Corrigé

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

Sélection des mesures

Q1. On effectue une jointure entre la table des comptages et la table des stations pour trouver la station voulue :

```
SELECT id_comptage, date, voie, q_exp, v_exp
FROM stations JOIN comptages
ON stations.id_station = comptage.id_station
WHERE stations.nom = "M8B"
```

Q2. Pour utiliser la fonction d'agrégation `SUM`, on regroupe les comptages par date.

```
SELECT date, SUM(q_exp)
FROM COMPTAGES_M8B
GROUP BY date
```

Diagramme fondamental

Q3. On trace les débits en fonction de la concentration.

[2]:
```
def trace(q_exp,v_exp):
    n = len(v_exp)        # Taille des vecteurs utilisés
    c_exp = zeros(n)     # Initialisation du vecteur des concentrations
    for i in range(n):
        c_exp[i] = q_exp[i]/v_exp[i]
        plt.plot(c_exp,q_exp,'o')  # on trace un point
    plt.show()
```

Version utilisant **numpy**

[3]:
```
def trace(q_exp,v_exp):
    c_exp = q_exp / v_exp # Calcul en utilisant des vecteurs numpy
    plt.plot(c_exp,q_exp,'o')
    plt.show()
```

Estimation de l'état de congestion

Q4. On reconnaît l'algorithme de tri par insertion.

[4]:
```
def congestion(v_exp):
    nbmesures = len(v_exp)
    for i in range(nbmesures):  # Pour chaque point de mesure
        v = v_exp[i]            # On note la valeur à classer
        j = i                   # et d'où l'on part.
        while 0 < j and v < v_exp[j-1]: # Tant qu'on n'arrive pas tout à gauche
            # ou qu'on ne trouve pas la place de v,
            v_exp[j] = v_exp[j-1]  # on décale vers la droite
            j = j-1                # et on regarde plus à gauche
```

```
        v_exp[j] = v            # on place finalement v au bon endroit
    return v_exp[nbmesures//2]  # et on renvoie l'élément milieu après tri
```

Q5.

Pendant la moitié des observations, il y a eu une vitesse inférieure à 30 km/h alors pendant la moitié du temps au moins, la situation était congestionnée.

Simulation par la mécanique des fluides : discrétisation

Q6.

Pour la discrétisation en espace, il y a $\left\lfloor \dfrac{La}{dx} \right\rfloor$ pas d'espace soit $\left\lfloor \dfrac{La}{dx} \right\rfloor + 1$ positions et $\left\lfloor \dfrac{\text{Temps}}{\text{dt}} \right\rfloor$ pas de temps soit $\left\lfloor \dfrac{\text{Temps}}{\text{dt}} \right\rfloor + 1$ temps. Le tableau C a donc deux dimensions (n, p) avec $n = \left\lfloor \dfrac{La}{dx} \right\rfloor + 1$ et $p = \left\lfloor \dfrac{\text{Temps}}{\text{dt}} \right\rfloor + 1$.

Avec cette discrétisation, si La n'est pas exactement un mulitple de dx, il restera un morceau d'espace non couvert à la fin : si La = 1.0 et dx = 0.3, les positions seront 0.0, 0.3, 0.6, 0.9 et l'intervalle]0.9,1.0] n'est pas couvert.

[5]:
```
La = 8500   # en m : 8,5 km
dx = 50     # en m
Temps = 100 # en s
dt = 1      # en s
n = int(La//dx)+1
p = int(Temps//dt)+1
C = np.zeros((p,n))
```

Modèle de diagramme fondamental

Q7.

On a la relation $v(t,x) = v_{\max}\left(1 - \dfrac{c(t,x)}{c_{\max}}\right)$ pour chaque instant t et position x.

On également la relation $q(t,x) = v(t,x) \times c(t,x)$.

Alors, $q(t_i, x_i) \simeq v_{\max}\left(1 - \dfrac{C_{i,j}}{c_{\max}}\right) C_{i,j}$ où $C_{i,j} \simeq c(t_i, x_i)$

[6]:
```
def debit(v_max, c_max, C_ligne):
    v = v_max * (1 - np.array(C_ligne)/c_max) # on utilise numpy car on a
                                              # convertit en utilisant np.array
    return C_ligne * v


def debit(v_max, c_max, C_ligne):
    q = []
    for k in range(len(C_ligne)):
        q.append(v_max * C_ligne[k] * (1 - (C_ligne[k])/c_max))
    return np.array(q)
```

Q8.

La fonction diagramme nécessite les mêmes informations que la fonction débit précédente.

[7]:
```
def diagramme(v_max, c_max, C_ligne):
    pass
```

On utilise v_max en mètre par seconde, c_max en véhicule par mètre.

Dans le diagramme fondamental de l'énoncé, on trace le débit en véhicule par heure en fonction de la concentration en véhicule par kilomètre.

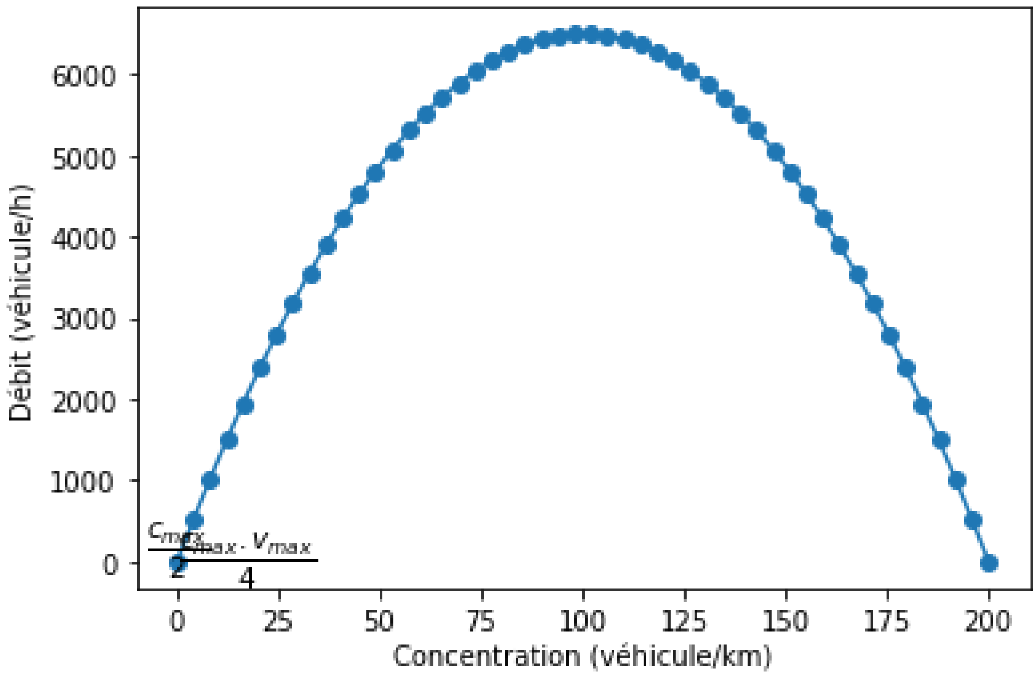
L'allure de la courbe correpond à une parabole car on a la relation entre le débit et la concentration de la forme : $q = v_{\max}\left(1 - \frac{c}{c_{\max}}\right)c$.

La relation ne dépend pas du temps car c'est une relation de la forme $q = \alpha c^2 + \beta c + \gamma$ avec α, β, γ constantes ne dépendant pas du temps.

[8]:
```
# non demandé
def diagramme(v_max, c_max, C_ligne):
    q = debit(v_max, c_max, C_ligne)
    plt.plot(C_ligne*1000, q*3600, marker='o')
    plt.xlabel('Concentration (véhicule/km)')
    plt.ylabel('Débit (véhicule/h)')

    plt.plot((0.04, c_max/2, c_max/2),
         (c_max * v_max / 4, c_max * v_max / 4, 0.2),linestyle=":")
    plt.text(0, c_max * v_max / 4, r'$\frac{c_{max} . v_{max}}{4}$',
         verticalalignment='center', fontsize=15)
    plt.text(c_max/2, 0, r'$\frac{c_{max}}{2}$',
         horizontalalignment='center', fontsize=15)
    plt.show()

v_max, c_max = 130*1000/3600, 200/1000
diagramme(v_max, c_max, np.linspace(0,c_max))
```



Résolution de l'équation : situation initiale

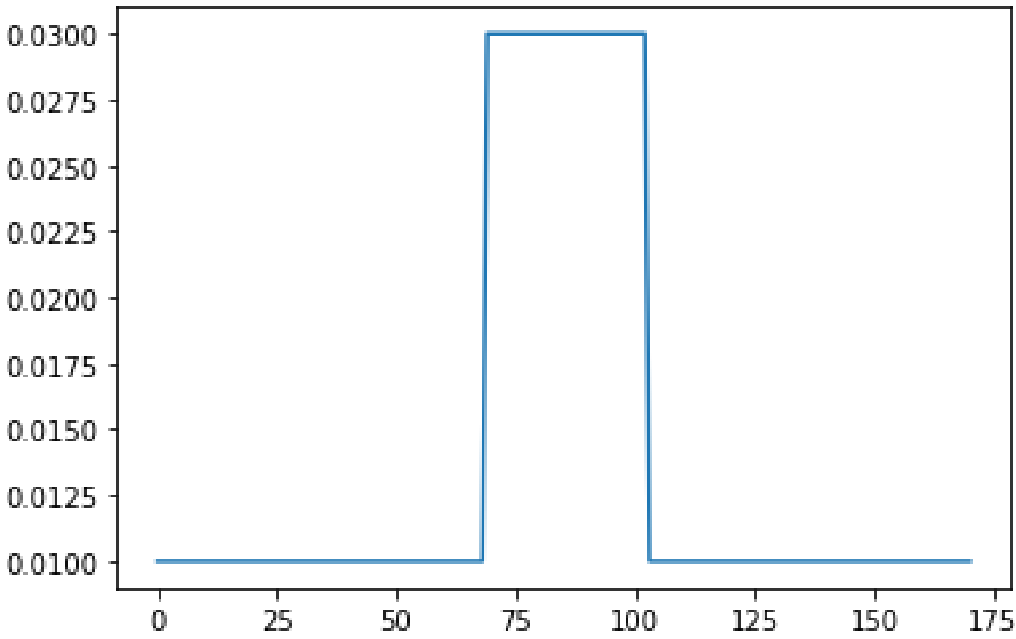
Q9.

La fonction demandée doit valoir c_1 pour les indices appartenant à l'intervalle `[[ 0 ; d1//dx ]]` puis c_2 sur l'intervalle `[[ d1//dx + 1 ; d2//dx ]]` et enfin à nouveau c_1 sur l'intervalle `[[ d2//dx + 1 ; La//dx ]]`.

[9]:
```
# Fonction non demandée
def C_depart(dx,d1,d2,c1,c2,C):
    n1,n2 = int(d1//dx), int(d2//dx)
    C[0,:] = c1  # On initialise tout à c1
    C[0,n1+1:n2+1] = c2 # Et on met à c2 les points de [[n1+1;n2]]
```

[10]:
```
c1 = 0.01 ; c2 = 0.03
d1 = 0.4*La ; d2 = 0.6 * La
C_depart(dx,d1,d2,c1,c2,C)
plt.plot(C[0])
print(n,dx,len(C[0]))
```

```
171 50 171
```



Résolution de l'équation : Résolution

Q10.

On utilise le schéma numérique de la méthode d'Euler : $C_{i+1,j} = C_{i,j} + (t_{i+1} - t_i)\dfrac{\partial c}{\partial t}(t_i, x_i)$

Mais l'équation (1) donne : $\dfrac{\partial c}{\partial t}(t_i, x_i) = -\dfrac{\partial q}{\partial x}(t_i, x_i)$.

On utilise l'approximation appelée ici schéma d'Euler «avant» : $\dfrac{\partial q}{\partial x}(t_i, x_i) \simeq \dfrac{Q_{j+1} - Q_j}{x_{j+1} - x_j}$.

On obtient le schéma numérique demandé

$$(2) \qquad \boxed{C_{i+1,j} = C_{i,j} - \frac{Q_{j+1} - Q_j}{dx} dt}$$

Q11.

La ligne C_i contient toutes les valeurs de concentrations aux différentes positions à l'instant t_i.

La condition aux limites périodique en espace s'écrit en utilisant `Q[j%n]` qui vaut `Q[0]` si $j = n$.

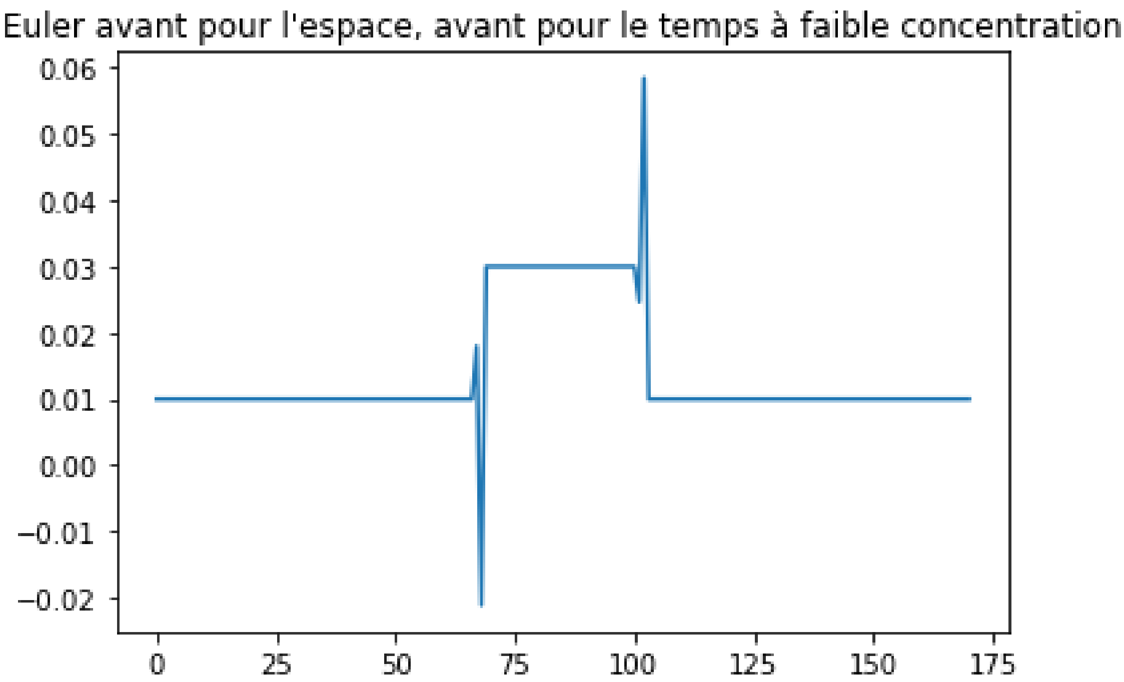
[11]:
```python
def resolution(C, dt, dx, c_max, v_max):
    p,n = C.shape                      # nombre de valeurs en temps (p) et en espace (n)
    for i in range(0,p-1):             # on itère sur tous les temps
        Q = debit(v_max,c_max,C[i]) # calcul des débits à l'instant t_i
        for j in range(n):             # on itère sur tout l'espace
            Qjp1 = Q[(j+1)%n]          # condition aux limites périodique
            C[i+1,j] = C[i,j] - dt * (Qjp1 - Q[j]) / dx  # schéma numérique
        # print(min(C[i+1]))
    # return C                          # inutile car C est modifié en place
```

[12]:
```
resolution(C, dt, dx, c_max, v_max)
```

```
<ipython-input-6-1f8fb0965998>:10: RuntimeWarning: overflow encountered in
double_scalars
  q.append(v_max * C_ligne[k] * (1 - (C_ligne[k])/c_max))
<ipython-input-11-a24be01002c3>:7: RuntimeWarning: invalid value encountered in
double_scalars
  C[i+1,j] = C[i,j] - dt * (Qjp1 - Q[j]) / dx  # schéma numérique
```

[13]:
```
plt.plot(C[1*p//40])
plt.title("Euler avant pour l'espace, avant pour le temps à faible concentration")
```

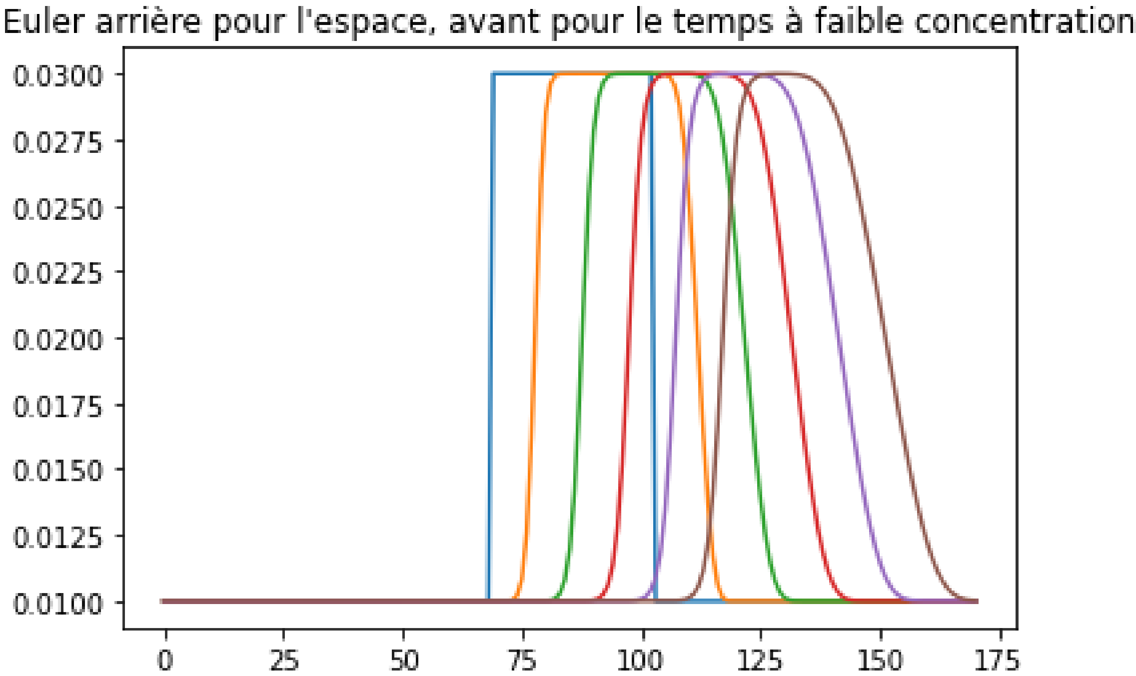
[13]:
```
Text(0.5, 1.0, "Euler avant pour l'espace, avant pour le temps à faible
concentration")
```



[14]:
```
def resolution_arriere(C, dt, dx, c_max, v_max):
    p,n = C.shape                     # nombre de valeurs en temps (p) et en espace␣
 ↪(n)
    for i in range(0,p-1):            # on itère sur tous les temps
        Q = debit(v_max,c_max,C[i])   # calcul des débits à l'instant t_i
        for j in range(n):            # on itère sur tout l'espace
            Qjm1 = Q[(j-1)%n]         # condition aux limites périodique
            C[i+1,j] = C[i,j] - dt * (Q[j] - Qjm1) / dx  # schéma numérique
        # print(min(C[i+1]))
    return C
```

[15]:
```
resolution_arriere(C, dt, dx, c_max, v_max)
for k in range(6):
    plt.plot(C[k*p//6])
```

```python
a=plt.title("Euler arrière pour l'espace, avant pour le temps à faible␣
↪concentration")
```



[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

Étude des solutions trouvées et modification du schéma

Q12.