N_voitures)*(n+1))+50


for n in range(4,len(t)):
    for i in range(1,N_voitures):

        U[i][n] = U[i][n-1] + alpha*pas*(U[i-1][n-k]-U[i][n-k])


df_U = pd.DataFrame(data = U.T)
df_U.columns = ["U_"+str(i) for i in range(N_voitures)]
#print(df_U)
```

```python
plt.plot(t,df_U["U_0"], label="Véhicule 0")
plt.plot(t,df_U["U_1"], label="Véhicule 1")
plt.plot(t,df_U["U_4"], label="Véhicule 4")
plt.plot(t,df_U["U_24"], label="Véhicule 24")

plt.xlim(0,200)

plt.xlabel("Instants t")
plt.ylabel("Vitesses des véhicules")
plt.title("Evolution de la vitesse des vitesses des véhicules pour une variation de vitesse sinusoïdale du véhicule leader")
plt.grid()

plt.legend()
plt.show()
```



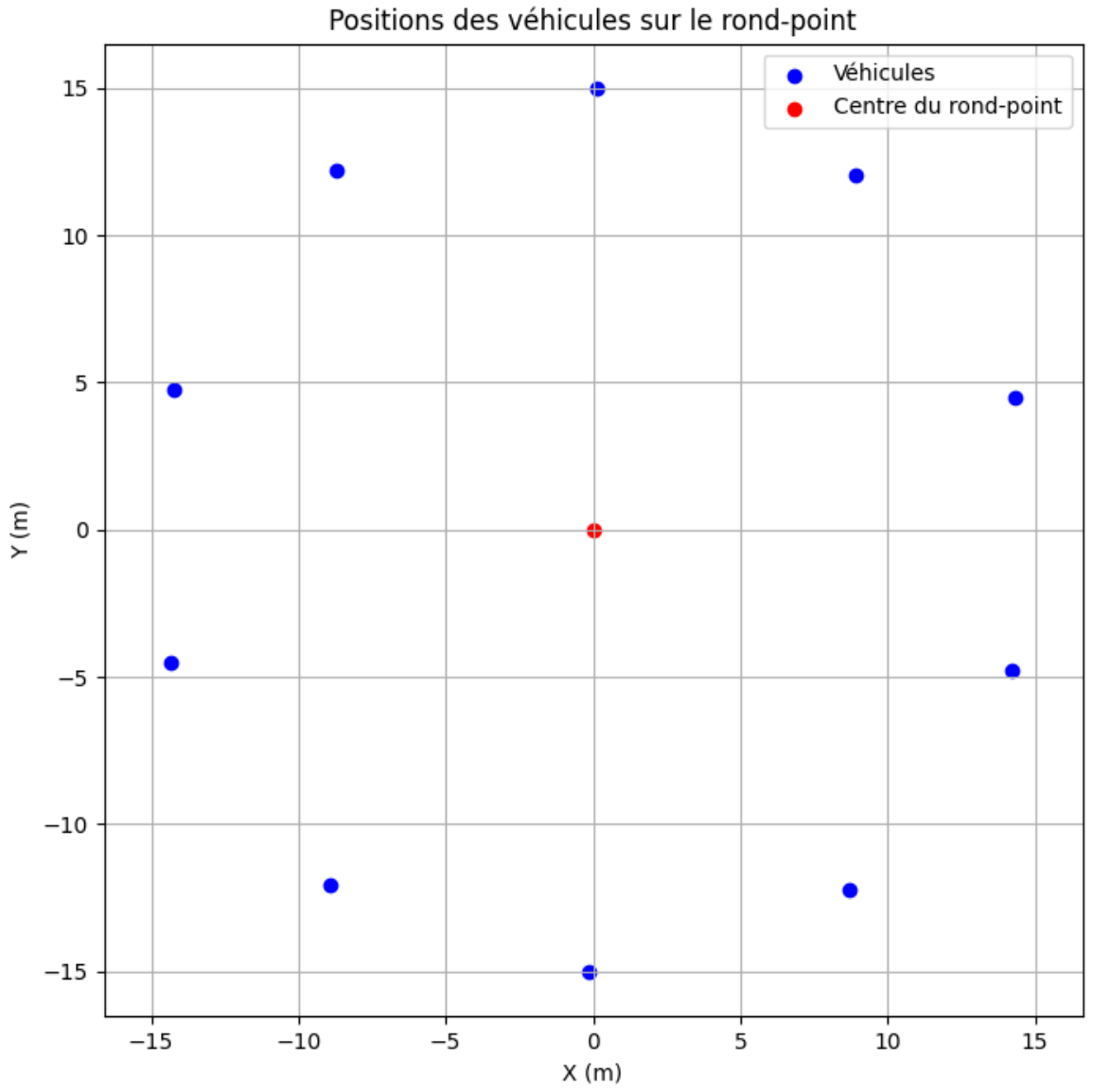
Partie du code pour une autoroute

```
# Création des vitesses pour la voie de gauche
U_1_left = [110 for _ in range(len(t))]
dic_U_left[1] = U_1_left

U_prec_left = U_1_left
for i in range(2, N_voitures + 1):
    U_i_left = [np.random.uniform(U_init_min, U_init_max)]
    U_i_left = U_i_left * 3
    for n in range(3, len(t)):
        U_i_left.append(U_i_left[n-1] + alpha * pas * (U_prec_left[n-1-k] - U_i_left[n-1-k]))
    U_prec_left = U_i_left
    dic_U_left[i] = U_i_left

depassements_left = set()
depassements_right = set()

for j in range(1, len(t)):
    for i in range(2, N_voitures):
        if dic_U_right[i][j] > dic_U_right[i-1][j] and dic_U_left[i][j] >= dic_U_left[i-1][j] and dic_U_left[i][j] <= dic_U_right[i-1][j]:
            dic_U_left[i][j] = vmax_highway
            dic_U_right[i][j] = vmax_highway - 10
            depassements_left.add(j)
        if dic_U_left[i][j] > dic_U_left[i-1][j]:
            if dic_U_left[i][j] > dic_U_right[i-1][j]:
                dic_U_right[i-1][j] = vmax_highway
                dic_U_left[i-1][j] = vmax_highway - 10
                depassements_right.add(j)
```


Evolution des vitesses des véhicules pour une autoroute à deux voies (limite à 110 km/h)
Vitesses (km/h)
110.0
107.5
105.0
102.5
100.0
97.5
95.0
92.5
90.0
0
50
100
150
200
250
300
Instants t
Voie gauche
Voie droite

Création d'un rond-point

```python
# Paramètres
N = 10  # Nombre de véhicules
radius = 15  # Rayon du rond-point en mètres
speed_kmh = 30  # Vitesse des véhicules en km/h
dt = 0.1  # Pas de temps en secondes
t_max = 120  # Durée de la simulation en secondes

# Calcul de la circonférence du rond-point
circumference = 2 * np.pi * radius

# Calcul de l'espacement équidistant des véhicules
spacing = circumference / N

# Initialisation des positions des véhicules équidistantes sur le rond-point
positions = np.arange(0, circumference, spacing)

# Liste pour stocker les positions des véhicules à chaque pas de temps
positions_history = [positions.copy()]



# Conversion des positions en angles
angles = positions / radius


# Visualisation des positions des véhicules sur le rond-point
plt.figure(figsize=(8, 8))
plt.title("Positions des véhicules sur le rond-point")
plt.scatter(radius * np.sin(angles), radius * np.cos(angles), label="Véhicules", color='blue')
plt.scatter(0, 0, label="Centre du rond-point", color='red')
plt.xlabel("X (m)")
plt.ylabel("Y (m)")
plt.axis('equal')
plt.legend()
plt.grid(True)
plt.show()
```



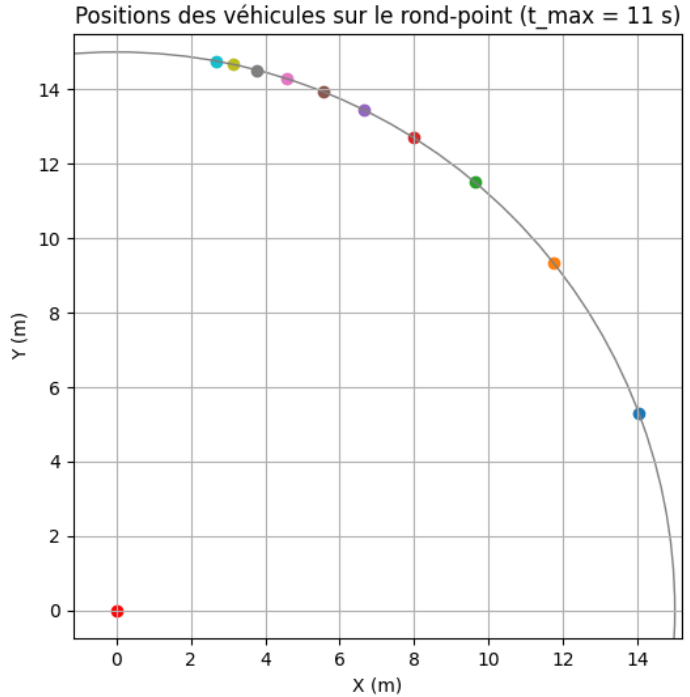
Mise en mouvement des véhicules

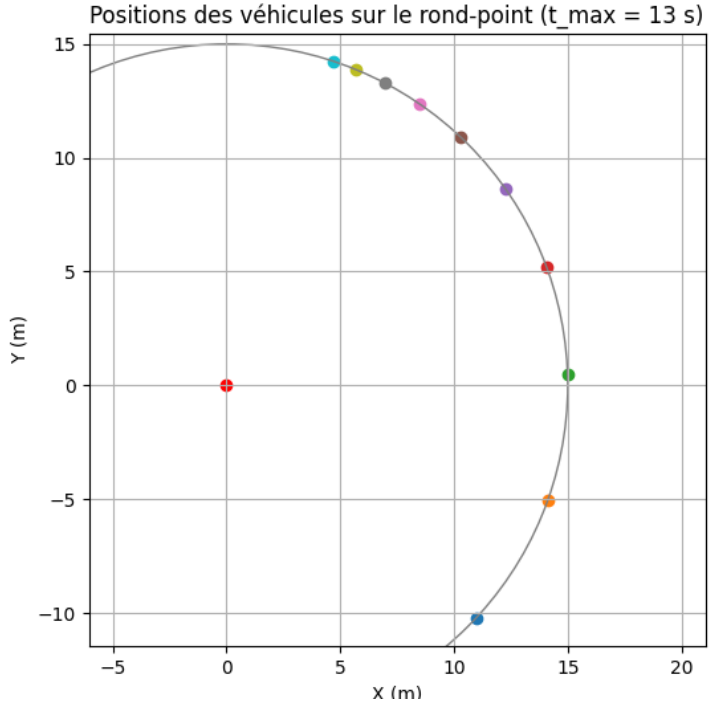
```python
# Paramètres constants
N = 10  # Nombre de véhicules
radius = 15  # Rayon du rond-point en mètres
speed_kmh = 30  # Vitesse des véhicules en km/h
initial_slow_speed_kmh = 10  # Vitesse initiale réduite de la première voiture en km/h
slow_duration = 5  # Durée pendant laquelle la première voiture ralentit en secondes
acceleration_duration = 5  # Durée pendant laquelle la première voiture accélère en secondes
alpha = 1.0  # Coefficient de décélération pour le ralentissement et l'accélération
dt = 0.25  # Pas de temps en secondes
```

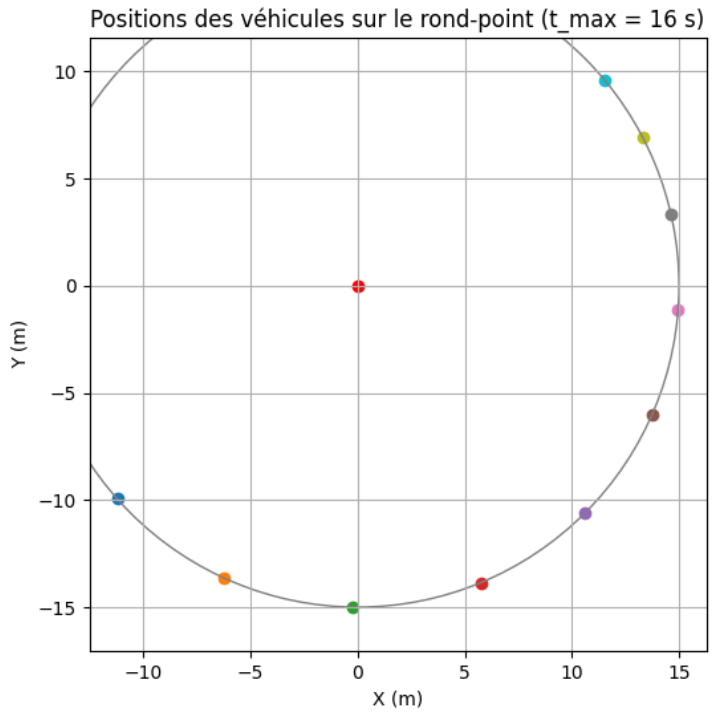
```python
# Fonction pour simuler et tracer les positions des véhicules
def simulate_and_plot(t_max):
    # Calcul de t_max basé sur la durée de ralentissement et d'accélération
    t_max = slow_duration + acceleration_duration + t_max

    # Calcul de la circonférence du rond-point
    circumference = 2 * np.pi * radius

    # Calcul des espacements équidistants des véhicules en fonction de leur vitesse
    spacing = (speed_kmh / 3.6) * dt  # Espacement équidistant basé sur la vitesse constante

    # Initialisation des positions des véhicules avec des espacements basés sur la vitesse
    positions = np.arange(0, N * spacing, spacing)

    # Réduction de la vitesse initiale de la première voiture
    positions[0] = 0  # Position initiale de la première voiture à l'origine

    # Liste pour stocker les positions des véhicules à chaque pas de temps
    positions_history = [positions.copy()]
```

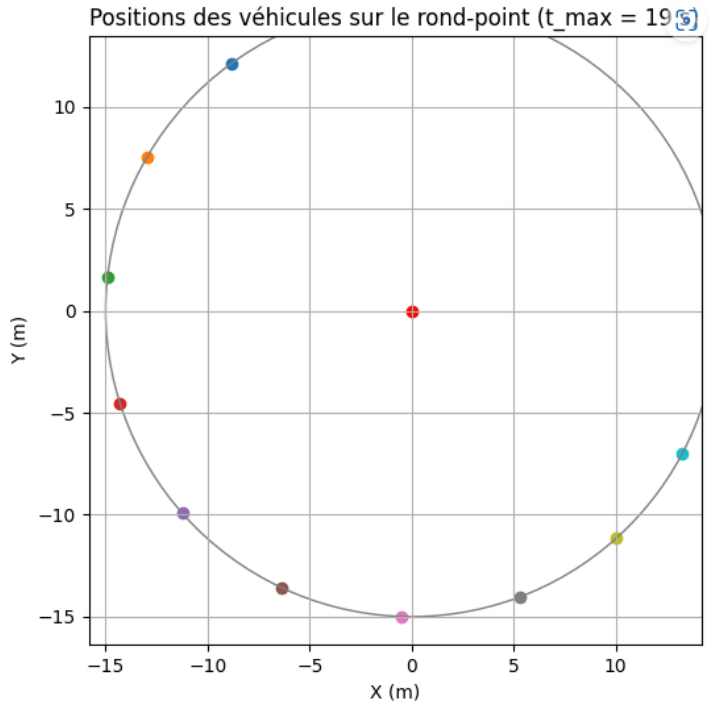
```python
# Boucle temporelle
for t in np.arange(dt, t_max, dt):
    # Copie temporaire des positions pour calculer les mises à jour
    positions_temp = positions.copy()

    # Ralentissement de la première voiture à 10 km/h pendant quelques secondes
    if t < slow_duration:
        # Application de l'équation de décélération pour le ralentissement
        positions_temp[0] -= alpha * dt * (positions_temp[0] - positions_temp[1])
    elif t < slow_duration + acceleration_duration:
        # Réaccélération progressive de la première voiture jusqu'à sa vitesse normale
        positions_temp[0] += alpha * dt * (initial_slow_speed_kmh / 3.6 - (positions_temp[0] - positions_temp[1]))
    else:
        # Vitesse normale de la première voiture
        positions_temp[0] += (speed_kmh / 3.6) * dt

    # Adaptation de la vitesse des autres véhicules en fonction de la vitesse du véhicule précédent
    for i in range(1, N):
        positions_temp[i] -= alpha * dt * (positions_temp[i] - positions_temp[i-1])

    # Mise à jour des positions
    positions = positions_temp

    # Enregistrement des positions dans l'historique
    positions_history.append(positions.copy())
```

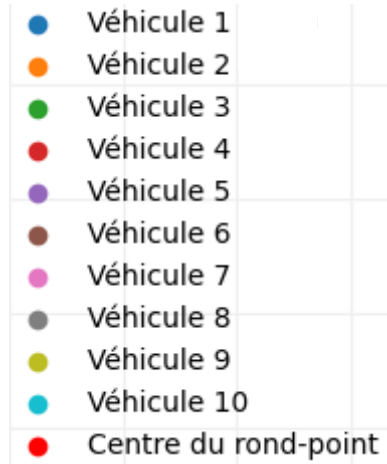
```python
# Simulation et tracé pour différentes valeurs de t_max
for t_max_value in range(1, 10):  # Valeurs de t_max de 1 à 5 secondes
    simulate_and_plot(t_max_value)
```

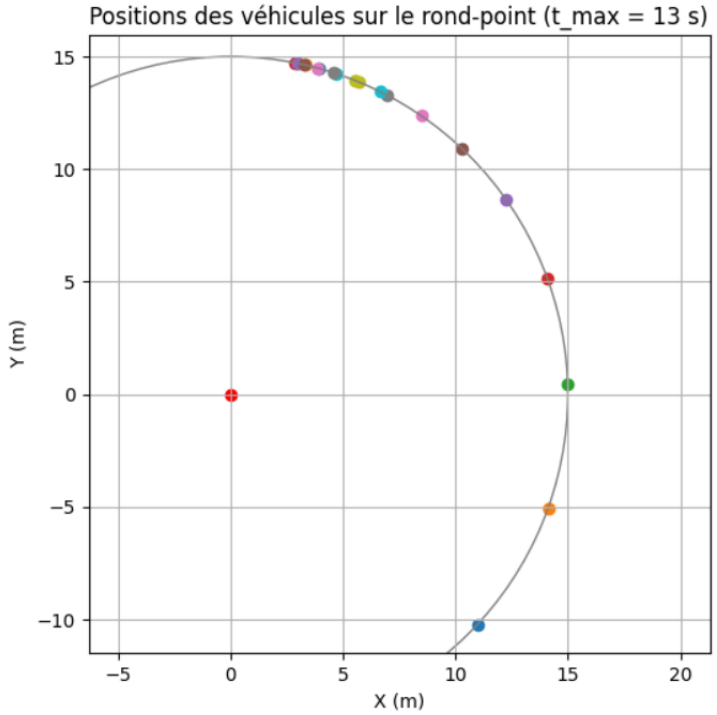

Positions des véhicules sur le rond-point (t_max = 11 s)
X (m)
Y (m)

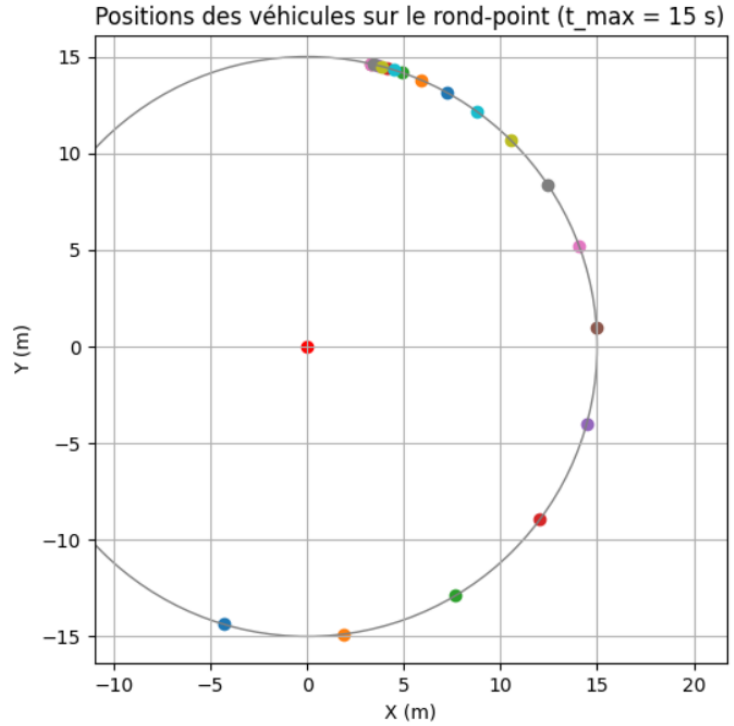

Positions des véhicules sur le rond-point (t_max = 13 s)
X (m)
Y (m)

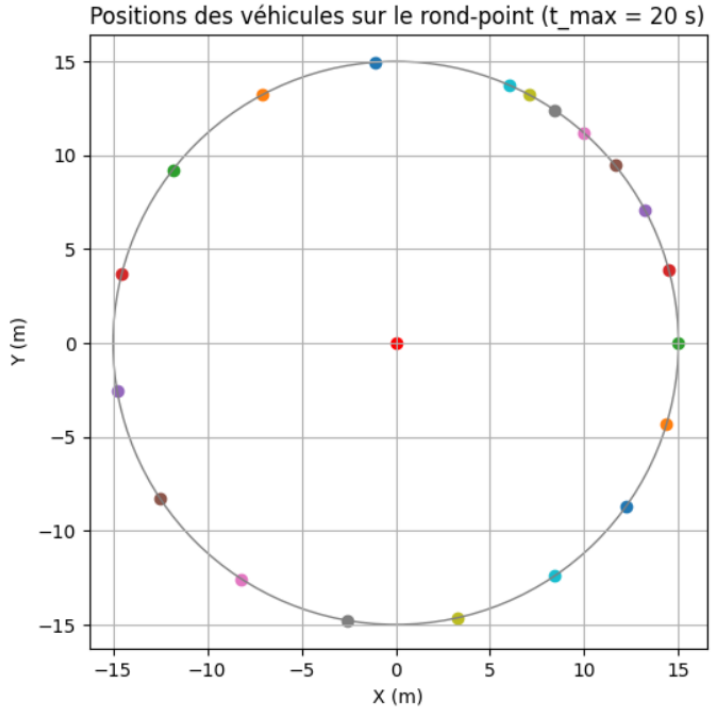

Positions des véhicules sur le rond-point (t_max = 16 s)
X (m)
Y (m)

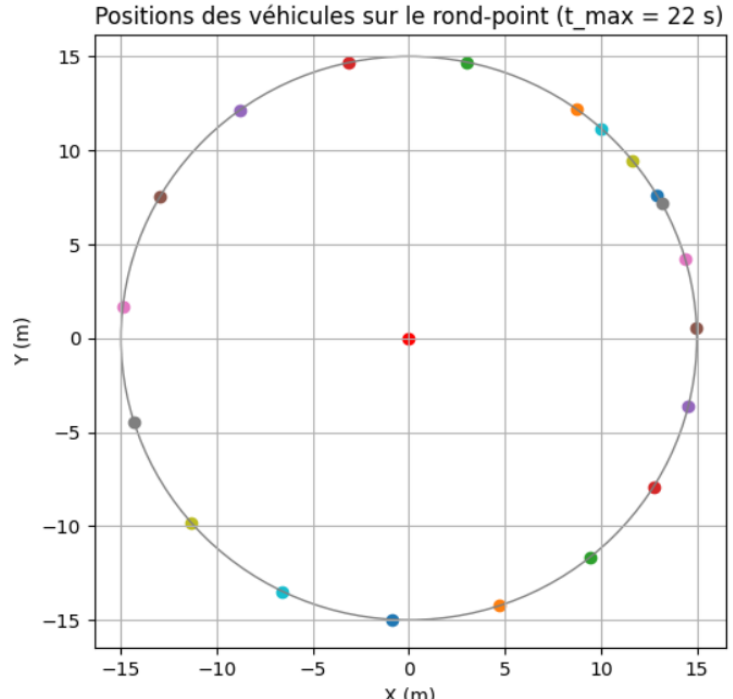

Positions des véhicules sur le rond-point (t_max = 19 s)
X (m)
Y (m)

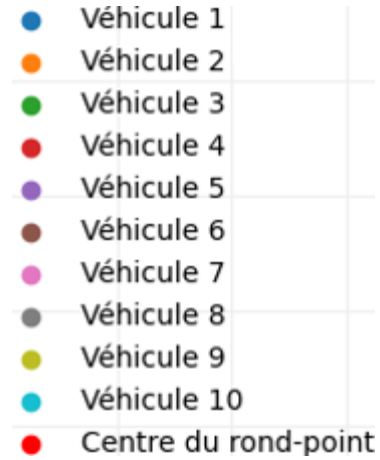

Véhicule 1
Véhicule 2
Véhicule 3
Véhicule 4
Véhicule 5
Véhicule 6
Véhicule 7
Véhicule 8
Véhicule 9
Véhicule 10
Centre du rond-point


Positions des véhicules sur le rond-point (t_max = 13 s)
X (m)
Y (m)


Positions des véhicules sur le rond-point (t_max = 15 s)
X (m)
Y (m)


Positions des véhicules sur le rond-point (t_max = 20 s)
X (m)
Y (m)


Positions des véhicules sur le rond-point (t_max = 22 s)
X (m)
Y (m)


Véhicule 1
Véhicule 2
Véhicule 3
Véhicule 4
Véhicule 5
Véhicule 6
Véhicule 7
Véhicule 8
Véhicule 9
Véhicule 10
Centre du rond-point

Annexe

- https://www.youtube.com/watch?v=wHz6S2dbYb4&pp=ygUbY2VzdCBwYXMgc29yY2llciByb25kIHBvaW50
- https://www.researchgate.net/publication/328637594_TIPE_modelisation_mathematique_du_trafic_routier
- https://www.researchgate.net/publication/331744735_TIPE_modelisation_mathematique_du_trafic_routier_-Complements_sur_l'approche_macroscopique