Corrigé info 2017

Une cible solidaire de l'arbre moteur dispose de soixante dents. Les dents sont numérotées de 1 à 60. L'épaisseur d'une dent est égale à l'espace entre deux dents. Un capteur fixe par rapport au carter moteur permet de détecter la présence ou l'absence de dent sur la cible.

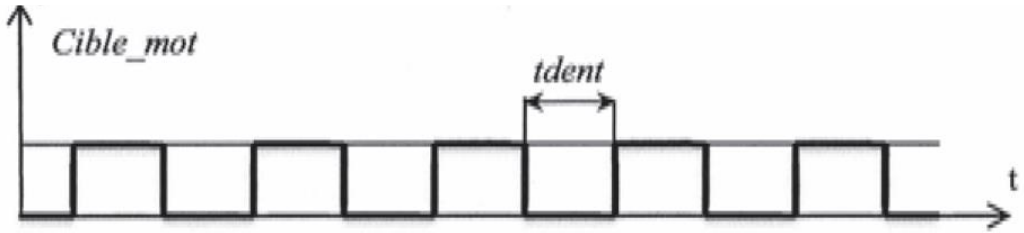


figure 1 : évolution de la variable *cible_mot*.

La lecture de l'état des variables associées aux principaux capteurs est réalisée par le logiciel de gestion du moteur avec une période d'échantillonnage de 2 ms au mieux. L'arbre moteur tourne jusqu'à 7000 tours/min.

Q1. Quelle est la valeur minimale du temps écoulé entre deux changements d'état de la variable *cible_mot* ? La période d'échantillonnage est-elle suffisante pour détecter le changement d'état entre deux fronts de la variable *cible_mot* ?

60 dents sur la périphérie avec un front montant et descendant par dent,
7000 tr/min, ça fait
60*2*7000 = 840000 impulsions en une minute, soit 14000 par seconde
il y donc une mesure tous les 1/14000 = $7{,}14.10^{-05}$ s
C'est beaucoup plus faible que 2.10^{-03}s, la période d'échantillonnage est trop faible

Q2. Ecrire une fonction *vitesse_moteur(tdent)* qui calcule et retourne la vitesse de rotation du moteur en tours par minute en fonction de la variable *tdent*.

```
def vitesse_moteur(tdent):
    """quand on a 120 tdent, on a fait un tour"""
    N = tdent*120    #secondes pour un tour
    N = N/60    #minutes pour un tour
    return 1/N    # en tour/minutes
```

A taper et tester

C / CONTROLE DU DOSAGE AIR-ESSENCE

Toutes les cartographies et programmes du moteur sont stockés sous forme de fichiers dans la mémoire morte du calculateur (ROM) qui dispose de 32 ko d'espace. Au démarrage du véhicule, le programme de gestion du moteur et certaines données seront chargés dans la mémoire vive (RAM) qui dispose de 3 ko d'espace.

Q3. Expliquer les différences entre les mémoires de type ROM ou RAM. Pourquoi stocker le programme de gestion dans la RAM ?

La mémoire ROM est une mémoire dite morte, elle n'est pas modifiable a priori, on n'y accède qu'en lecture. La mémoire RAM en revanche est volatile, on peut lire et écrire dans la RAM très facilement. A chaque extinction du système la mémoire RAM s'efface.

L'accès à la mémoire RAM est plus rapide que l'accès à la ROM, dans la ram on peut écrire par exemple des variables qui représentent des grandeurs du moteur, qui ne sont pas constantes, ce qui peut justifier ce choix.

Q4. Exprimer en bits l'espace mémoire disponible pour la mémoire RAM et la mémoire ROM sous la forme : $(nbits)_{10} = a \cdot b^i$. Préciser les valeurs numériques de a, b et i sans réaliser l'application numérique de $(nbits)_{10}$.

Les espaces mémoire sont donnés en ko (kilo-octet). Un octet représente 8 bits et un kilo octet représente 10^3 octets. Soit les résultats suivants :

- RAM : 3ko → $(nbits)_{10} = 3 \cdot 8 \cdot 10^3$
- ROM : 32ko → $(nbits)_{10} = 32 \cdot 8 \cdot 10^3$

Cette question est étrange. Peut-être a-t-on voulu parler de kibioctets, à ce moment-là il aurait fallu écrire en toute rigueur 3kio et 32kio et le résultat aurait été :

- RAM : 3kio → $(nbits)_{10} = 3 \cdot 2^{10}$
- ROM : 32kio → $(nbits)_{10} = 32 \cdot 2^{10}$

1Mega-octet =1024 octet, pas 1000

Pour accéder à la mémoire, le processeur dispose d'un bus d'adresse de 24 bits. Cela signifie que pour communiquer en lecture ou en écriture avec un emplacement mémoire particulier, le processeur envoie un mot binaire codé sur 24 bits via le bus d'adresse. On appelle espace adressable le nombre d'adresses différentes accessibles par le bus d'adresse.

Q5. Quel est alors l'espace adressable en décimal ? Donner l'adresse maximale en binaire (base 2).

L'espace adressable est de 2^{24} adresses, soit 16 777 216 adresses.

L'adresse maximale en binaire est : $(1111\ 1111\ 1111\ 1111\ 1111\ 1111)_2$ ② Indique la base 2

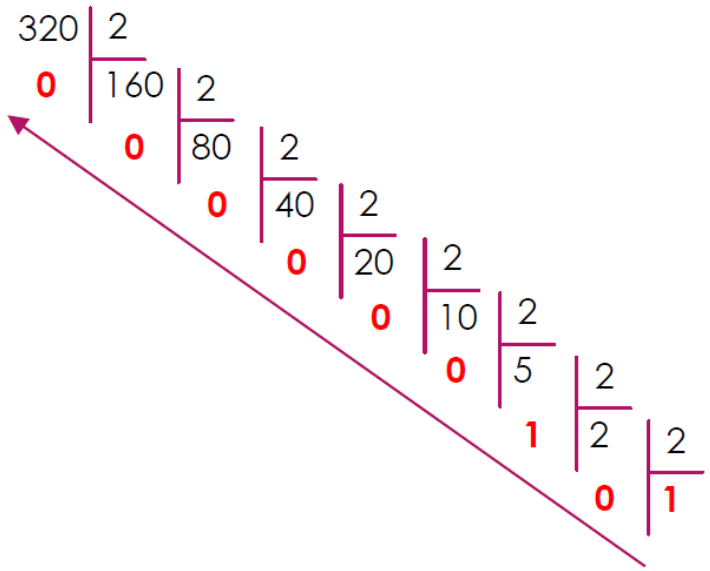
24 bits à 1

base 10 : d_i=	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
base 16 : h_i=	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

On a alors pour un nombre n écrit en base 16 $(n)_{16} = h_k \dots h_1 h_0$ la conversion en base 10 :

$$(n)_{10} = \sum_{i=0}^{k} d_i \cdot 16^i$$

Q6. Ecrire en binaire puis en hexadécimal l'adresse suivante : $(320)_{10}$.



$(320)_{10} = (101000000)_2$

Conversion de la 2 vers la base 16

$$\underbrace{0001}_{1}\ \underbrace{0100}_{4}\ \underbrace{0000}_{0}$$

$(101000000)_2 = (140)_{16}$

ou

320/16 = 20 reste 0
20/16 = 1 reste 4
1/16 = 0 reste 1
donc c'est 140 en héxa"""

Q7. Quel est l'intérêt d'utiliser le code hexadécimal pour la programmation ?

le code hexadecimal est beaucoup plus compact et lisible. De plus la conversion est instantanée puisqu'un chiffre en hexadécimal correspond à 4 bits en binaire. Les données étant organisées en octets (8 bits), chaque octet est représenté par 2 chiffres en hexadécimal.

C.3 Cartographie des durées d'injection

Nous allons nous intéresser au traitement de la cartographie qui permet de déterminer la durée d'injection. La figure 2 représente l'affichage d'une cartographie des durées d'injection pour un moteur 4 temps.

		Pression au collecteur **P** en bars						
		0,295	0,39	0,48	0,565	0,645	0,72	0,79
	600	2,42	3,454	4,408	5,44	6,484	7,522	8,548
	900	2,702	3,776	4,852	5,9	6,962	8,004	9,036
	1300	3,064	4,162	5,248	6,33	7,418	8,432	9,434
ligne i	1700	3,27	4,412	5,552	6,644	7,734	8,774	9,766
	2200	3,432	4,63	5,804	6,952	8,062	9,126	10,154
Rotation	2700	3,498	4,71	5,916	7,09	8,23	9,334	10,39
moteur **Nm**	3200	3,56	4,778	6,022	7,214	3,388	9,552	10,614
en tour/min	3800	3,658	4,878	6,168	7,358	8,558	9,742	10,844
	4400	3,74	5,008	6,324	7,51	8,738	9,962	11,066
	5000	3,816	5,134	6,48	7,708	8,962	10,204	11,32
	5600	3,908	5,29	6,622	7,89	9,174	10,408	11,582
	6300	3,99	5,39	6,754	8,076	9,372	10,612	11,79
	7000	4,03	5,436	6,808	8,152	9,452	10,7	11,898

colonne j

figure 2 : cartographie des durées d'injection en ms.

La cartographie est alors chargée en mémoire sous la forme suivante (voir figure 3) :

- La première ligne qui contient des valeurs de pression à l'admission en bar est stockée dans une liste de valeurs : $P = [P_j]_{0 \leq j \leq M-1}$.
- La première colonne qui contient des valeurs de la vitesse de rotation moteur en tour/min est stockée dans une liste de valeurs : $Nm = [Nm_i]_{0 \leq i \leq N-1}$.
- Les durées d'injection sont stockées sous forme d'un tableau T à deux dimensions de taille $N \times M$. On peut lire pour chaque couple de valeurs de pression et de rotation moteur la valeur de la durée d'injection correspondante $T[i,j]$ en ms.

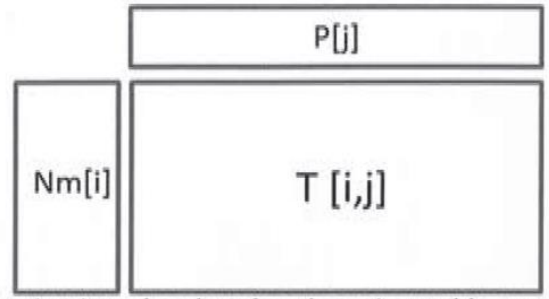


figure 3 : structure des données de cartographie en mémoire.

Etape 1 :

Le calculateur doit déterminer les indices i et j tels que $P[j] \leq Pcol < P[j+1]$ et $Nm[i] \leq Nmot < Nm[i+1]$.

Q8. Ecrire une fonction *indice(A,val)* qui prend pour argument une liste notée *A* et un réel noté *val*. La liste *A* est triée par ordre croissant. Votre fonction doit retourner un entier *id* tel que : $A[id] \leq val < A[id+1]$. On supposera que *id* existe toujours.

2 solutions

```
def indice(A, val):
    for i in range(1, len(A)):
        if A[i] > val: # On relève l'indice à partir duquel la valeur devient supérieure à val
            return i - 1
```

```
def indice(A, val):
    for i, v in enumerate(A):
        if v > val: # On relève l'indice à partir duquel la valeur devient supérieure à val
            return i-1
```

Le calculateur doit ensuite lire les durées d'injection dans le tableau T correspondant aux indices i et j précédemment déterminés.

Pour la suite du sujet, un tableau pourra être représenté :

- **soit par un tableau de type numpy.array ;**
- **soit par une liste de listes.**

Q9. Ecrire une fonction *extraire(T, P, Nm, i, j)* qui prend pour argument le tableau de dimension $N \times M$ noté T, les listes P et Nm, et deux entiers i et j. Votre fonction doit retourner un tableau ST de dimension 2X4 :

$$ST = \begin{bmatrix} T[i,j] & T[i,j+1] & P[j] & Nm[i] \\ T[i+1,j] & T[i+1,j+1] & P[j+1] & Nm[i+1] \end{bmatrix}$$

Solution listes

```
def extraire(T, P, Nm, i, j):
    return  [[T[i][j],  T[i][j  +  1],
P[j], Nm[i]], \ # Le \ permet d'aller à
la ligne
        [T[i + 1][j], T[i + 1][j + 1],
P[j + 1], Nm[i + 1]]]
```

Solution numpy

```
def extraire(T, P, Nm, i, j):
    ST = T[i:i+2, j:j+2]
    ST = np.hstack(ST, P[j: j+2])
    ST = np.hstack(ST, Nm[i: i+2])
    return ST
```

Q10. Ecrire la suite d'instructions qui à partir des variables $Nmot$, $Pcol$, des listes P et Nm et du tableau T permet de déterminer le sous tableau ST tel que défini à la question Q9. Le résultat pourra être affecté à une variable nommée ST.

```
>>>idp = indice(P, Pcol)
>>>idNm = indice(Nm, Nmot)
>>>ST = extraire(T, P, Nm, idNm, idp)
```

Q11. Ecrire en pseudo-code dans le cas d'une matrice carrée de dimension $n \times n$ l'algorithme du pivot de Gauss permettant de résoudre le système $A \cdot b = c$. On admettra que les termes diagonaux sont tous non nuls.

La question de la mort !

Le mieux si vous n'avez pas appris l'algo par cœur est de répondre par une phrase du type:

- On utilise gauss Jordan avec pivot partiel;
- On cherche d'abord la plus grande valeur absolue dans la colonne et on la prend comme pivot;
- On fait ensuite une transvection au dessus et en dessous pour faire apparaitre des zéros;
- A la fin on a une matrice diagonale avec une colonne sur la droite;
- Il ne reste plus qu'a diviser le terme de droite par le terme correspondant sur la diagonale.

Sinon: Algorithme en pseudo-code, résolution de $A \cdot b = c \Rightarrow b = A^{-1} \cdot c$

```
r = 0
Pour j de 0 à n - 1
   Rechercher Max(|A[i,j]|) pour i entre r + 1 et n - 1
   Noter k l'indice de la ligne de ce Max
   Si A[k,j] ≠ 0 alors (c'est le cas d'après les hypothèses)
      r ← r + 1
      Echanger les lignes k et r de A et c (échange)
      c[r] ← c[r] / A[r,j] (dilatation en premier sur c avant de modifier A[r,j])
      A[r,:] ← A[r,:] / A[r,j] (dilatation)
      Pour i de 0 à n - 1
         Si i ≠ r alors
            c[i] ← c[i] – c[r] * A[i,j] (transvection)
            A[i,:] ← A[i,:] - A[r,:] * A[i,j] (transvection)
         Fin Si
      Fin Pour
   Fin Si
Fin Pour
b ← c (le résultat est stocké dans c)
```

Q12. Quelle est la complexité du pivot de Gauss Justifier votre réponse.

La boucle externe est répétée O(n) fois. À chaque étape j, la recherche de pivot coûte O(n) opérations, tout comme les échanges de lignes et les dilatations. Chaque transvection coûte O(n) opérations et est répétée O(n) fois. Les transvections sont donc en O(n^2) et même en Θ(n^2).

On posera pour la suite :

$$x = \frac{Pcol - P[j]}{P[j+1] - P[j]} \text{ ; et } y = \frac{Nmot - Nm[i]}{Nm[i+1] - Nm[i]}$$

On connaît les valeurs de $t(x,y)$ pour quatre couples de valeurs (x,y) par lecture de la cartographie :

$$\begin{pmatrix} t(x=0,y=0) = ST[0,0] & t(x=1,y=0) = ST[0,1] \\ t(x=0,y=1) = ST[1,0] & t(x=1,y=1) = ST[1,1] \end{pmatrix}$$

Le problème à résoudre devient alors :

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} ST[0,0] \\ ST[0,1] \\ ST[1,0] \\ ST[1,1] \end{bmatrix}$$

On peut donc résoudre le système de façon systématique par substitution de la manière suivante :

$$\begin{aligned} b_0 &= ST[0,0] \\ b_1 &= ST[0,1] - ST[0,0] \\ b_2 &= ST[1,0] - ST[0,0] \\ b_3 &= ST[1,1] - ST[0,1] - ST[1,0] - ST[0,0] \end{aligned}$$

Q13. Comparer la complexité de cet algorithme à celui présenté à la question Q11. Conclure.

En ne considérant que les 4 équations donnant les b_i, l'algorithme réalise 9 opérations élémentaires (soustractions). On ne parle pas ici de complexité en O(n), ce qui serait aberrant pour n petit (ici n=4).
Cette résolution est donc beaucoup plus rapide que la résolution par la méthode du pivot de Gauss, ce qui est indispensable pour cette application « temps réel ».
Par ailleurs, ici on parle d'inversion d'une matrice triangulaire très simple, qui se fait très bien à la main, d'où l'inutilité du pivot de Gauss.

Q14. Ecrire une fonction *interpol(ST,Pcol,Nmot)* qui retourne une valeur de durée d'injection obtenue par interpolation bilinéaire des éléments de la table.

Erreur dans l'énoncé sur le calcul de b_3 : $b_3 = ST[1,1] - ST[0,1] - ST[1,0] + ST[0,0]$

```
def interpol(ST, Pcol, Nmot):
    b0 = ST[0, 0]
    b1 = ST[0, 1] - ST[0, 0]
    b2 = ST[1, 0] - ST[0, 0]
    b3 = ST[1, 1] - ST[0, 1] - ST[1, 0] + ST[0, 0]
    x = (Pcol - ST[0, 2]) / (ST[1, 2] - ST[0, 2])
    y = (Nmot - ST[0, 3]) / (ST[1, 3] - ST[0, 3])
    return b0 + b1 * x + b2 * y + b3 * x * y
```