DS INFORMATIQUE

Dans tout le devoir, on suppose que les bibliothèques math, numpy et matplotlib.pyplot ont été importées de la manière suivante :

```
from math import *
import numpy as np
import matplotlib.pyplot as plt
```

Problème 1 : Modélisation du mouvement de plateforme en mer

(d'après CCINP 2019 PC)

On s'intéresse à la résolution d'équation du mouvement d'une plateforme en mer. Le modèle envisagé est un système à un degré de liberté considéré comme oscillateur harmonique : une masse est reliée à un ressort, avec ou sans amortissement, et peut être soumise à une excitation externe.
Dans la suite de l'énoncé, toutes les grandeurs vectorielles sont indiquées en gras. Un aide-mémoire sur numpy *est donné en Annexe.*

On considère le mouvement d'une plateforme en mer soumise à un courant marin. Sa partie supérieure de masse m = 110 tonnes est considérée comme rigide et le mouvement principal de la plateforme a lieu suivant x (**figure 1(a)**). Afin d'étudier le mouvement de cette plateforme, on la modélise par une masse m, liée à un ressort de constante de raideur k et à un amortisseur de constante d'amortissement γ, pouvant subir une excitation externe de force $\mathbf{F}_{exc}$, et se déplaçant sur un support (**figure 1(b)**). Le ressort représente la rigidité de l'ensemble du support de la plateforme. L'amortisseur permet de prendre en compte l'effet de l'eau environnante et la force d'excitation externe celui des vagues qui frappent périodiquement la plateforme. La masse est supposée se déplacer selon une seule direction parallèle à l'axe Ox en fonction du temps t.

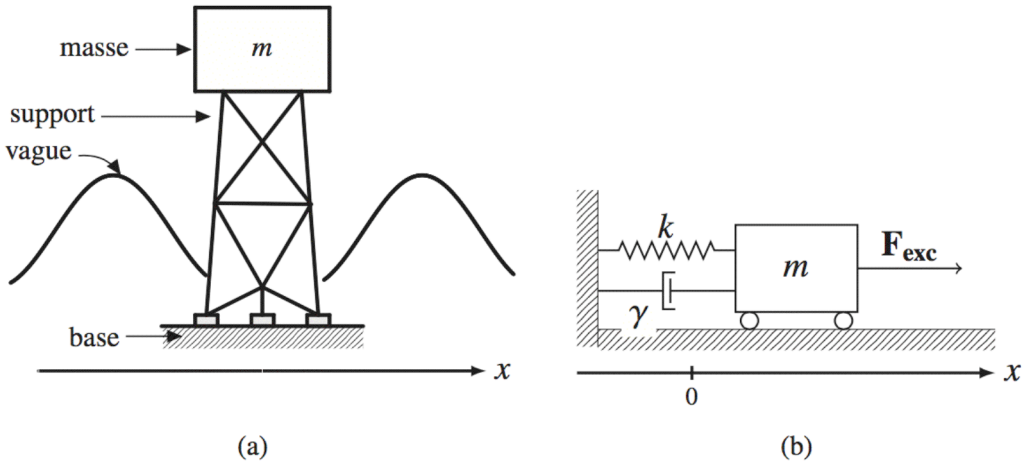

Figure 1 – (a) Plateforme en mer soumise aux vagues marines, (b) système masse (m), ressort (k), amortisseur (γ) et excitation externe ($\mathbf{F}_{exc}$)

La force totale $\mathbf{F}_{tot}$ agissant sur la masse correspond à la réaction normale $\mathbf{R}_N$ de la base horizontale, à la force de frottement $\mathbf{F}_d = -\gamma\,\mathbf{v}$ (γ constante d'amortissement et $\mathbf{v}$ vecteur vitesse), à la force de rappel $\mathbf{F}_k$ du ressort, au poids $\mathbf{P}$ de la masse et à la force $\mathbf{F}_{exc}$ d'excitation externe supposée sinusoïdale, avec $\mathbf{F}_{exc}(t) = F_0 \cos(\omega t)\,\mathbf{e}_x$, où $\mathbf{e}_x$ représente le vecteur unitaire sur l'axe Ox. La position d'équilibre de la masse sera choisie à $x = 0$.

En effectuant une projection sur l'axe Ox de l'équation du mouvement de la masse, l'équation différentielle caractérisant le système vaut :

$$\ddot{x} + 2\zeta\omega_0\dot{x} + \omega_0^2 x = \frac{F_0}{m}\cos(\omega t) \qquad (11)$$

où ζ et ω_0 (pulsation propre en absence de frottements et force d'excitation) dépendent de k, m et γ.

Partie II - Modélisation : codage

On souhaite obtenir $x(t)$ et $E(t)$ de façon numérique, où $x(t)$ et $E(t)$ représentent respectivement la position de la masse et l'énergie mécanique totale en fonction du temps.

Pour cela, le temps est discrétisé en N points $t = 0$, Δt, $2\Delta t$, ..., $(N–1)\ \Delta t$ avec un pas de temps constant Δt. Les $N–1$ pas sont effectués pendant la simulation de durée totale t_{max}.

On note respectivement x_n, v_n, a_n, E_n et F_n les valeurs de $x(t)$, $\dot{x}(t)$, $\ddot{x}(t)$, $E(t)$ et $F_{exc}(t)$ à $t = n\Delta t$.

À chaque pas, les équations du mouvement reliant x_{n+1} et v_{n+1} à x_n et v_n sont utilisées afin d'obtenir les valeurs de x, v et E. Les conditions initiales x_0 et v_0 sont connues et permettent de démarrer le processus d'intégration numérique. Deux algorithmes distincts (Euler et Leapfrog) vont être utilisés dans la suite. Pour l'écriture du code, on se place dans le cas le plus général, c'est-à-dire avec amortissement et excitation harmonique externe. Les variables et tableaux suivants sont notamment choisis :

N	nombre de points sur l'axe des temps utilisés pendant toute la simulation
T[]	tableau des temps (s), de dimension N
X[]	tableau des positions (m), de dimension N
V[]	tableau des vitesses ($m.s^{-1}$), de dimension N
E[]	tableau des énergies totales (J), de dimension N
F[]	tableau des forces d'excitation (N), de dimension N
dt	pas de temps (s)
tmax	temps total de la simulation (s)
k	constante de raideur du ressort (67 900 $N.m^{-1}$)
m	masse du système (110 000 kg)
om0	pulsation propre ω_0 (0,786 rad.s^{-1})
zeta	ζ = 0,0501 (sans unité)

Tableau 1 – Principaux tableaux et principales variables utilisés pour la résolution numérique

Q1. On suppose t_{max} et Δt connus et fixés par l'utilisateur en début de code (100 s et 0,01 s par exemple). Écrire les lignes de code permettant de définir l'entier N.

Q2. Écrire alors l'instruction permettant de définir le tableau T qui contient toutes les valeurs de t, discrétisé en N points avec un pas de temps constant Δt, tel que : $0 \leq t \leq t_{max}$.

Q3. Écrire une fonction force(f0,t,w) qui prend en argument F_0, t, et ω de l'excitation externe supposée sinusoïdale, et retourne la valeur de $F_{exc}(t) = F_0 \cos(\omega t)$ pour un temps t donné.

Q4. Écrire une fonction force_exc() qui complète et retourne le tableau F[] des forces d'excitation en fonction du booléen exc défini globalement, valant True si une excitation est appliquée au système, False sinon. Dans le cas où exc vaut True, on appellera la fonction force définie précédemment en question Q3. Dans le cas où exc vaut False, on prendra alors : F[i]=0, $\forall i$. On précisera les variables globales éventuellement introduites en supposant qu'elles ont été définies dans le code.

Q5. En effectuant des développements de Taylor de x et v tronqués à l'ordre 1, on obtient l'algorithme d'Euler, où x et v sont évalués au même temps t selon le schéma donné figure 3. Donner les expressions de x_{n+1} et v_{n+1} en fonction de x_n, v_n et F_n.

$$\begin{cases} x_{n+1} \approx x_n + v_n \cdot \Delta t \\ v_{n+1} \approx v_n + a_n \cdot \Delta t \end{cases}$$

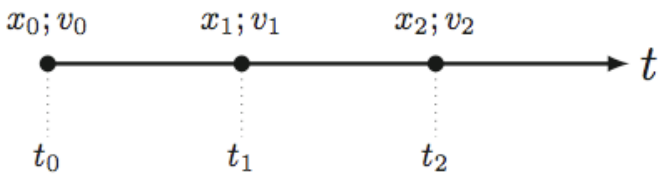

Figure 3 – Discrétisation en temps utilisée dans le cas de l'algorithme d'Euler. Les conditions initiales correspondent à (x_0; v_0)

Q6. Écrire une fonction euler(F,x0,v0) qui prend en arguments le tableau F[] des forces d'excitation, l'abscisse initiale x0 et la vitesse initiale v0, et qui renvoie les tableaux X, V et E contenant toutes les valeurs de x[i], v[i] et E[i] où i correspond à un point sur l'axe des temps. On précisera les variables globales éventuellement introduites en supposant qu'elles ont été définies dans le code.

Q7. Dans l'algorithme de Leapfrog, les x sont évalués aux temps entiers, c'est-à-dire à t=0, Δt, $2\Delta t$,..., $(N-1)\Delta t$, alors que les v sont évalués à $t = -\Delta t/2$, $\Delta t/2$, $3\Delta t/2$, . . . , $(N-1)\Delta t - \Delta t/2$ selon le schéma donné figure 4. Ainsi, pour cet algorithme, x[i] représente de façon approchée la position à l'instant $i\Delta t$, et v[i] représente de façon approchée la vitesse à l'instant $(i-1/2)\Delta t$.

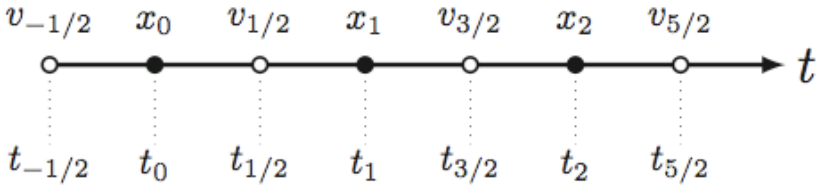

Figure 4 – Discrétisation en temps utilisée dans le cas de l'algorithme de Leapfrog.
Les conditions initiales correspondent à (x_0; $v_{-1/2}$), x_{n+1} est obtenu à partir de x_n et $v_{n+1/2}$, tandis que $v_{n+1/2}$ est obtenu à partir de $v_{n-1/2}$ et a_n.

Pour le système considéré, montrer alors que x_{n+1} et $v_{n+1/2}$ prennent les formes suivantes :

$$\begin{cases} x_{n+1} \approx x_n + v_{n+1/2} \cdot \Delta t \\ v_{n+1/2} \approx -\dfrac{\omega_0^2 \Delta t}{1+\zeta\omega_0\Delta t} x_n + \dfrac{1-\zeta\omega_0\Delta t}{1+\zeta\omega_0\Delta t} v_{n-1/2} + \dfrac{F_n}{m(1+\zeta\omega_0\Delta t)}\Delta t \end{cases}$$

Q8. Comment obtenir E[i] à partir de x[i] et v[i] ? Quel problème pourrait poser l'évaluation de E[i+1] ? Dans la suite, on préfèrera ainsi évaluer E[i].

Q9. En introduisant deux variables locales fac1 et fac2 correspondant respectivement à $1-\zeta\omega_0\Delta t$ et $1/(1+\zeta\omega_0\Delta t)$, écrire une fonction leapfrog(F,x0,v0) qui prend en arguments le tableau F[] des forces d'excitation, l'abscisse initiale x0 et la vitesse initiale $v_{-1/2}$, et qui renvoie les tableaux X, V et E contenant toutes les valeurs de x[i], v[i] et E[i]. On précisera les variables globales éventuellement introduites en supposant qu'elles ont été définies dans le code.

Q10. On cherche à effectuer une simulation numérique en utilisant l'algorithme de Leapfrog dans le cas du système amorti soumis à une force excitatrice $F_{exc}(t) = F_0 \cos(\omega t)$ pour laquelle F_0 = 10 000 N et ω = 0,5 rad.s^{-1}. Donner les lignes de code permettant de réaliser cette simulation numérique à partir des fonctions précédentes puis d'en représenter graphiquement le portrait de phase (vitesse en fonction de la position). L'affectation des variables globales correspondant aux paramètres physiques et/ou numériques du système est également attendue dans ce code.

Q11. On souhaite désormais estimer la qualité des résultats numériques par rapport aux données analytiques de référence. Écrire une fonction ema(d,dref) qui calcule l'erreur maximale absolue entre un jeu de données numériques *dn* et un jeu de données analytiques *da*. On supposera que les tableaux *dn* et *da* sont de même dimension *n*.

Partie III - Modélisation : analyse des résultats d'un cas simple

Toutes les données numériques suivantes ont été obtenues avec : t_{max} = 10 s, m = 110 tonnes, x_0 = 0,02 m et v_0 = 0 m.s^{-1} et la valeur de k précisée plus haut, dans le cas d'un système sans amortissement et sans excitation externe. Le tableau 2 présente les erreurs maximales absolues (EMA) calculées avec la fonction ema des énergies obtenues numériquement par rapport à celles obtenues analytiquement, pour les deux algorithmes envisagés et divers pas de temps.

Q12. Justifier l'ordre de grandeur des Δt considérés du tableau 2 pour la discrétisation en temps.

Q13. En utilisant les données numériques du tableau 2 donner l'ordre approximatif de l'erreur globale sur E des deux algorithmes considérés. Justifier votre réponse.

Q14. Dans le cas simple de l'algorithme d'Euler, comment augmente E lorsqu'on passe de t_n à t_{n+1} ? Relier alors ce résultat aux données du tableau 2.

Δt (s)	EMA	
	Euler	Leapfrog
0,050	128,0203160	0,0837314
0,010	15,1373529	0,0033484
0,005	7,1159053	0,0008371
0,001	1,3556230	0,0000335

Tableau 2 – EMA obtenues pour E (en J) par rapport à la valeur analytique, pour différents pas de temps Δt, dans le cas où il n'y a pas d'amortissement et d'excitation externe

Problème 2 : Intelligence Artificielle - Application en médecine

(CCINP 2019 PSI, extraits)

Les techniques d'apprentissage machine, qui permettent à un logiciel d'apprendre automatiquement à partir de données au lieu d'être explicitement programmé, ont de nombreuses applications dans la reconnaissance d'image, la compréhension de la parole ou de textes, dans l'assistance à la décision, dans la classification de données, dans la robotique... L'Intelligence Artificielle progresse également dans le domaine de la santé. Le logiciel Watson développé par IBM peut analyser les données d'un patient : ses symptômes, ses consultations médicales, ses antécédents familiaux, ses données comportementales, ses résultats d'examen, etc. Il établit alors une prévision de diagnostic le plus vraisemblable et propose des options de traitement en s'appuyant sur une base de données établie sur un grand nombre de patients.

Objectif

L'objectif du travail proposé est de découvrir quelques techniques d'Intelligence Artificielle en s'appuyant sur un problème médical. À partir d'une base de données comportant des propriétés sur le bassin et le rachis lombaire, on cherche à déterminer si un patient peut être considéré comme « normal » ou bien si ces données impliquent un développement anormal de type hernie discale (saillie d'un disque intervétébral) ou spondylolisthésis (glissement du corps vertébral par rapport à la vertèbre sous-jacente). Le sujet abordera les points suivants :

- analyse et représentation des données,
- prédiction à l'aide de la méthode KNN.

Partie II - Analyse des données

La base de données contient des informations administratives sur les patients et des informations médicales. Pour simplifier le problème, on considère deux tables : PATIENT et MEDICAL.

La table PATIENT contient les attributs suivants :

- id : identifiant d'un individu (entier), clé primaire ;
- nom : nom du patient (chaîne de caractères) ;
- prenom : prénom du patient (chaîne de caractères) ;
- adresse : adresse du patient (chaîne de caractères) ;
- email : (chaîne de caractères) ;
- naissance : année de naissance (entier).

La table MEDICAL contient les attributs suivants :

- id : identifiant d'un ensemble de propriétés médicales (entier), clé primaire ;
- data1 : donnée (flottant) ;
- data2 : donnée (flottant) ;
- ... ;
- idpatient : identifiant du patient représenté par l'attribut id de la table PATIENT (entier) ;
- etat : description de l'état du patient (chaîne de caractères).

Les attributs data1, data2... sont des données relatives à l'analyse médicale souhaitée (dans notre cas des données biomécaniques). L'attribut « etat » permet d'affecter un label à un ensemble de données médicales : « normal », « hernie discale », « spondylolisthésis ».

Q1. Écrire une requête SQL permettant d'extraire les identifiants des patients ayant une « hernie discale ».
Q2. Écrire une requête SQL permettant d'extraire les noms et prénoms des patients atteints de « spondylolisthésis ».
Q3. Écrire une requête SQL permettant d'extraire chaque état et le nombre de patients pour chaque état.

Une telle base de données permet donc d'essayer de trouver des liens entre les données des patients et une maladie grâce à des algorithmes d'Intelligence Artificielle. Pour l'étude qui va suivre, on extrait de la base de données les attributs biomécaniques de chaque patient que l'on stocke dans un tableau de réels (codés sur 32 bits) à N lignes (nombre de patients) et n colonnes (nombre d'attributs égal à 6 dans notre exemple). On nomme data ce tableau. L'état de santé est stocké dans un vecteur de taille N contenant des valeurs entières (codées sur 8 bits) correspondant aux différents états pris en compte (0 : normal, 1 : hernie discale, 2 : spondylolisthésis). On le nomme : etat. Les variables data et etat sont stockées dans des objets de type array de la bibliothèque Numpy.
Le tableau suivant montre les premières valeurs du tableau data.

incidence_bassin	orientation_bassin	angle_lordose	pente_sacrum	rayon_bassin	glissement_spon
63,03	22,55	39,61	40,48	98,67	– 0,25
39,06	10,06	25,02	29,0	114,41	4,56
68,83	22,22	50,09	46,61	105,99	– 3,53
69,3	24,65	44,31	44,64	101,87	11,21
49,71	9,65	28,32	40,06	108,17	7,92
40,25	13,92	25,12	26,33	130,33	2,23
48,26	16,42	36,33	31,84	94,88	28,34

Tableau 1 – Données (partielles) des patients à diagnostiquer

Les 6 attributs considérés dans notre exemple (les n colonnes du tableau) sont donc :

- angle d'incidence du bassin en ° ;
- angle d'orientation du bassin en ° ;
- angle de lordose lombaire en ° ;
- pente du sacrum en ° ;
- rayon du bassin en mm ;
- distance algébrique de glissement de spondylolisthésis en mm.

Q6. Écrire une fonction separationParGroupe(data,etat) qui sépare le tableau data en 3 sous-tableaux en fonction des valeurs du vecteur etat correspondantes (on rappelle que celui-ci contient les valeurs 0, 1 et 2 uniquement). La fonction doit renvoyer une liste de taille 3 de sous-tableaux. Les sous-tableaux ne seront pas nécessairement représentés par un type 'array' ; 'liste de listes', 'liste d'array' conviennent également.

Partie III - Apprentissage et prédiction par la méthode KNN

La méthode KNN est une méthode d'apprentissage dite supervisée ; des données sont classées par groupes clairement identifiés et on cherche dans quels groupes appartiennent de nouvelles données. Le principe de la méthode est simple. Après avoir calculé la distance euclidienne entre toutes les données connues des patients et les données d'un nouveau patient à classer, on extrait les K données connues les plus proches. L'appartenance du nouveau patient à un groupe est obtenue en cherchant le groupe majoritaire, c'est-à-dire, le groupe qui apparaît être le plus représentatif parmi les K données connues.
On note X_i , avec i entier compris entre 0 et n–1 inclus, le vecteur colonne i du tableau de données data, c'est-à-dire les valeurs prises par l'attribut i des données des patients déjà classées. On cherche à déterminer à quel groupe appartient un nouveau patient dont les attributs sont représentés par un n-uplet *z* de valeurs notées (z_0, z_1, ..., z_{n-1}).

Préparation des données

Avant de calculer la distance euclidienne, il est préférable de normaliser les attributs pour éviter qu'un attribut ait plus de poids par rapport à un autre. On choisit de ramener toutes les valeurs des attributs entre 0 et 1. Une technique de normalisation consiste à rechercher pour chaque attribut X le minimum (min(X)) qui est placé à 0 et le maximum (max(X)) qui est placé à 1. On note alors Xnorm un des vecteurs colonne X après normalisation (toutes les valeurs de X_{norm} sont donc comprises entre 0 et 1).

Q9. Proposer une expression de $x_{norm\ k}$ un élément du vecteur X_{norm} en fonction de l'élément x_k du vecteur X correspondant et de min(X) et max(X).

Q10. Écrire une fonction min_max(X) qui retourne les valeurs du minimum et du maximum d'un vecteur X passé en argument. La fonction devra être de complexité linéaire.

Q11. Écrire une fonction distance(*z*,data) qui parcourt les N lignes du tableau data et calcule les distances euclidiennes entre le n-uplet *z* et chaque n-uplet *x* du tableau de données connues (x représente une ligne du tableau). La fonction doit renvoyer une liste de taille N contenant les distances entre chaque n-uplet x et le n-uplet z.

Détermination des K plus proches voisins

Pour déterminer les K plus proches voisins avec K un entier choisi arbitrairement, il suffit d'utiliser un algorithme de tri efficace. La liste T à trier est une liste de listes à 2 éléments contenant :

- la distance entre le n-uplet à classer et un n-uplet connu (on trie par ordre croissant sur ces valeurs) ;
- la valeur de l'état correspondant au n-uplet connu.

On retient l'algorithme suivant :

```
1    def tri ( T ) :
2        if len ( T ) <= 1 :
3            return T
4        else :
5            m = len ( T ) / / 2
6            tmp1 = T[0:m]
7            tmp2 = T[m:len(T)]
8            return fct (tri (tmp1) , tri (tmp2) )
9
```

```
10  def fct (T1 , T2) :
11      n1, n2 = len(T1), len(T2)
12      T=[]
13      i , k = 0 , 0
14      while  i < n1 and  k < n2 :
15          if T1[i] [0] < T2[k] [0] :
16              T.append (T1[i])
17              i = i +1
18          else :
19  . . . . . # l i g n e 1 à compléter
20  . . . . . # l i g n e 2 à compléter
21      if  i = = n1 :
22          for j in range (k , n2) :
23              T.append (T2[j])
24      else :
25  . . . . . #  ligne 3  à completer (1 ou 2 lignes)
26      return T
```

Q12. Donner le nom du tri proposé. Écrire une fonction tri_insertion(T) qui permet de trier en place la liste de listes T comme la fonction tri proposée, mais en utilisant le tri par insertion. Quel est l'intérêt de la fonction tri proposée par rapport à tri_insertion ?

Q13. Compléter les lignes 1 à 3 de la fonction fct pour que l'algorithme de tri soit fonctionnel.

L'algorithme de la méthode KNN est décrit par la fonction python suivante :

```
1  def KNN(data,etat,z,K,nb):
2      #partie 1
3      T = []
4      dist = distance(z,data)
5      for i in range(len(dist)):
6          T.append([dist[i],i])
7      tri(T)
8
9      #partie 2
10     select = [0]*nb
11     for i in range(K):
12         select[etat[T[i][1]]]+=1
13
14     #partie 3
15     ind = 0
16     res = select[0]
17     for k in range(1,nb):
18         if select[k] > res:
19             res = select[k]
20             ind = k
21     return ind
```

K représente le nombre de voisins proches retenus, nb correspond au nombre d'état (3 dans notre exemple) et z correspond aux attributs du patient à classer.

Q14. Expliquer ce que font globalement les parties 1 (lignes 3 à 7), 2 (lignes 10 à 12) et 3 (lignes 15 à 21) de l'algorithme. Préciser ce que représentent les variables locales T, dist, select, ind.

Validation de l'algorithme

Pour tester l'algorithme, on utilise un jeu de données supplémentaires normalisées dont l'état des patients est connu (100 patients). On note *datatest* ces données et *etattest* le vecteur d'état connu pour ces patients. On applique ensuite l'algorithme sur chaque élément de ce jeu de données pour une valeur de K fixée. On définit la fonction suivante qui renvoie une matrice appelée "matrice de confusion".

```
1  def test_KNN(datatest,etattest,data,etat,K,nb):
2      etatpredit = []
3      for i in range(len(datatest)):
4          res = KNN(data,etat,datatest[i],K,nb)
5          etatpredit.append(res)
6
7      mat = np.zeros((nb,nb))
8      for i in range(len(etattest)):
9          mat[etattest[i],etatpredit[i]] += 1
10     return mat
```

On obtient pour K = 8 la matrice suivante :

$$\begin{pmatrix} 23 & 4 & 7 \\ 7 & 11 & 1 \\ 5 & 2 & 40 \end{pmatrix}$$

Q15. Indiquer l'information apportée par la diagonale de la matrice. Exploiter les valeurs de la première ligne de cette matrice en expliquant les informations que l'on peut en tirer. Faire de même avec la première colonne. En déduire à quoi sert cette matrice.

On peut également tracer l'efficacité de l'algorithme pour différentes valeurs de K sur le jeu de données test (100 éléments sur un échantillon de 310 données). On trace le pourcentage de réussite en fonction de la valeur de K sur la figure 4.

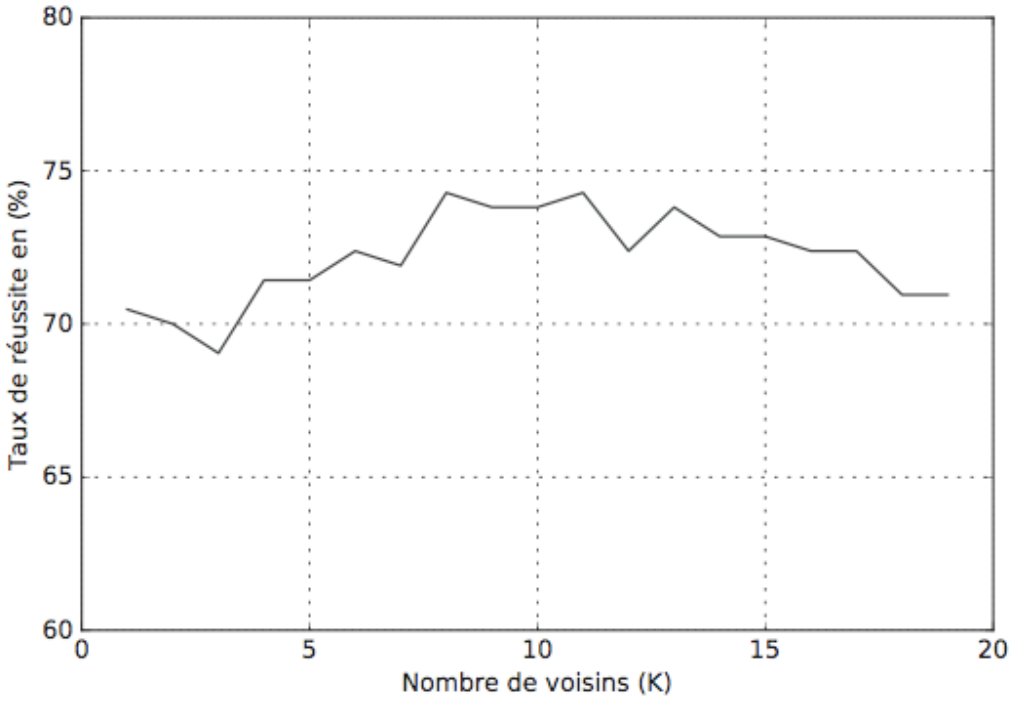

Figure 4 – Pourcentage de réussite de l'algorithme en fonction de K

Q16. Commenter la courbe obtenue et critiquer l'efficacité de l'algorithme.

ANNEXE : Rappels des syntaxes en Python et sur les tableaux Numpy (array)

tableau à une dimension possédant *n* éléments, valant tous 0	tab=np.zeros(n)

Creation d'un tableau tab à une dimension à partir d'une liste Python L	L=[1,2,3] # liste Python
	tab=np.array([1,2,3]) # ou np.array(L)

tableau à une dimension à *n* éléments, uniformément répartis entre deux valeurs debut et fin (inclus)	debut=0 ; fin=10 ; n=5 tab=np.linspace(debut, fin, n)
Accès à un élément du tableau tab avec tab[i]	tab[2]=100 print (tab) >>> array([0, 2.5, 100, 7.5, 10])
Ajouter un élément à la queue (pour les listes seulement)	L.append(10)

Tableau à deux dimensions (matrice)	LL=[[1,2,3],[4,5,7]] # liste de listes
	M=np.array(LL)

Tableau à deux dimensions, rempli de zéros (avec nl lignes et nc colonnes)	M=np.zeros((nl,nc))
Accès à l'élément i,k du tableau à deux dimensions	M[i,k]
Extraire une partie de tableau (2 premières colonnes)	M[: , 0:2]
Extraire la colonne k	M[: ,k]
Extraire la ligne i	M[i, :]
Dimension d'un tableau de taille (i,k)	M.shape donne (i,k)
Première dimension d'un tableau (ou taille d'une liste)	len(M) # nombre de lignes de M
Deuxième dimension d'un tableau	len(M[0]) # nombre de colonnes de M
Tracé d'une courbe à partir de 2 listes X et Y	plt.plot(X,Y)
Tracé d'une courbe 3d à partir de 3 listes X, Y et Z	plt.gca(projection='3d').plot(X,Y,Z)