

INFORMATIQUE BANQUE PT 2017 : CORRIGE

Q1. $f = 7000 \text{ tours/min} = 116,7 \text{ tours/s}$, donc un tour en $1/116,7 \text{ s} = 8,571 \text{ ms}$

Or 60 dents sur un tour, donc $t_{dent} = 8,571 / 120 = 0,07143 \text{ ms}$.

$T(\text{échantillonnage}) = 2 \text{ ms} \gg t_{\text{dent}}$, donc $T(\text{échantillonnage})$ est trop grand.

Cet échantillonnage ne permet pas de détecter le changement d'état entre deux fronts de la variable cible mot.

Q2. Notons f la vitesse de rotation du moteur en tours par minutes.

Il y a 60 dents et 60 espaces entre les dents, tous de même taille.

Donc on fait un tour complet en un temps $t = (120 \times t_{\text{dent}})$ secondes (avec t_{dent} en secondes).

Or 1 min = 60 s, donc $t = (120 \times t_{\text{dent}} / 60)$ minutes (avec t_{dent} en secondes).

Or $f = 1 / t$ par définition de f .

Donc f [en tours/min] = $1 / (2 \times t_{\text{dent}}$) (avec t_{dent} en secondes).

```
def vitesse_moteur(tdent) :
```

```
return 1 / (2 * tident)
```

Q3.

ROM	RAM
Non volatile (le contenu perdure malgré l'absence d'alimentation)	Volatile (le contenu disparaît en l'absence d'alimentation)
Temps d'accès : quelques 10 ns	Temps d'accès : quelques 1 ns

Programme de gestion stocké dans la RAM car le temps d'accès est plus faible.

Q4. 1 ko = 1024 o = 2^{10} o = 2^{13} bits car 1 o = 8 bits

RAM : 3 ko = $3 \cdot 2^{13}$ bits, donc $a = 3$; $b = 2$; $i = 13$

ROM : 32 ko = $32 \cdot 2^{13} = 2^{18}$ bits, donc a = 1 ; b = 2 ; i = 18

Q5. Espace adressable en décimal : 2^{24} (de 0 à $2^{24} - 1$)

Adresse maximale en binaire : 24 fois le chiffre 1 : 1111 1111 1111 1111 1111 1111

[illegible]

$$\boxed{(320)_{10} = (1\ 0100\ 0000)_2}$$

Autre méthode :

On peut aussi sortir les puissances de 2 : $320 = 5 \times 64 = 5 \times 2^6$

Or $(5)_{10} = (101)_2$ donc on rajoute six 0 en base 2 : $(320)_{10} = (1\ 0100\ 0000)_2$

$$\boxed{(320)_{10} = (1\ 0100\ 0000)_2 = (140)_{16}} \text{ car } (0000)_2 = (0)_{16} ; (0100)_2 = (4)_{16} ; (1)_2 = (1)_{16}$$

Autre méthode :

$$\begin{array}{r|l} 320 & _16 \\ \hline 0 & 20 \\ & _16 \\ & 4 \\ & 1 \\ & _16 \\ & 1 \\ & 0 \end{array}$$

$$\text{Donc } (320)_{10} = (140)_{16}$$

- Q7.**
- Ecriture plus compacte qu'en binaire
 - Facile de passer du binaire à l'hexadécimal, et réciproquement
 - 1 octet = 8 bit = 2 chiffres en hexadécimal

Q8.

def indice(A, val) :

```
id = 0
while val >= A[id + 1]:
    id += 1
return id
```

Q9.

def extraire(T, P, Nm, i, j):

```
ST = [[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]]]
return ST
```

Version NumPy :

def extraire(T, P, Nm, i, j):

```
ST = np.zeros((2, 4))
ST[:2, :2] = T[i:i+2, j:j+2]
ST[:, 2] = P[j:j+2]
ST[:, 3] = Nm[i:i+2]
return ST
```

Q10.

i = indice(Nm, Nmot)

j = indice(P, Pcol)

ST = extraire(T, P, Nm, i, j)

Q11. Cf cours ou TP

Q12. Cf cours : $O(n^3)$

Q13. Supposons que la matrice est triangulaire inférieure, de taille n. La complexité de la méthode proposée, qui est une remontée de pivot, est en $O(n^2)$: il y a k opération à la k-ième ligne, donc au total $1 + 2 + 3 + \dots + n = n(n+1)/2$ opérations, c'est-à-dire $O(n^2)$.

Q14.

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])
t = b0 + b1 * x + b2 * y + b3 * x * y
return t
```

Q15.

```
def sonde(r) :
    lambda = 1 / r
    if lambda < 1 :
        Usonde = 0.9
    else :
        Usonde = 0.1
    return Usonde
```

Q16.

```
def duree_injection(Usonde, Kpp, Kpn, Ki, tinjc0, integri, dt) :
    if Usonde > 0.5 :
        integri1 = integri - Ki * tinjc0 * dt
        tinji1 = tinjc0 * Kpn + integri1
    else :
        integri1 = integri + Ki * tinjc0 * dt
        tinji1 = tinjc0 * Kpp + integri1
    return tinji1, integri1
```

Q17. $\tau \frac{s(t_{i+1}) - s(t_i)}{dt} + s(t_i) = K e(t_i)$ Donc $s(t_{i+1}) = K \frac{dt}{\tau} e(t_i) + \left(1 - \frac{dt}{\tau}\right) s(t_i)$

Q18.

```
def Euler(tau, K, dt, si, ei) :
    si1 = K * dt / tau * ei + (1 - dt / tau) * si
    return si1
```

Q19.

```
def richesse(tau1, tau2, K, dt, ri, wi, tinji) :
    ri1 = Euler(tau2, 1, dt, ri, wi)
    wi1 = Euler(tau1, K, dt, wi, tinji)
    return ri1, wi1
```

Q20. 4000 tours/min et 2 cycles moteur par tour, donc 8000 cycles moteur / min

Donc Tcycle = 1 / 8000 min = 60 / 8000 s = 0,0075 s = 7,5 ms = Tcycle

Q21.

```
temps = [t0]
t = t0
while t < tn :
    t = t + dt
    temps.append(t)
```

Remarque : si le temps initial t0 et le pas de temps dt sont imposés, on ne peut pas imposer plus que temps[-1] <= tn. En particulier on ne peut pas imposer l'égalité.

Q22.

```
liste_temps_U = [] # Contient la liste des instants où U est lue
for t in temps:
    # A chaque instant,
    if len(liste_temps_U) * Tcycle <= t:
        # on regarde si le moment de lire une nouvelle valeur de U est venu
        liste_temps_U.append(t)
```

```

liste_w = [w0]
liste_integ = [integ]
for i in range(len(temps) - 1):
    # Question 19:
    rich, w = richesse(tau1, tau2, K, dt, liste_richesse[i], liste_w[i], liste_tinj[i])
    liste_richesse.append(rich)
    liste_w.append(w)
    # Question 16:
    tinj, integ = duree_injection(liste_U[i], Kpp, Kpn, Ki, tinjc0, liste_integ[i], dt)
    liste_tinj.append(tinj)
    liste_integ.append(integ)
    if temps[i] in liste_temps_U:
        # Si l'instant correspond à un moment où la sonde lit U
        # Question 15:
        liste_U.append(sonde(liste_richesse[i]))
    else:
        # Sinon, U ne change pas:
        liste_U.append(liste_U[i-1])

```

Q23.

```

import matplotlib.pyplot as plt
plt.plot(temps, liste_richesse)
plt.axis([0, 0.8, 0.9, 1.02])
plt.xlabel('temps (s)')
plt.legend('richesse')
plt.show()

```

Q24. CREATE TABLE table_injection_95 AS (SELECT * FROM table_injection WHERE Qualité > 95)

Q25. SELECT Pression, Rotation, Durée_injection FROM table_injection_95 WHERE Qualité = (SELECT MAX(Qualité) FROM table_injection_95 WHERE (Pression > 0,3) AND (Pression < 0,4) AND (Rotation > 1300) AND (Rotation < 1700))