

ÉTUDE DE TRAFIC ROUTIER

La calculatrice n'est pas autorisée.

Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre.

Ce sujet concerne la conception d'un logiciel d'étude de trafic routier. On modélise le déplacement d'un ensemble de voitures sur des files à sens unique (voir Figures 1 et 2). C'est un schéma simple qui peut permettre de comprendre l'apparition d'embouteillages et de concevoir des solutions pour fluidifier le trafic.

Le sujet comporte des questions de programmation. Le langage à utiliser est Python.

Notations : soit L une liste,

- on note $len(L)$ sa longueur ;
- pour i entier, $0 \leq i < len(L)$, l'élément de la liste d'indice i est noté $L[i]$;
- pour i et j entiers, $0 \leq i < j \leq len(L)$, $L[i : j]$ est la sous-liste composée des éléments $L[i], \dots, L[j - 1]$;
- $p * L$, avec p entier, est la liste obtenue en concaténant p copies de L . Par exemple, $3 * [0]$ est la liste $[0, 0, 0]$.

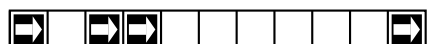


FIGURE 1 – Représentation d'une file de longueur onze comprenant quatre voitures, situées respectivement sur les cases d'indices 0, 2, 3 et 10.

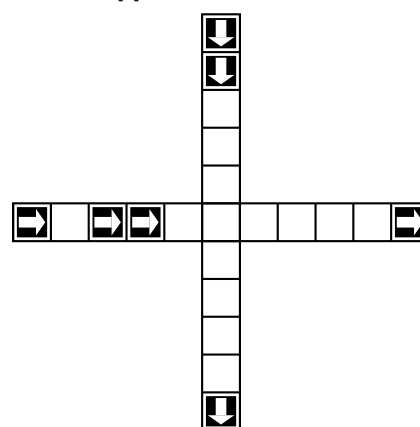


FIGURE 2 – Configuration représentant deux files de circulation à sens unique se croisant en une case. Les voitures sont représentées par un carré noir.

A. Préliminaires

Dans un premier temps, on considère le cas d'une seule file, illustré par la Figure 1. Une file de longueur n est représentée par n cases. Une case peut contenir au plus une voiture. Les voitures présentes dans une file circulent toutes dans la même direction (sens des indices croissants, désigné par les flèches sur la Figure 1) et sont indifférenciées.

- **Q1** – Expliquer comment représenter une file de voitures à l'aide d'une liste de booléens.
- **Q2** – Donner une ou plusieurs instructions Python permettant de définir une liste A représentant la file de voitures illustrée par la Figure 1.
- **Q3** – Soit L une liste représentant une file de longueur n et i un entier tel que $0 \leq i < n$. Définir en Python la fonction `occupe(L, i)` qui renvoie `True` lorsque la case d'indice i de la file est occupée par une voiture et `False` sinon.
- **Q4** – Combien existe-t-il de files différentes de longueur n ? Justifier votre réponse.
- **Q5** – Écrire une fonction `egal(L1, L2)` retournant un booléen permettant de savoir si deux listes $L1$ et $L2$ sont égales.
- **Q6** – Que peut-on dire de la complexité de cette fonction ?
- **Q7** – Préciser le type de retour de cette fonction.

B. Déplacement de voitures dans la file

On identifie désormais une file de voitures à une liste. On considère les schémas de la Figure 3 représentant des exemples de files. Une étape de simulation pour une file consiste à déplacer les voitures de la file, à tour de rôle, en commençant par la voiture la plus à droite, d'après les règles suivantes :

- une voiture se trouvant sur la case la plus à droite de la file sort de la file ;
- une voiture peut avancer d'une case vers la droite si elle arrive sur une case inoccupée ;
- une case libérée par une voiture devient inoccupée ;
- la case la plus à gauche peut devenir occupée ou non, selon le cas considéré.

On suppose avoir écrit en Python la fonction `avancer` prenant en paramètres une liste de départ, un booléen indiquant si la case la plus à gauche doit devenir occupée lors de l'étape de simulation, et renvoyant la liste obtenue par une étape de simulation.

Par exemple, l'application de cette fonction à la liste illustrée par la Figure 3(a) permet d'obtenir soit la liste illustrée par la Figure 3(b) lorsque l'on considère qu'aucune voiture nouvelle n'est introduite, soit la liste illustrée par la Figure 3(c) lorsque l'on considère qu'une voiture nouvelle est introduite.

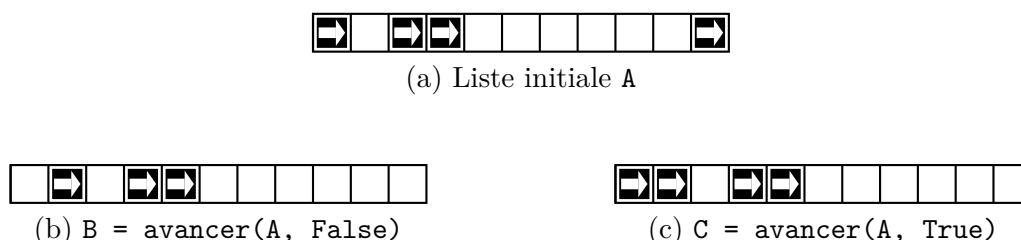
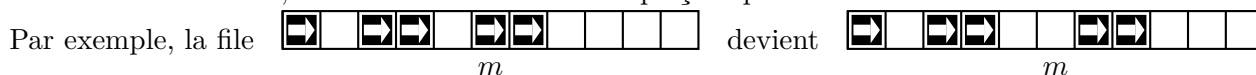


FIGURE 3 – Étape de simulation

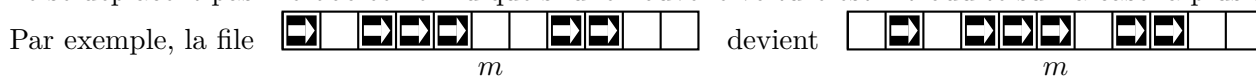
❑ **Q8** – Étant donnée A la liste définie à la question 2, que renvoie `avancer(avancer(A, False), True)` ?

❑ **Q9** – On considère L une liste et m l'indice d'une case de cette liste ($0 \leq m < len(L)$). On s'intéresse à une étape partielle où seules les voitures situées sur la case d'indice m ou à droite de cette case peuvent avancer normalement, les autres voitures ne se déplaçant pas.



Définir en Python la fonction `avancer_fin(L, m)` qui réalise cette étape partielle de déplacement et renvoie le résultat dans une nouvelle liste sans modifier L .

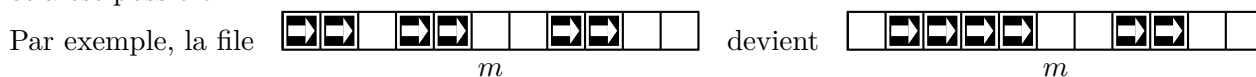
❑ **Q10** – Soient L une liste, b un booléen et m l'indice d'une case inoccupée de cette liste. On considère une étape partielle où seules les voitures situées à gauche de la case d'indice m se déplacent, les autres voitures ne se déplacent pas. Le booléen b indique si une nouvelle voiture est introduite sur la case la plus à gauche.



lorsque aucune nouvelle voiture n'est introduite.

Définir en Python la fonction `avancer_debut(L, b, m)` qui réalise cette étape partielle de déplacement et renvoie le résultat dans une nouvelle liste sans modifier L .

❑ **Q11** – On considère une liste L dont la case d'indice $m > 0$ est temporairement inaccessible et bloque l'avancée des voitures. Une voiture située immédiatement à gauche de la case d'indice m ne peut pas avancer. Les voitures situées sur les cases plus à gauche peuvent avancer, à moins d'être bloquées par une case occupée, les autres voitures ne se déplacent pas. Un booléen b indique si une nouvelle voiture est introduite lorsque cela est possible.



lorsque aucune nouvelle voiture n'est introduite.

Définir en Python la fonction `avancer_debut_bloque(L, b, m)` qui réalise cette étape partielle de déplacement et renvoie le résultat dans une nouvelle liste.

On considère dorénavant deux files $L1$ et $L2$ de même longueur impaire se croisant en leur milieu ; on note m l'indice de la case du milieu. La file $L1$ est toujours prioritaire sur la file $L2$. Les voitures ne peuvent pas quitter leur file et la case de croisement ne peut être occupée que par une seule voiture. Les voitures de la file $L2$ ne peuvent accéder au croisement que si une voiture de la file $L1$ ne s'apprête pas à y accéder. Une étape de simulation à deux files se déroule en deux temps. Dans un premier temps, on déplace toutes les voitures situées sur le croisement ou après. Dans un second temps, les voitures situées avant le croisement sont déplacées en respectant la priorité. Par exemple, partant d'une configuration donnée par la Figure 4, les configurations successives sont données par les Figures 4(b), 4(c), 4(d), 4(e) et 4(f) en considérant qu'aucune nouvelle voiture n'est introduite.

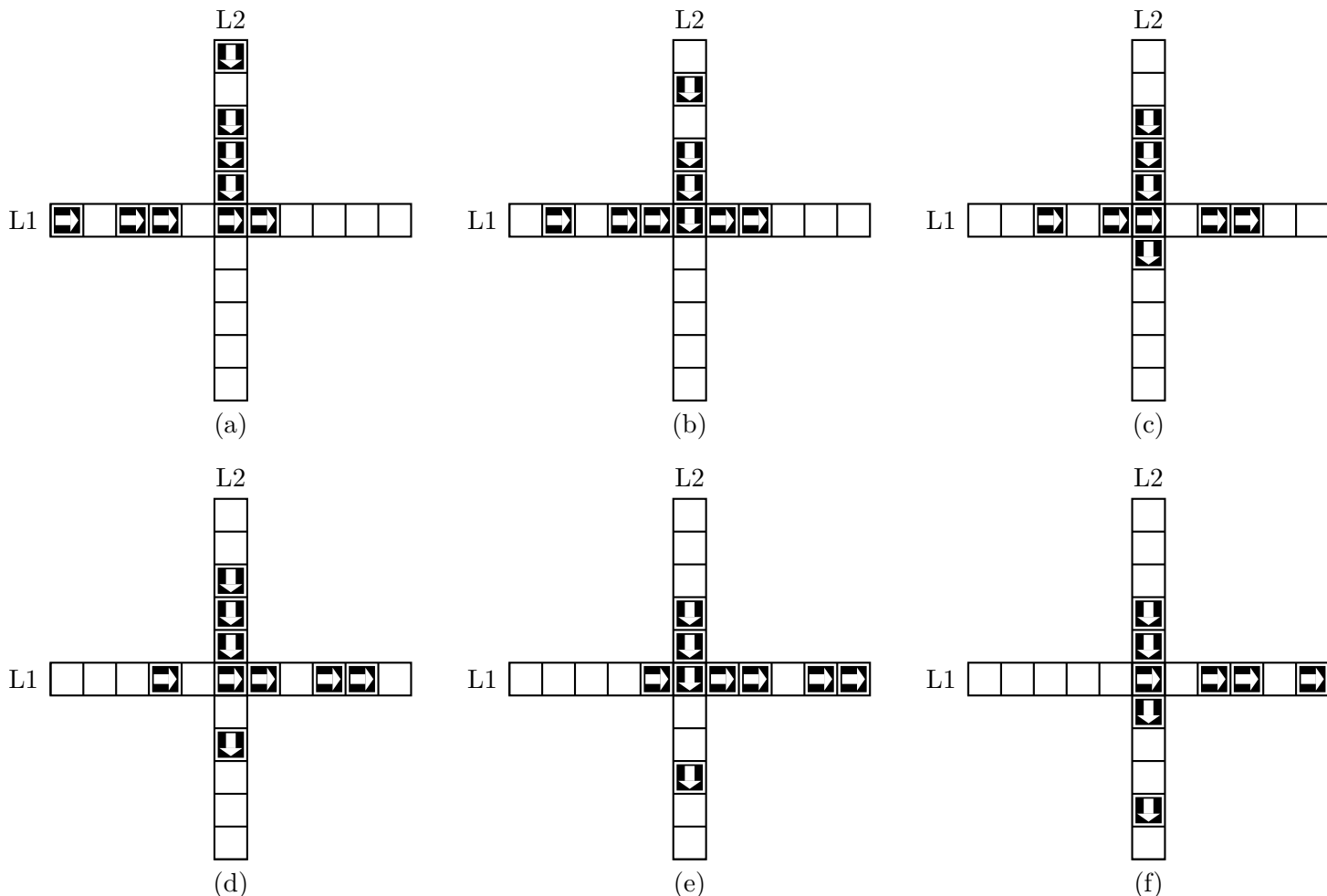


FIGURE 4 – Étapes de simulation à deux files

C. Une étape de simulation à deux files

L'objectif de cette partie est de définir en Python l'algorithme permettant d'effectuer une étape de simulation pour ce système à deux files.

❑ **Q12** – En utilisant le langage Python, définir la fonction `avancer_files(L1, b1, L2, b2)` qui renvoie le résultat d'une étape de simulation sous la forme d'une liste de deux éléments notée $[R1, R2]$ sans changer les listes $L1$ et $L2$. Les booléens $b1$ et $b2$ indiquent respectivement si une nouvelle voiture est introduite dans les files $L1$ et $L2$. Les listes $R1$ et $R2$ correspondent aux listes après déplacement.

❑ **Q13** – On considère les listes

`D = [False, True, False, True, False], E = [False, True, True, False, False]`

Que renvoie l'appel `avancer_files(D, False, E, False)` ?

D. Transitions

□ **Q14** – En considérant que de nouvelles voitures peuvent être introduites sur les premières cases des files lors d’une étape de simulation, décrire une situation où une voiture de la file *L2* serait indéfiniment bloquée.

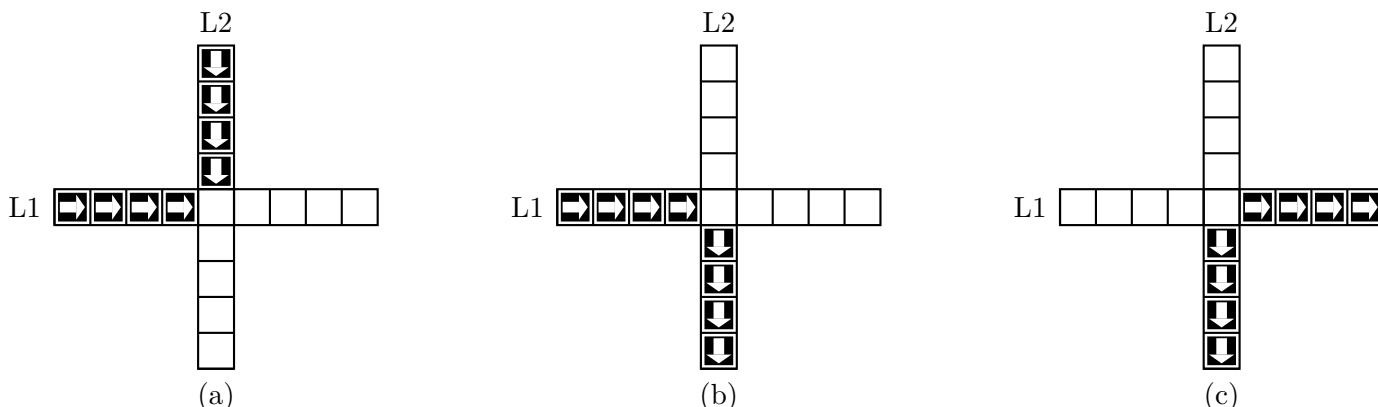


FIGURE 5 – Étude de configurations

□ **Q15** – Étant données les configurations illustrées par la Figure 5, combien d’étapes sont nécessaires (on demande le nombre minimum) pour passer de la configuration 5(a) à la configuration 5(b)? Justifier votre réponse.

□ **Q16** – Peut-on passer de la configuration 5(a) à la configuration 5(c)? Justifier votre réponse.

E. Atteignabilité

Certaines configurations peuvent être néfastes pour la fluidité du trafic. Une fois ces configurations identifiées, il est intéressant de savoir si elles peuvent apparaître. Lorsque c’est le cas, on dit qu’une telle configuration est atteignable. Pour savoir si une configuration est atteignable à partir d’une configuration initiale, on a écrit le code incomplet donné en annexe. Le langage Python sait comparer deux listes de booléens à l’aide de l’opérateur usuel `<`, on peut ainsi utiliser la méthode `sort` pour trier une liste de listes de booléens.

□ **Q17** – Ecrire en langage Python une fonction `elim_double(L)` non récursive, de complexité linéaire en la taille de `L`, qui élimine les éléments apparaissant plusieurs fois dans une liste triée `L` et renvoie la liste triée obtenue. Par exemple `elim_double([1, 1, 3, 3, 3, 7])` doit renvoyer la liste `[1, 3, 7]`.

On dispose de la fonction suivante :

```

1 def doublons(liste):
2     if len(liste) > 1:
3         if liste[0] != liste[1]:
4             return [liste[0]] + doublons(liste[1:])
5         liste1 = [liste[0]] + liste[2:]
6         return doublons(liste1)
7     else:
8         return liste
    
```

□ **Q18** – Que retourne l’appel suivant ? `doublons([1, 1, 2, 2, 3, 3, 3, 5])`

□ **Q19** – Cette fonction est-elle utilisable pour éliminer les éléments apparaissant plusieurs fois dans une liste non triée? Justifier.

□ **Q20** – La fonction recherche donnée en annexe permet d’établir si la configuration correspondant à `but` est atteignable en partant de l’état `init`. Préciser le type de retour de la fonction `recherche`, le type des variables `but` et `espace`, ainsi que le type de retour de la fonction `successeurs`.

□ **Q21** – Afin d'améliorer l'efficacité du test `if but in espace`, ligne 10 de l'annexe, on propose de le remplacer par `if in1(but, espace)` ou bien par `if in2(but, espace)`, avec `in1` et `in2` deux fonctions définies ci-dessous. On considère que le paramètre `liste` est une liste triée par ordre croissant.

Quel est le meilleur choix ? Justifier.

```

1 def in1(element, liste):
2     a = 0
3     b = len(liste) - 1
4     while a <= b and element >= liste[a]:
5         if element == liste[a]:
6             return True
7         else:
8             a = a + 1
9     return False

12 def in2(element, liste):
13     a = 0
14     b = len(liste)-1
15     while a < b:
16         pivot = (a + b) // 2 # l'opérateur // est la division entière
17         if liste[pivot] < element:
18             a = pivot + 1
19         else:
20             b = pivot
21     if element == liste[a]:
22         return True
23     else:
24         return False

```

□ **Q22** – Afin de comparer plus efficacement les files représentées par des listes de booléens on remarque que ces listes représentent un codage binaire où `True` correspond à 1 et `False` à 0.

Écrire la fonction `versEntier(L)` prenant une liste de booléens en paramètre et renvoyant l'entier correspondant. Par exemple, l'appel `versEntier([True, False, False])` renverra 4.

□ **Q23** – On veut écrire la fonction inverse de `versEntier`, transformant un entier en une liste de booléens. Que doit être au minimum la valeur de taille pour que le codage obtenu soit satisfaisant ? On suppose que la valeur de taille est suffisante. Quelle condition booléenne faut-il écrire en ligne 4 du code ci-dessous ?

```

1 def versFile(n, taille):
2     res = taille * [False]
3     i = taille - 1
4     while ...:
5         if (n % 2) != 0: # % est le reste de la division entière
6             res[i] = True
7         n = n // 2 # // est la division entière
8         i = i - 1
9     return res

```

□ **Q24** – Montrer qu'un appel à la fonction `recherche` de l'annexe se termine toujours.

□ **Q25** – Compléter la fonction `recherche` pour qu'elle indique le nombre minimum d'étapes à faire pour passer de `init` à `but` lorsque cela est possible. Justifier la réponse.

F. Base de données

On modélise ici un réseau routier par un ensemble de croisements et de voies reliant ces croisements. Les voies partent d'un croisement et arrivent à un autre croisement. Ainsi, pour modéliser une route à double sens, on utilise deux voies circulant en sens opposés. La base de données du réseau routier est constituée des relations suivantes :

- Croisement(id, longitude, latitude)
- Voie(id, longueur, id_croisement_debut, id_croisement_fin)

Dans la suite on considère c l'identifiant (id) d'un croisement donné.

□ **Q26** – Écrire la requête SQL qui renvoie les identifiants des croisements atteignables en utilisant une seule voie à partir du croisement ayant l'identifiant c .

□ **Q27** – Écrire la requête SQL qui renvoie les longitudes et latitudes des croisements atteignables en utilisant une seule voie, à partir du croisement c .

□ **Q28** – Que renvoie la requête SQL suivante ?

```

1 SELECT V2.id_croisement_fin
2 FROM Voie as V1
3 JOIN Voie as V2
4 ON V1.id_croisement_fin = V2.id_croisement_debut
5 WHERE V1.id_croisement_debut = c

```

G. Simulation dynamique

On introduit les bibliothèques Math et NumPy à l'aide des lignes suivantes :

```

1 import math as m
2 import numpy as np

```

On s'intéresse maintenant à l'apparition des voitures en début de file. On se place alors dans le cas d'un parking lors d'une sortie d'usine entre 17 h et 19 h. Afin de simplifier le problème on considère qu'à $t = 0$, il est 17 h et à $t = 1$, il est 19 h.

Tous les employés de l'usine ne sortent pas au même moment. On donne sur la Figure 6 l'évolution de $Q(t)$ qui représente le nombre de véhicules quittant le stationnement par unité de temps, pour $t \in [0,1]$. On considère que cette fonction est nulle en dehors de cet intervalle. On appelle db_stat_max le maximum de Q entre 0 et 1.

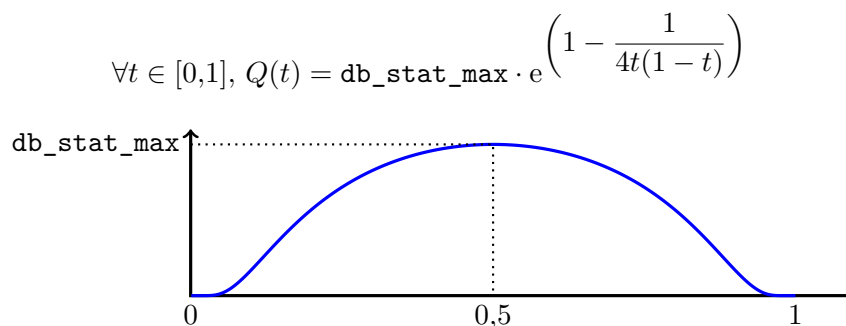


FIGURE 6 – évolution du nombre de véhicules quittant le stationnement par unité de temps

□ **Q29** – Écrire une fonction $Q(t)$ permettant d'obtenir, pour tout réel t , le nombre de véhicules quittant le stationnement par unité de temps. On considère db_stat_max comme une variable globale. Faire attention aux cas où $t \leq 0$ et $t \geq 1$.

□ **Q30** – Écrire une fonction `integrale(f, a, b, n)` permettant, avec la méthode des trapèzes, d'estimer l'intégrale de f entre a et b à partir de $n + 1$ points équirépartis (soit n sous-segments).

□ **Q31** – On appelle $N(t)$ le nombre total de véhicules ayant quitté le stationnement à l'instant t . Écrire une fonction `Nb(t, n)` renvoyant une valeur approchée de ce nombre, en utilisant $n + 1$ points.

La suite de cette partie permettant d'obtenir $N(t)$ par une autre méthode, la fonction `Nb` ne sera plus utilisée.

La vitesse de sortie des véhicules est limitée par le nombre de véhicules sortants. En effet, quand peu de véhicules sortent du parking, ces derniers peuvent circuler à vitesse maximale V_{max} . En revanche, lorsque beaucoup de véhicules sont en mouvement, ces derniers se ralentissent les uns les autres. On appelle K la concentration de véhicules cherchant à rejoindre la sortie du parking à un instant donné et K_{sat} le nombre de véhicules à partir duquel la vitesse de sortie est minimale (V_{min}).

L'expression de la vitesse des véhicules V en fonction de $K \in [0, K_{sat}]$ est donnée par

$$V(K) = \frac{V_{max} - V_{min}}{2} \cdot \left(1 + \cos \left(\pi \cdot \frac{K}{K_{sat}} \right) \right) + V_{min}.$$

□ **Q32** – Écrire une fonction `V(K)` prenant en entrée un flottant K et renvoyant la valeur de la vitesse correspondante. On considère `Vmin`, `Vmax` et `Ksat` comme des variables globales.

La conservation du nombre total de véhicules conduit au système différentiel suivant :

$$\frac{dN}{dt}(t) = Q(t) \tag{1}$$

$$\frac{dK}{dt}(t) = Q(t) - K(t) \cdot V(K) \tag{2}$$

$$\frac{dS}{dt}(t) = K(t) \cdot V(K) \tag{3}$$

où $S(t)$ désigne le nombre de véhicules sortis du parking à l'instant t .

□ **Q33** – Rappeler la relation de récurrence de la méthode d'Euler explicite pour un problème défini par son équation différentielle $y'(t) = F(y(t), t)$ avec $h = \Delta t$ le pas de temps supposé constant.

On considère que cette méthode est implémentée dans une fonction `odeint(F, Y0, T)` où F est la fonction associée au problème de Cauchy, `Y0` est un tableau NumPy contenant les valeurs initiales de la fonction Y et `T` un tableau NumPy contenant les différents pas de temps.

On cherche à vectorialiser les équations (1) à (3) et on pose

$$Y(t) = \begin{pmatrix} N(t) \\ K(t) \\ S(t) \end{pmatrix}.$$

On rappelle qu'une liste `L` peut être convertie en tableau NumPy en utilisant `T = np.array(L)`.

□ **Q34** – Écrire une fonction `F(Y, t)` permettant de simuler les équations (1) à (3) à partir de la fonction `odeint` (mais on ne demande pas d'écrire le code utilisant la fonction `odeint`).

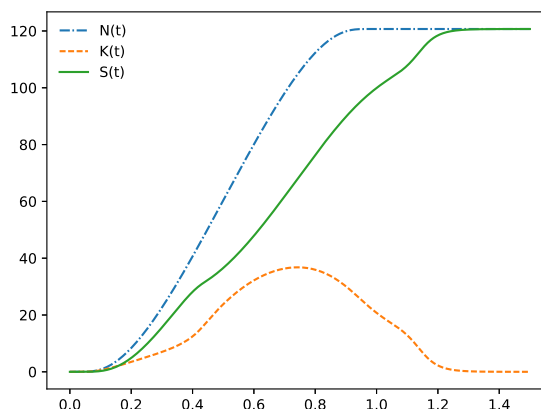


FIGURE 7 – Simulation de la sortie des véhicules du parking d'une usine

La Figure 7 donne une simulation de la sortie des véhicules du parking d'une usine avec `db_stat_max = 200`, `Vmin = 4`, `Vmax = 24` et `Ksat = 18`.

□ **Q35** – Commenter en quelques lignes les allures des courbes de la Figure 7.

ANNEXE

```
1 def recherche(but, init):
2     espace = [init]
3     stop = False
4     while not stop:
5         ancien = espace
6         espace = espace + successeurs(espace)
7         espace.sort() # permet de trier espace par ordre croissant
8         espace = elim_double(espace)
9         stop = egal(ancien, espace) # fonction définie à la question 5
10        if but in espace:
11            return True
12    return False

15 def successeurs(L):
16     res = []
17     for x in L:
18         L1 = x[0]
19         L2 = x[1]
20         res.append( avancer_files(L1, False, L2, False) )
21         res.append( avancer_files(L1, False, L2, True) )
22         res.append( avancer_files(L1, True, L2, False) )
23         res.append( avancer_files(L1, True, L2, True) )
24    return res

26 # dans une liste triée, elim_double enlève les éléments apparaissant plus
27 # d'une fois
28 # exemple : elim_double([1, 1, 2, 3, 3]) renvoie [1, 2, 3]
29 def elim_double(L):
30     # code à compléter

31 # exemple d'utilisation
32 # debut et fin sont des listes composées de deux files de même longueur
33 # impaire,
34 # la première étant prioritaire par rapport à la seconde
35 debut = [5 * [False], 5 * [False]]
36 fin = [3 * [False] + 2 * [True], 3 * [False] + 2 * [True]]
print(recherche(fin, debut))
```

FIN DE L'ÉPREUVE