

# Modélisation d'un trafic routier congestionné, estimation et optimisation de la perte énergétique sur une voie rapide.

Comment modéliser une voie rapide afin d'estimer la perte  
énergétique de l'ensemble du trafic ?

# **Sommaire**

## **I. Modélisation du trafic**

- Discrétisation informatique
- Principe du code

## **II. Aspect Énergétique**

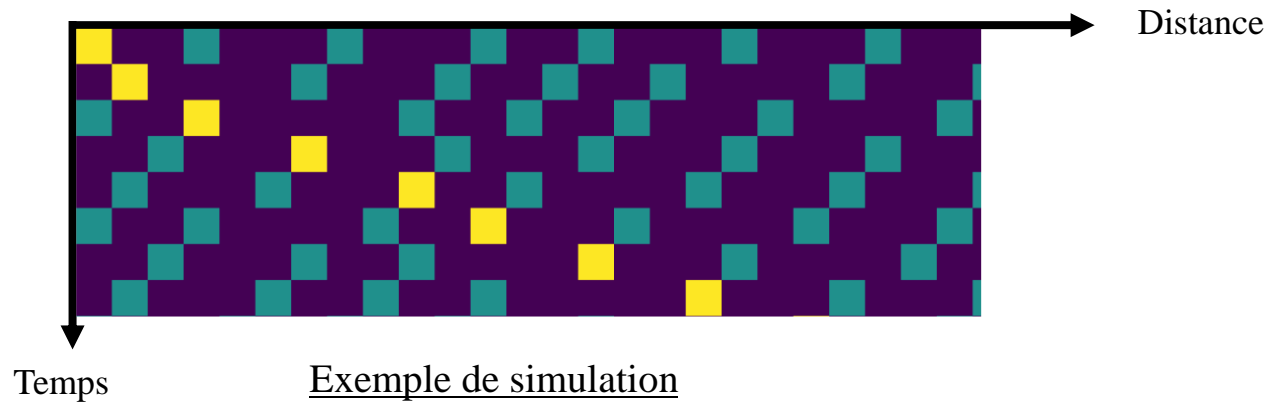
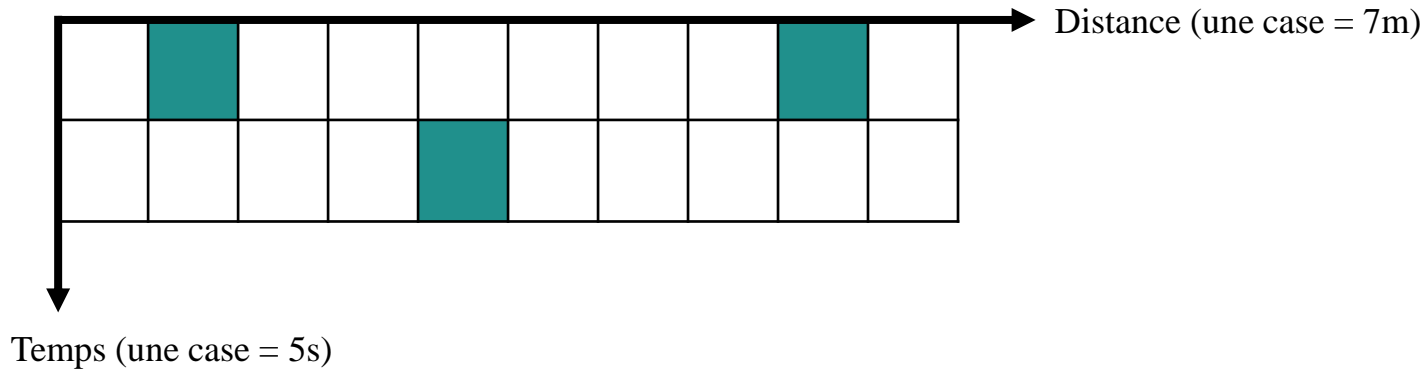
- Expressions énergétiques
- Détermination de la Puissance liée au contact avec la route
- Détermination expérimentale du coefficient de résistance au roulement

## **III. Analyse de différents résultats**

- Vérification de la cohérence du modèle informatique
- Influence de la limitation de vitesse
- Étude de la dépense énergétique
- Recherche d'une vitesse optimale

# I Modélisation du trafic

# Discrétisation informatique

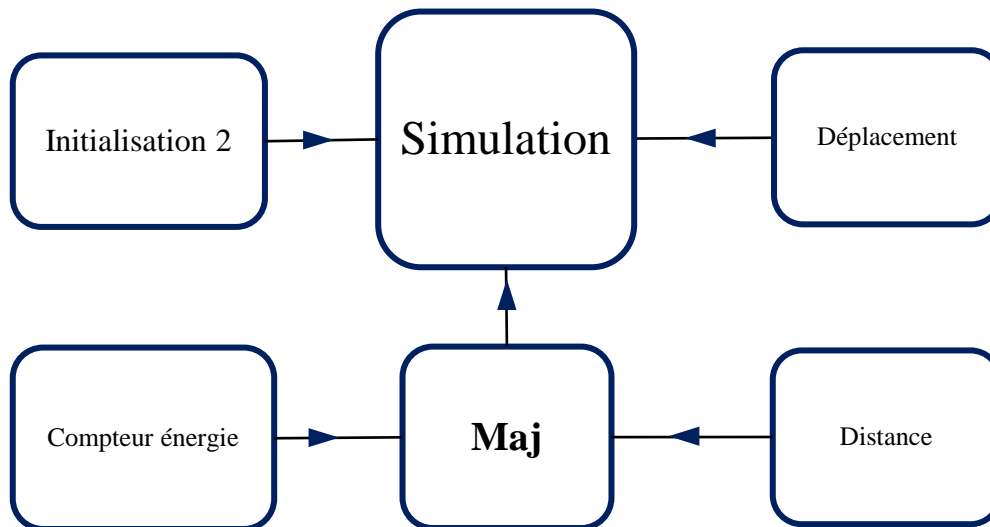


# Principe du code

- Modèle de poursuite en conduite libre

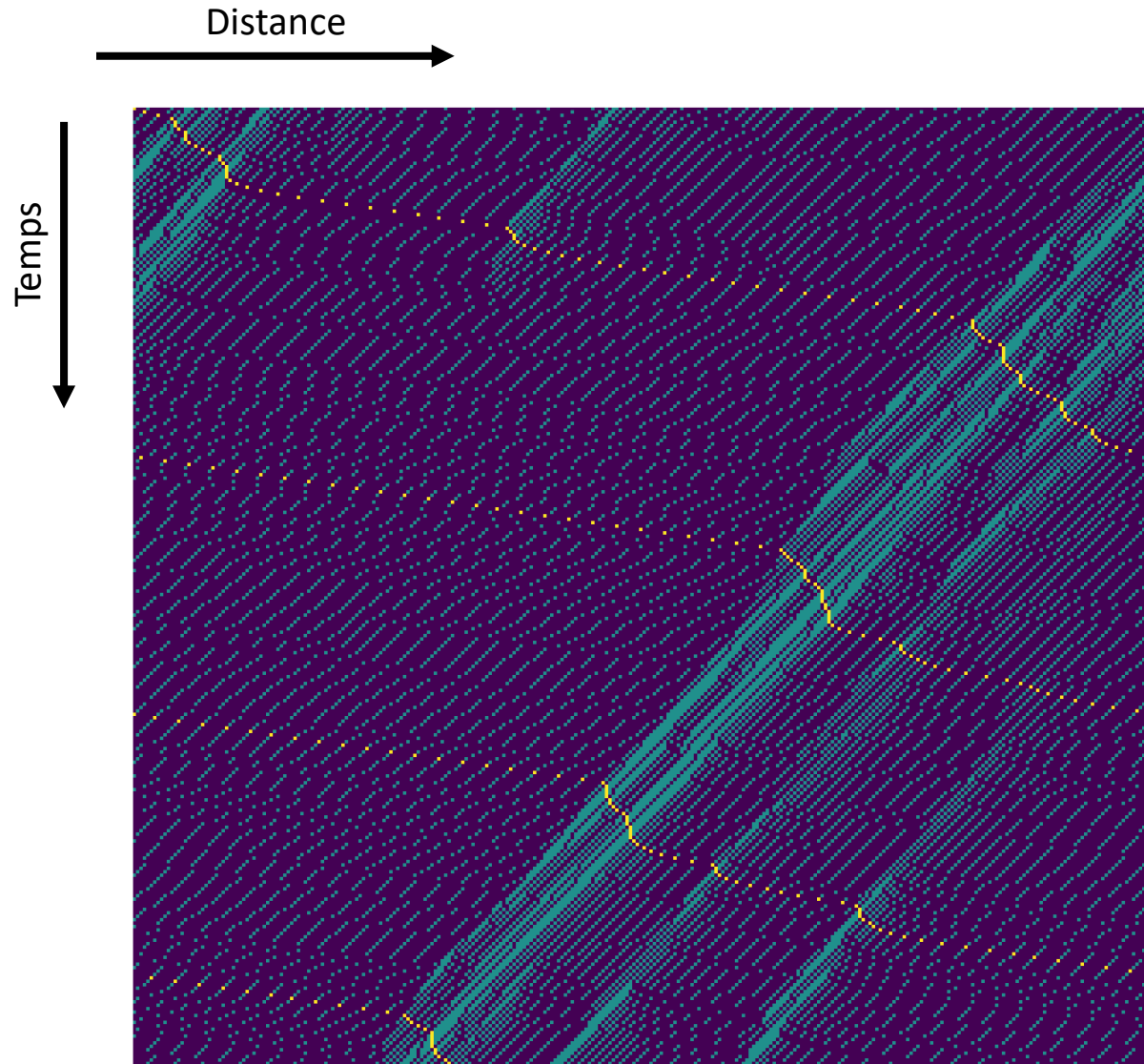
## Étapes de la simulation :

1. Initialisation de la route (*initialisation\_simulation\_2*)
2. Mise à jour des vitesses (*maj*)
3. Calcul de l'énergie dépensée (*compteur\_énergie*)
4. Mise à jour des positions (*déplacement / distance*)



# Exemple de simulation :

Exemple sur 2,1km et 25 min.



## II Aspect Énergétique

## Aspect énergétique:

Sur  $\Delta t = 5 \text{ s}$ , 3 cas possibles :

- À l'arrêt :  $\Delta E = \Delta E_{\text{arrêt}}$
- Vitesse constante :  $\Delta E = \Delta E_{\text{frott roulement}} + \Delta E_{\text{frott air}}$
- Accélération :  $\Delta E = \Delta E_{\text{cinétique}} + \Delta E_{\text{frott roulement}} + \Delta E_{\text{frott air}}$

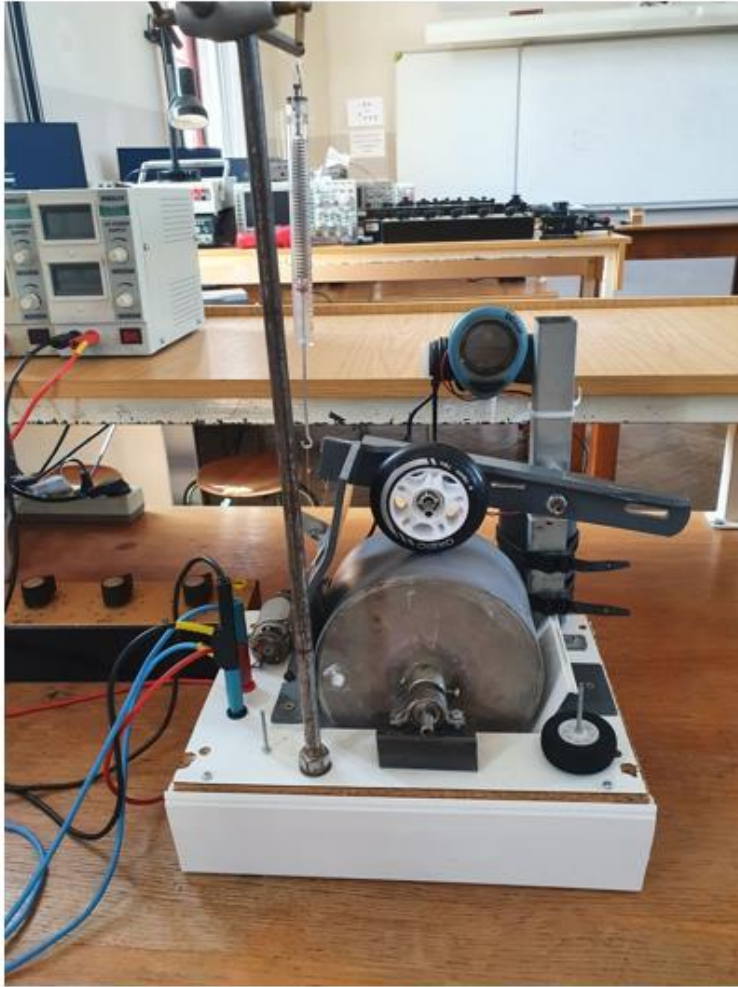


## Expressions énergétiques:

- Énergie à l'arrêt :  $\Delta E_{\text{arrêt}} = \Delta V_{\text{essence}} \Delta t \times P_c = 47.10^3 \text{ J}$
- Énergie cinétique :  $\Delta E_{\text{cinétique}} = \frac{1}{2} m (v_{i+1}^2 - v_i^2) \frac{1}{\eta_{\text{moteur}}}$
- Énergie due à la trainée : Force de trainée :  $F = \frac{1}{2} \rho S C_x v^2$   
$$\Delta E_{\text{frottair}} \approx \frac{\rho C_x S}{2} \left( \frac{v_{i+1} + v_i}{2} \right)^3 \Delta t \times \frac{1}{\eta_{\text{moteur}}}$$
- Expression de l'énergie due au frottement de la route :

- $\Delta E_{\text{frottroulement}} = P \times \Delta t \quad \text{avec } P = P(F_p, v) ?$

# Détermination de la Puissance liée au contact avec la route :



Dispositif expérimental

1. Roue de la voiture
2. Roue modélisant le contact roue-route

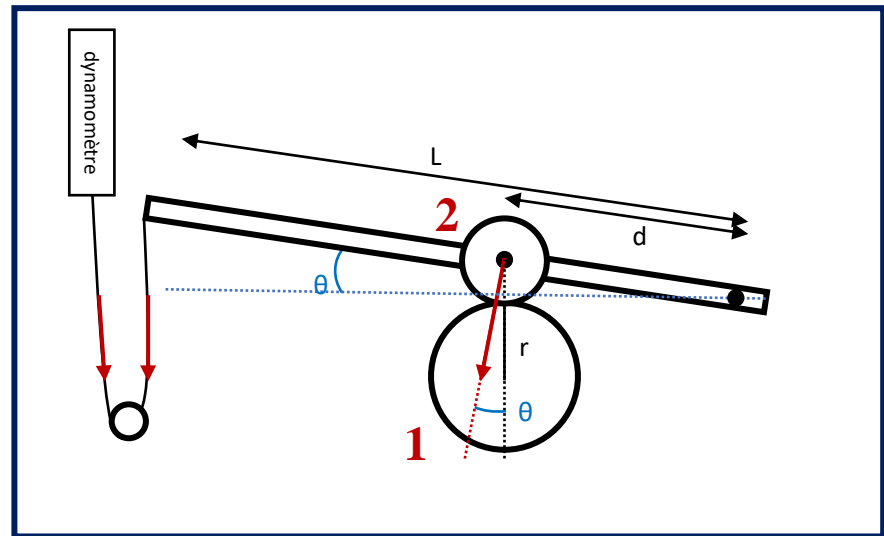


Schéma du dispositif expérimental

$$\Delta E_{frott_{roul}} = \mu_r \times F_p \times v \times \Delta t \times \frac{1}{\eta_{moteur}}$$

# Détermination expérimentale du coefficient de résistance au roulement

- Théorème de la puissance cinétique
- On néglige les effets de la force de trainée.

$$\begin{aligned}\frac{dE_c}{dt} = \sum P &\Leftrightarrow \frac{1}{2} m \frac{dv^2}{dt} = -\mu_r \cdot m \cdot g \cdot v \\ &\Leftrightarrow \frac{1}{2} 2v \frac{dv}{dt} = -\mu_r \cdot g \cdot v \\ &\Leftrightarrow \frac{dv}{dt} = -\mu_r \cdot g\end{aligned}$$

On obtient une durée moyenne de **14,7s**, avec un écart-type sur la moyenne de **0,83s**.

Ainsi :

$$\mu_r = \frac{-\Delta v}{\Delta t \times g}$$

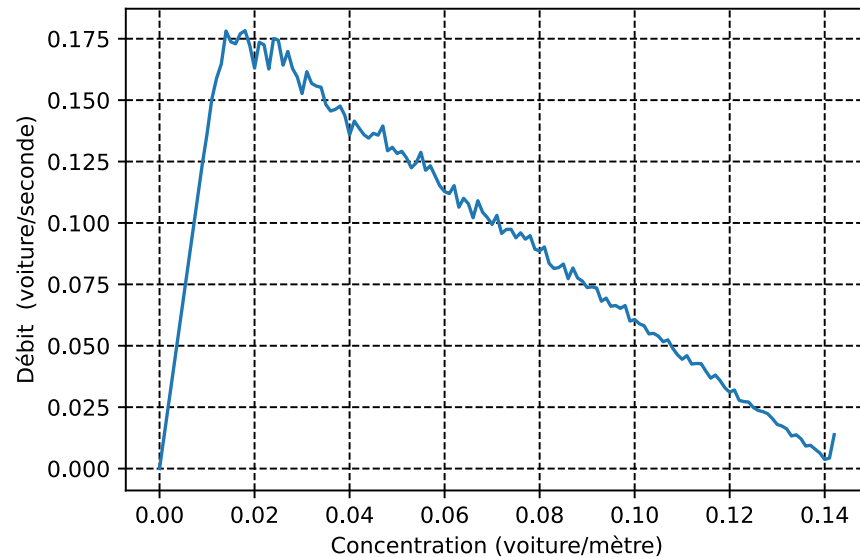
Donc :  **$\mu_r=0,07$**  pour le véhicule considéré.

On prend une valeur plus faible en prenant en compte la force de trainée.

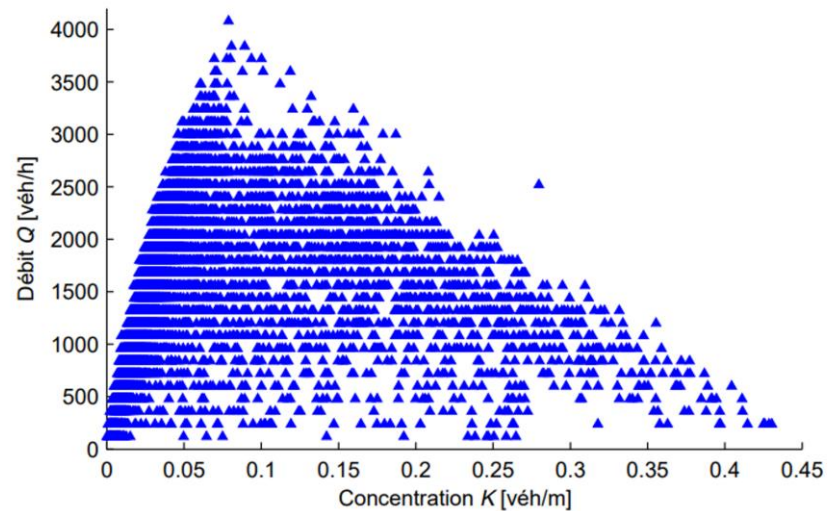
# III Analyse de différents résultats

# Vérification de la cohérence du modèle informatique

Débit de véhicules en fonction de la concentration

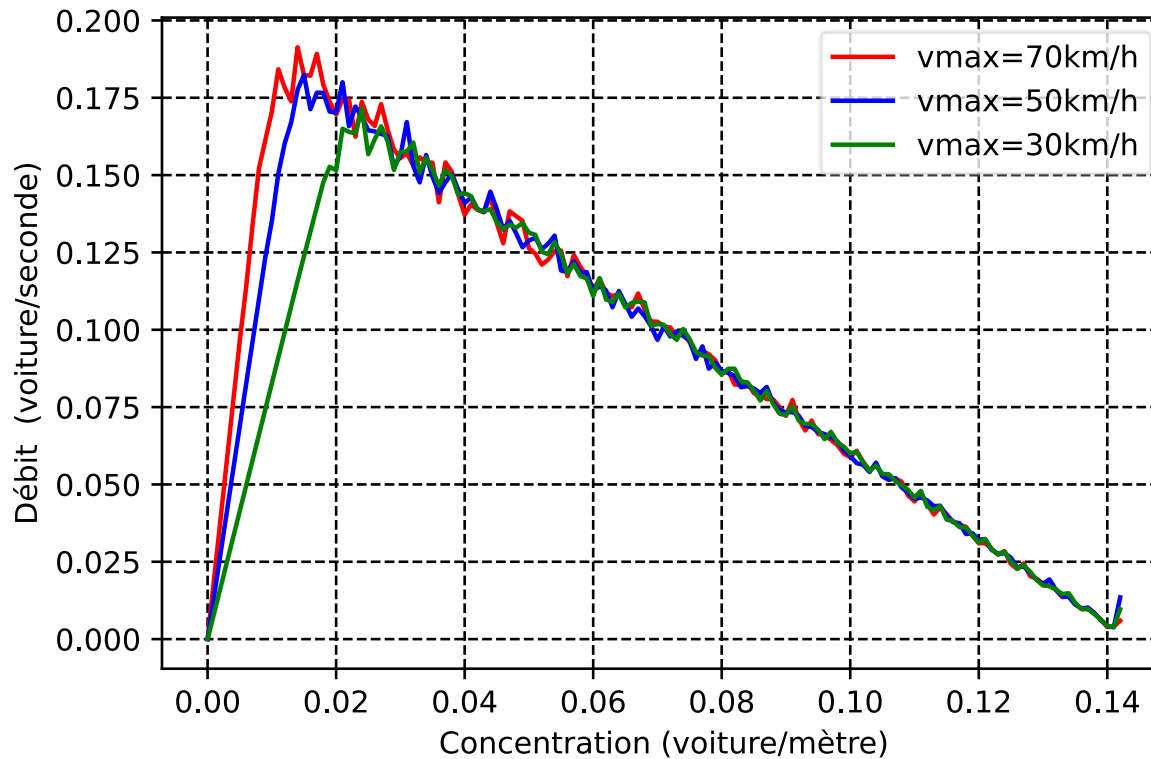


Données expérimentales



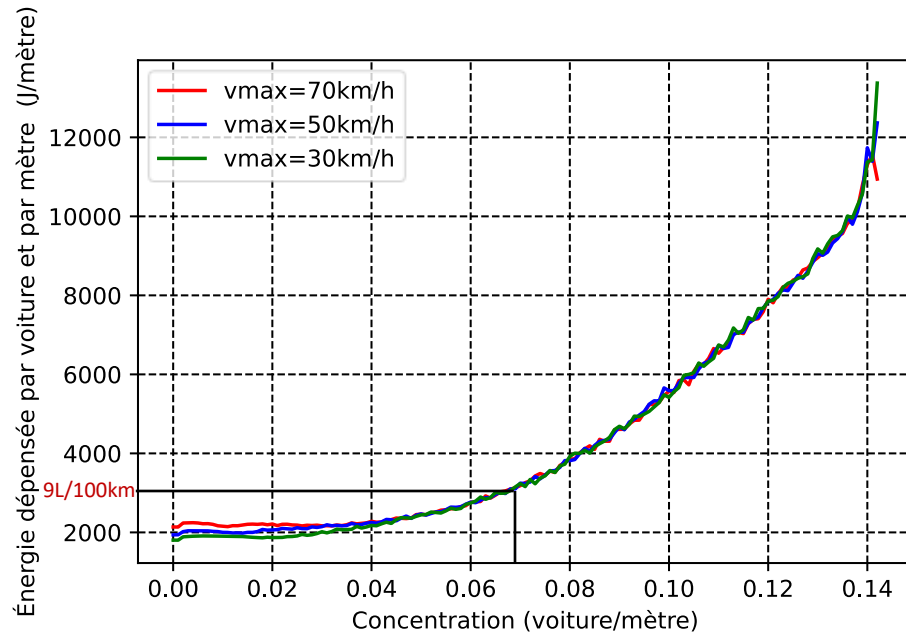
*[Ludovic Leclercq, Modélisation du trafic et applications  
à l'estimation du bruit routier, INSA Lyon 2002]*

# Influence de la limitation de vitesse ( $v_{\max}$ )

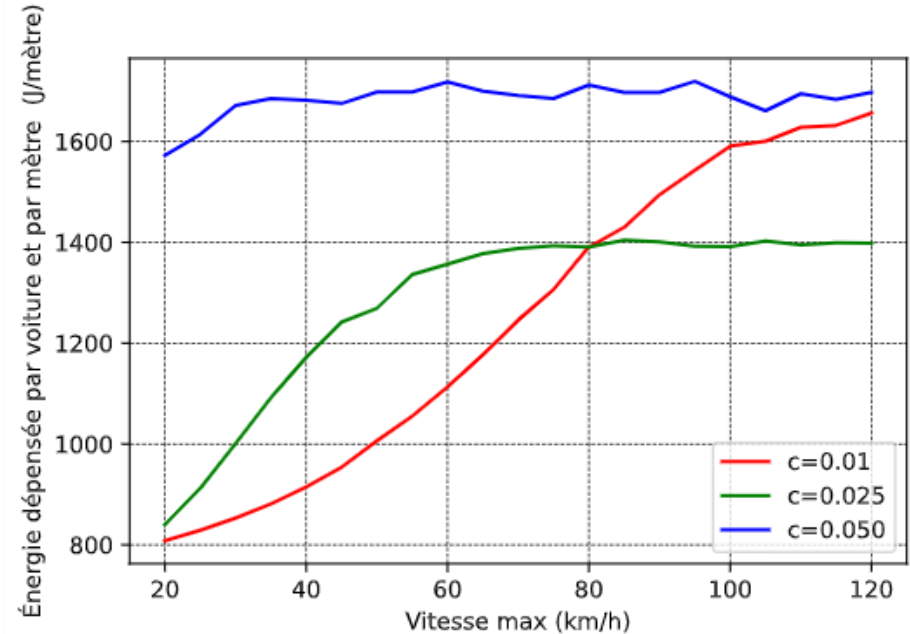


Débit de véhicule en fonction de la concentration

# Étude de la dépense énergétique



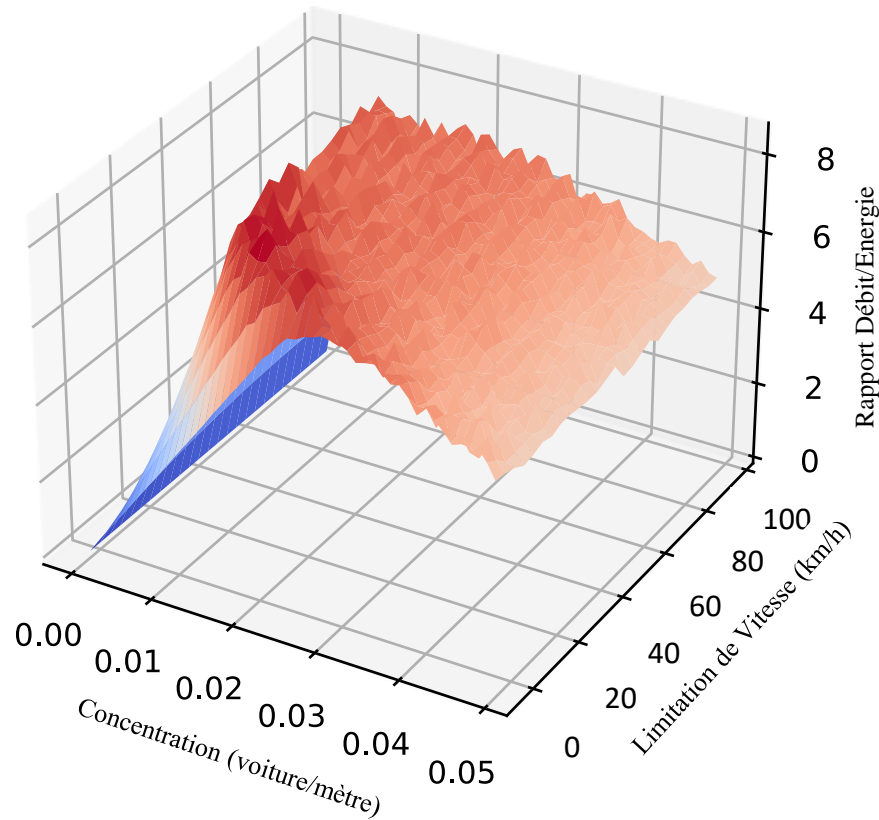
Énergie dépensée par véhicule et par mètre en fonction de la concentration



Énergie dépensée par véhicule et par mètre en fonction de la vitesse max pour différentes concentrations

# Recherche d'une vitesse optimale

Rapport Débit/Énergie en fonction de la concentration et de  $v_{max}$ .



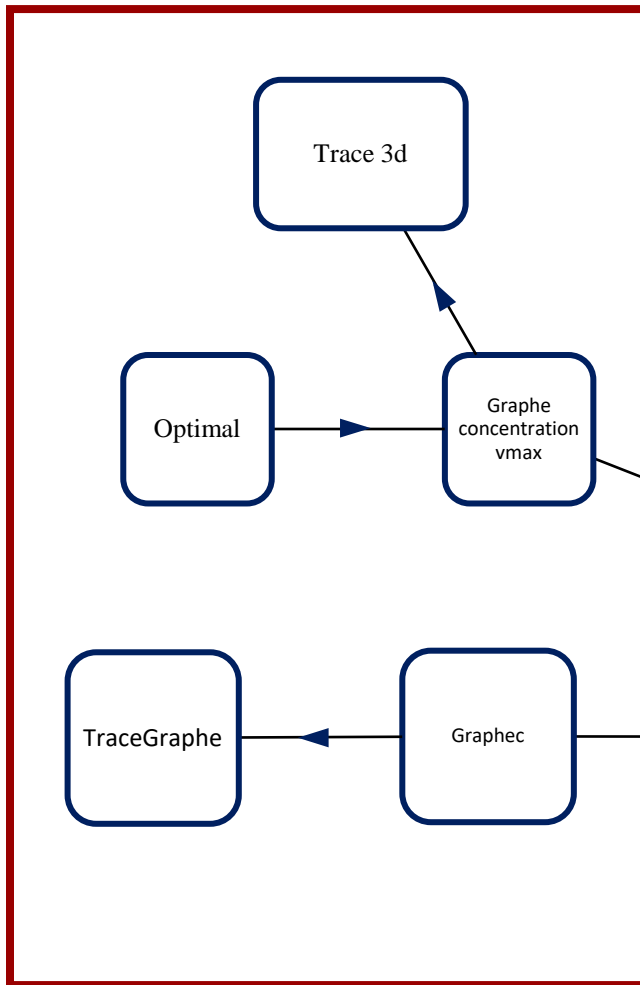


## Conclusion :

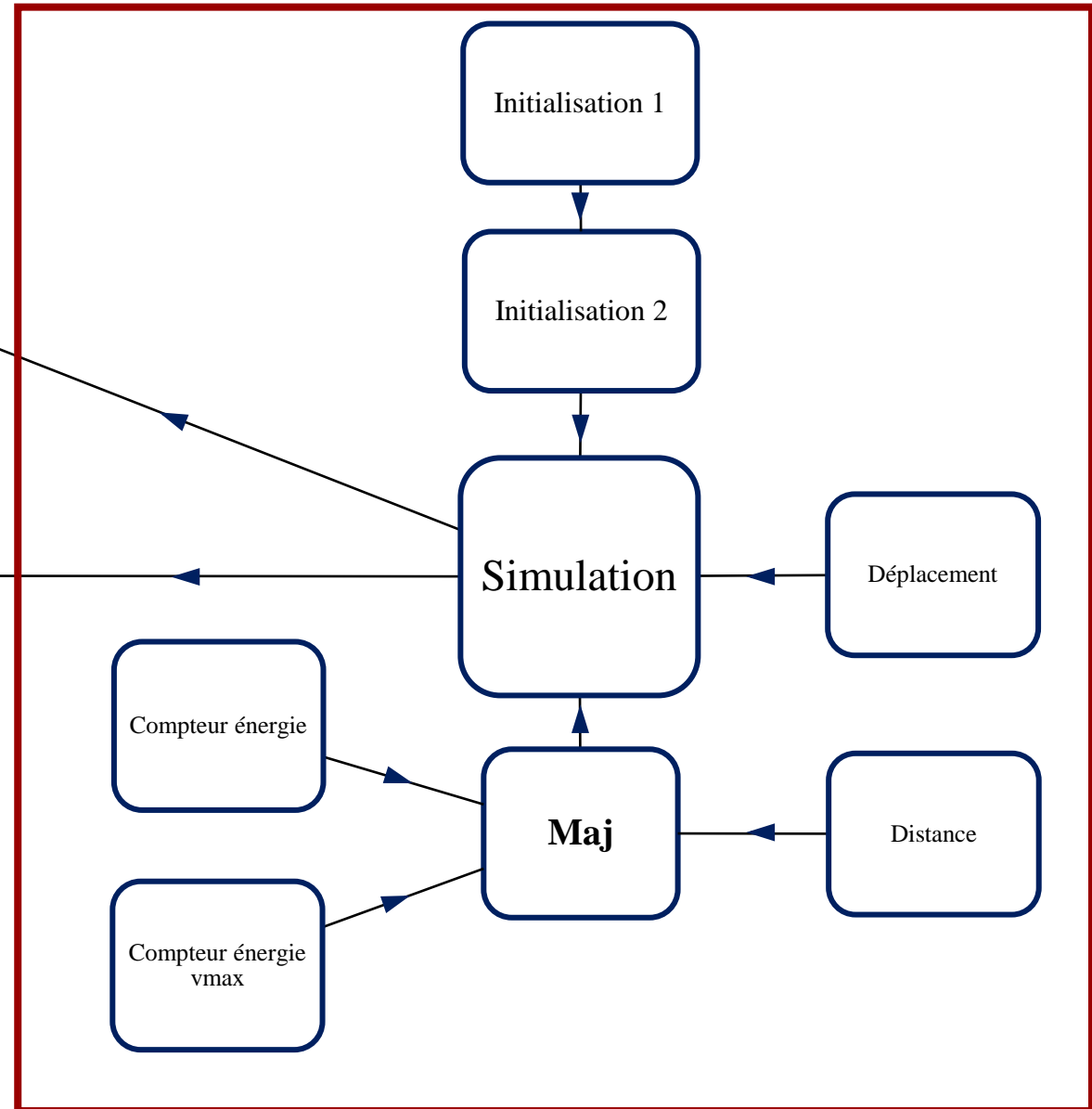
- Mettre en place le principe de distance de sécurité variant selon la vitesse.
- Réaliser la simulation sur un route à plusieurs voies et permettre les dépassements.
- Inclure des voies d'insertions et de décélération afin de faire varier la concentration en véhicule le long de la portion de route.
- Pour la recherche d'une vitesse optimale, selon la récurrence de certaines concentrations leur attribuer un « poids ».

# Annexes

## Ensemble des fonctions constituant la simulation



Fonctions traçant les différents graphes



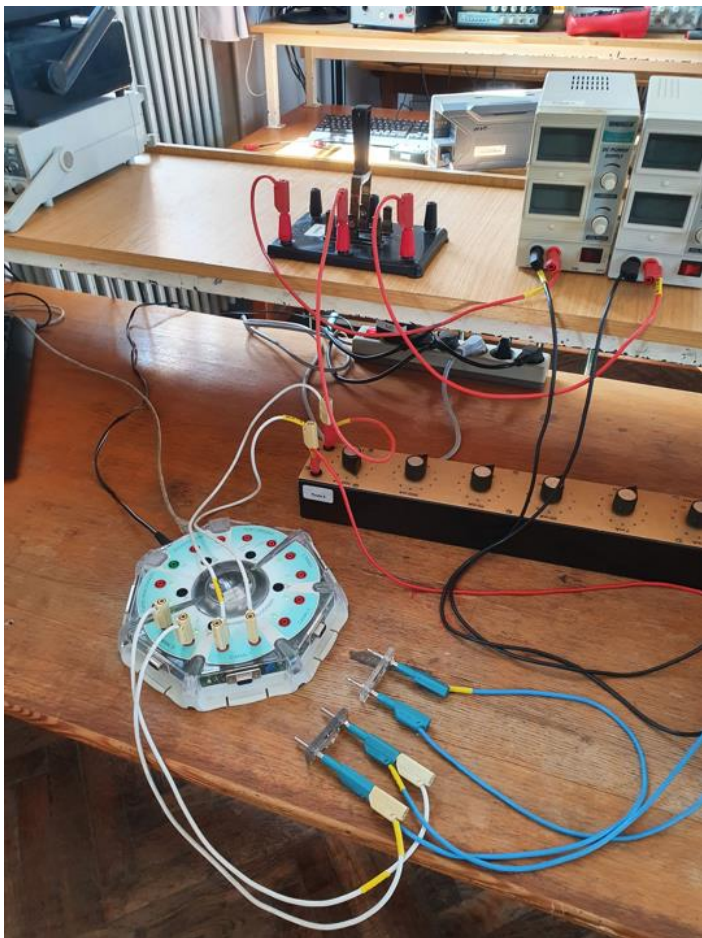


Schéma du circuit électrique permettant d'obtenir la puissance utilisée par le moteur

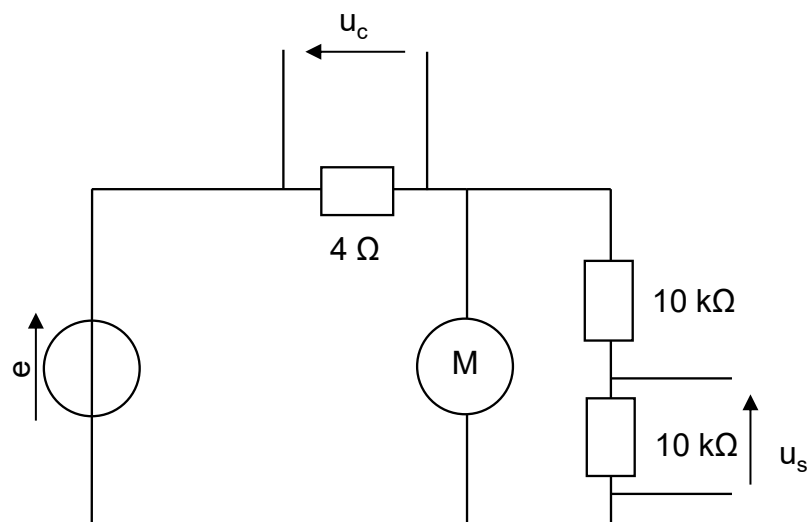


Tableau des mesures des différents  $\Delta t$

$\Delta t$	19	16	14	15
13	17	17	11	14
11				

Calcul de la moyenne	Calcul de l'écart-type sur la mesure	Calcul de l'écart-type sur la moyenne
$\langle \Delta t \rangle = \frac{1}{N} \sum_i \Delta t_i = 14.7s$	$\sigma_{\Delta t} = \sqrt{\frac{1}{N-1} \cdot \sum_i (\Delta t_i - \langle \Delta t \rangle)^2} = 2.63s$	$\sigma_{\langle \Delta t \rangle} = \frac{\sigma_{\Delta t}}{\sqrt{N}} = 0.83s$

▼ Dépense à l'arrêt

Consommation moyenne à l'arrêt :  $0,6 L_{essence} \cdot h^{-1} \cdot L_{cylindrée}^{-1}$  or cylindrée moyenne :  $1523cm^3 = 1,523L$

$$\Delta L_{essence\ case} = \frac{0.6}{3600} \times 1.523 \times \Delta t (= 5.04s) = 1.28 \cdot 10^{-3} L.case^{-1}$$

$$\Delta E_{arrêt} = \Delta L_{essence\ case} \times P_c = 1.28 \cdot 10^{-3} \times 36.8 \cdot 10^6 = 47.10^3 J.case^{-1}$$



# 1 Initialisation

## 1.1 Paramètres et Bibliothèques

```
import matplotlib.pyplot as plt      #imports de bibliothèques
import random
import numpy as np
import matplotlib.image as img
from IPython.display import Image

eta=0.414      #rendement du moteur
l_case=7      #longueur de la case en mètres
dt=5
rho=1.2      #masse volumique de l'air ( kg.m-3)
Cx=0.40
S=1.9
mu=0.02
m=1400      #masse d'une voiture (kg)
g=9.8
p=0.06      #facteur régissant les freinages aléatoires

#initialisation des compteurs
nb_voitures=[0] #compteur de voiture sur la section
energie=[0] #dépense énergétique
nbparcours=[0] #nombre de parcours de chaque voiture
```

## 1.2 Initialisation de la route

```
def initialisation_simulation_1(longueur,temps,Concentration,vmax):
    """ crée une première répartition plus naturelle de la route afin
        d'éviter des "effets de bords" en début de simulation due à la répartition
        initiale aléatoire"""

    Route=[[0 for i in range(longueur)]for j in range(temps)] #initialisation tableau route
    Vitesses=[[0 for i in range(longueur)]for j in range(temps)] #initialisation tableau vitesses

    Proba=Concentration*l_case #probabilité de trouver une voiture par case

    Route[0]=[random.randint(1,100) for i in range(len(Route[0]))] #création de la route à t=0
    for i in range(len(Route[0])):
        if Route[0][i]<=Proba*100: #présence ou non d'une voiture selon la concentration fixée
            Route[0][i]=1
        else :
            Route[0][i]=0

    for i in range(len(Route[0])):#répartition aléatoire de vitesses initiales
        if Route[0][i]==1:
            Vitesses[0][i]=random.randint(vmax//2,vmax-2)
    return Route, Vitesses

def initialisation_simulation_2(longueur,temps,Concentration,p,vmax):
    """utilise la première initialisation pour obtenir une route initiale
        plus naturelle"""
    Route, Vitesses=initialisation_simulation_1(longueur,21,Concentration,vmax)
    for i in range(len(Route)-1): #exécution de la simulation sur route initiale
        Route,Vitesses=deplacement(Vitesses, Route, maj(Route,Vitesses,p,vmax,i), i,vmax)

    Route2=[[0 for i in range(longueur)]for j in range(temps)] #initialisation route
    Vitesses2=[[0 for i in range(longueur)]for j in range(temps)] #initialisation vitesses
    Route2[0]=Route[20] #création d'une nouvelle route naturelle

    Vitesses2[0]=Vitesses[len(Vitesses)-1]
    Route2[0][0]=2
    Vitesses2[0][0]=random.randint(vmax//2,vmax-2)
    energie=[0]
    return Route2, Vitesses2
```



## 2 Corps du programme

```
def maj(Route,Vitesses,p,v_max,i):
    """ met à jour la liste de voiture et de vitesse et le compteur énergétique"""
    Vitesses_suivantes = np.array(Vitesses[i]) # Copie de l'état actuel
    N = len(Vitesses_suivantes)
    for j in range(N):
        if Route[i][j] == 1 or Route[i][j] == 2 : # Si il y a une voiture dans la case
            # Étape 1: Décélération si écart trop faible avec la vitesse devant
            dn = distance(Route,i,j)
            if Vitesses_suivantes[j] > dn - 1:
                Vitesses_suivantes[j] = dn - 1
                if Vitesses_suivantes[j]==0:
                    compteurenergie(Vitesses_suivantes,j,v_max)
            # Étape 2: Accélération
            elif Vitesses_suivantes[j] < v_max and Vitesses_suivantes[j] < dn-1:
                #si l'utilisateur peut encore accélérer
                Vitesses_suivantes[j] = Vitesses_suivantes[j] + 1
                compteurenergie(Vitesses_suivantes,j,v_max)
            elif Vitesses_suivantes[j]==v_max :
                compteurenergie_vmax(Vitesses_suivantes,j,v_max)
            # Étape 3: Facteur aléatoire de freinage
            if np.random.rand() < p and Vitesses_suivantes[j] > 0:
                Vitesses_suivantes[j] = Vitesses_suivantes[j] - 1
    return Vitesses_suivantes
```

## 2.1 Calcul d'énergie

```
def compteurenergie(Vitesses_suivantes,j,v_max):
    Vitesse=Vitesses_suivantes[j] #Vitesse suivante : chiffre entre 1 et v_max, tranche de 5 km/h
    energie[0]+= (mu*m*g*l_case)*Vitesse*1/eta
    if Vitesse !=0 :
        energie[0] +=(rho*Cx*S/2)*(((Vitesse+Vitesse-1)*5/(3.6*2))**3)*dt*1/eta
    elif Vitesse == 0:
        energie[0]+=5*10**4 #énergie dépensée par le moteur à l'arrêt cf annexe pour le calcul
    if Vitesse <= v_max and Vitesse !=0 :
        energie[0]+=1/2*m*((Vitesse*5/3.6)**2-(((Vitesse-1)*5/3.6)**2))*1/eta

def compteurenergie_vmax(Vitesses_suivantes,j,v_max):
    """ l'énergie est calculée après mise à jour des vitesses,
        le passage de vmax-1 à vmax est traité séparément"""

    Vitesse=Vitesses_suivantes[j] #Vitesse suivante : v_max,
    energie[0]+= (mu*m*g*l_case)*Vitesse*1/eta
    if Vitesse !=0 :
        energie[0] +=(rho*Cx*S/2)*(((Vitesse)*5/(3.6))**3)*dt*1/eta
```

## 2.2 Calcul du prochain déplacement

```
def distance(Route,i,j):
    """calcul du nombre de case entre deux véhicules"""
    N = len(Route[i])
    d = 1
    while Route[i][(j+d)%N] != 1 and Route[i][(j+d)%N] != 2:
        # Tant qu'on ne rencontre pas d'autre voiture,
        d = d+1 # on continue à regarder devant soi
    return d

def deplacement(Vitesses,Route,Vitesses_suivantes,i,vmax):
    """ prédit la position en fonction de l'état de la route"""
    N = len(Vitesses_suivantes)
    for j in range(N):
        if Route[i][j] == 1: # Si il y a une voiture,
            # on prédit sa nouvelle position
            prochain = (j + Vitesses_suivantes[j])%N
            # et on met à jour

            Route[i+1][prochain] = 1
            Vitesses[i+1][prochain] = Vitesses_suivantes[j]

        elif Route[i][j] == 2 : #si il y a la voiture
            # on prédit sa nouvelle position
            prochain = (j + Vitesses_suivantes[j])%N
            # et on met à jour
            Route[i+1][prochain] = 2
            Vitesses[i+1][prochain] = Vitesses_suivantes[j]

    return Route, Vitesses
```

## 2.3 Exécution globale

```
def simulation(p,Concentration,longueur,temps,vmax):
    """ fonction-mère qui exécute la simulation"""

    energie[0]=0
    nbparcours[0]=0
    indicev2=0

    Route2,Vitesses2 = initialisation_simulation_2(longueur, temps, Concentration,p,vmax)

    for i in range(len(Route2)-1): #exécution de la simulation
        Route2,Vitesses2=deplacement(Vitesses2, Route2, maj(Route2,Vitesses2,p,vmax,i), i,vmax)
    for j in range(len(Route2[0])):
        if Route2[i][j]==2:
            if j<indicev2:
                nbparcours[0]+=1
                indicev2=j

    positionfinale=0
    for i in range(len(Route2[0])-1):
        if Route2[len(Route2)-1][i]==2:
            positionfinale=i

    if positionfinale==0:
        nbparcours[0]+=1
    else :
        nbparcours[0]+=(positionfinale/len(Route2[0]))
    #permet grâce à la position finale de la voiture marquée de savoir combien de parcours
    #ont été réalisés

    plt.imshow(Route2)
    plt.show()    #affiche l'évolution de la route

    compi=0
    compf=0
    for j in range(len(Route2[len(Route2)-1])):
        if Route2[len(Route2)-1][j]==1 or Route2[len(Route2)-1][j]==2:
            compf+=1
    for j in range(len(Route2[0])):
        if Route2[0][j]==1 or Route2[0][j]==2:
            compi+=1
```

```

nbvoitureinitiale=comp_i
nbvoiturefinale=comp_f

Route3=np.array(Route2)
distancetotale_m=nbparcours[0]*l_case*longueur
tempstotal_s=5.04*temps
vitesse moyenne=distancetotale_m/tempstotal_s
# energie dépensée par voiture, par mètres
energie_ind=(energie[0])/(nbvoiturefinale*distancetotale_m)
print("Une voiture a parcourue ",nbparcours[0],"boucles")
print("soit",distancetotale_m*0.001,"kilomètres en ",tempstotal_s/60,"minutes.")
print(" La vitesse moyenne est donc de,",3.6*vitesse moyenne,"km/h.")
print("La consommation totale est de ",energie[0]*1E-9,"Gigajoules")
print(nbvoitureinitiale,nbvoiturefinale)

return energie_ind, vitesse moyenne

def parcours(Route,t,vmax):
    """ permet de prendre en compte la voiture 2, compteur"""
    for presence in range(vmax):
        if Route[t][presence] == 2:
            return True
    return False

```

## 3 Graphes

### 3.1 Graphes 2D

```
def graphec(bornes:tuple,vmax,Concentration,longueur,temps):
    """ renvoie énergie en fonction de concentration sous forme de deux listes"""
    #bornes contient l'intervalle [cmin,cmax] et le pas de calcul
    #bornes=(cmin,cmax,pas)
    mini,maxi,pas=bornes
    c=mini
    Liste_simulation=[]
    C=[]
    E=[]
    while c<=maxi:
        for i in range(10):
            energie,vitesse moyenne=simulation(p,c,longueur,temps,vmax)
            if (energie > 1e5 or energie == 0) and Liste_simulation != []:
                energie = Liste_simulation[-1]
            elif energie > 1e5 or energie == 0 and Liste_simulation == []:
                energie=E[-1]
            Liste_simulation.append(energie)
        E.append(np.mean(Liste_simulation))
        C.append(c)
        print(int((c*100/(maxi-mini))),'%')
        c+=pas
        Liste_simulation=[]
    return C,E
```

```
def trace_graphe(fonction,bornes:tuple,vmax,Concentration,longueur,temps):
    """ trace énergie en fonction d'un paramètre ( C ici par exemple ) """
    P,E=fonction(bornes,vmax,Concentration,longueur,temps)
    plt.plot(P,E)
    plt.grid(color='black', linestyle='--', linewidth=0.5)
    plt.xlabel("Concentration (voiture/mètre)")
    plt.ylabel("Vitesse moyenne (km/h)")
    plt.show()
    plt.show()
```

## 3.2 Graphes 3D

```
def graphe_concentration_vmax(bornes_concentration:tuple,bornes_vmax:tuple,longueur,temps):
    #bornes_concentration contient l'intervalle [cmin,cmax] et le pas de calcul
    #bornes_vmax contient l'intervalle [vmax(min),vmax(max)] et le pas de calcul
    #bornes_concentration=(cmin,cmax,pas)
    cmini,cmaxi,cpas=bornes_concentration
    vmini,vmaxi,vpas=bornes_vmax
    c=cmini
    v=vmini
    v2=vmini
    Liste_simulation=[]
    Energie_temp=[]
    E=[]
    C=np.arange(cmini,cmaxi,cpas)
    V=np.arange(vmini,vmaxi,vpas)
```



```

C3=[[C[j] for i in range(len(V))] for j in range(len(C))]
V3=[[V[i] for i in range(len(V))] for j in range(len(C))]
R=[[ ] for i in range (len(C))]
for i in range(len(C)):
    for j in range(len(V)):
        for k in range(50):
            energie,vitesse moyenne=simulation(p,C[i],longueur,temps,V[j])
            if (energie > 1e5 or energie == 0) and Liste_simulation != []:
                energie = Liste_simulation[-1]
            elif energie > 1e5 or energie == 0 and Liste_simulation == []:
                energie=E[-1]
            Liste_simulation.append((C[i]*vitesse moyenne)/(energie))
            Energie_temp.append(energie)
        E.append(np.mean(Energie_temp))
        R[i].append(np.mean(Liste_simulation))
        Liste_simulation=[]
        Energie_temp=[]
Vitesseopti=optimal(C3,V3,R)
return C3,V3,R,Vitesseopti

```

```

def trace_3d(bornes_concentration,bornes_vmax,longueur,temps):

```

```

    C,V,R,Vitesseopti=graphe_concentration_vmax(bornes_concentration,bornes_vmax,longueur,temps)

```

```

    C2=np.array(C)

```

```

    V2=np.array(V)

```

```

    R2=np.array(R)

```

```

    print("La vitesse optimale est : ",Vitesseopti*5,"km/h")

```

```

    fig = plt.figure()

```

```

    ax = fig.gca(projection='3d') # Affichage en 3D

```

```

    ax.plot_surface(C2, V2, R2, cmap=cm.coolwarm, linewidth=2) # Tracé d'une surface

```

```

    ax.set_xlabel('')

```

```

    ax.set_ylabel('')

```

```

    ax.set_zlabel('')

```

```

    plt.tight_layout()

```

```

    plt.show()

```



```

def optimal(C,V,R):
    maximum=0
    Liste_Moyenne=[]
    Liste_temp=[[ for i in range(len(R[0]))]
    Vitesseopti=0
    for j in range (len(R[0])):
        for i in range (len(R)):
            Liste_temp[j].append(R[i][j])
    for k in range(len(Liste_temp)):
        Liste_Moyenne.append(np.mean(Liste_temp[k]))
    for j in range(len(Liste_Moyenne)):
        if Liste_Moyenne[j]>maximum:
            maximum=Liste_Moyenne[j]
            Vitesseopti=V[0][j]
    return Vitesseopti

```