Informatique - Corrigé DM1

Q18. La précision demandée est 0,02 mm Hg : on choisit donc de représenter cette valeur par une unité de la valeur numérique : 1 bit <-> 0,02 mm Hg.

On veut échantillonner des valeurs jusqu'à 1350 hPa ce qui donne 1350/1013*750 = 999,5 mmHg.

Cela représente une valeur après numérisation de 999,5/0,02 = 49975

10 bits permettent de représenter des entiers de 0 à 2**10 = 1024, 12 bits donnent une plage 0 à 4096 et 16 bits de 0 à 65536.

On choisit des CAN avec une résolution de 16 bits.

Et, on règle les tensions pour avoir 1 bit $\leftrightarrow$ 5/65536 = 7,6.10^5 V $\leftrightarrow$ 999,5/65536 = 0,016 mm Hg.

Q19. La durée d'échantillonnage multipliée par la fréquence d'échantillonnage donne 60*1000 = 60000 valeurs numériques à stocker.

Une valeur numérique est une valeur entière sur 16 bits entre 0 : tension de 0V et 65536 : tension de 5V. Alors on choisit de stocker les valeurs sous forme d'entiers non signés sur 16 bits. C'est le stockage optimal.

Un entier sur 16 bits est stocké dans 2 octets, alors le stockage nécessite 60000 $\times$ 2 = 120000 octets.

Q20. Il s'agit d'un filtre analogique du deuxième ordre.

La forme de l'équation différentielle $\frac{1}{\omega_0^2}\ddot{U} + \frac{1}{\omega_0 Q}\dot{U} + U = Ue$ nous indique qu'il s'agit d'un filtre passe-bas de pulsation de coupure $\omega_0 = 20$ rad/s et de facteur de qualité $Q = 1/(2z) = 0,71$.

La fréquence de coupure est $f_0 = \omega_0/(2 \times \pi) = 3,183$ Hz et la fréquence des battements est $f = 190$ min^{-1} $= 3,167$ Hz.

Le choix de la pulsation ω supérieure à la pulsation maximale des battements du coeur permet de ne conserver que la fréquence principale des pulsations cardiaques.

Le facteur de qualité inférieur à $\sqrt{2}/2$ permet d'éviter tout phénomène de résonance.

Q20,5. Le schéma numérique de la méthode d'Euler est

$$\dot{U}_{f,i+1} = \dot{U}_{f,i} + T_e\ddot{U}_{f,i}$$
$$U_{f,i+1} = U_{f,i} + T_e\dot{U}_{f,i}$$

Q21. En notant U pour U_f et U_e pour U_e, on écrit l'équation différentielle au temps i+1 ce qui donne

$\ddot{U}_{i+1} = \omega^2 U_{e,i+1} - 2z\omega\dot{U}_{i+1} - \omega^2 U_{i+1}$

$\ddot{U}_{i+1} = \omega^2 U_{e,i+1} - 2z\omega\left(\dot{U}_i + (1-\gamma)T_e\ddot{U}_i + \gamma T_e\ddot{U}_{i+1}\right) - \omega^2\left(U_i + T_e\dot{U}_i + T_e^2(1/2-\beta)\ddot{U}_i + T_e^2\beta\ddot{U}_{i+1}\right)$

$\ddot{U}_{i+1} = \omega^2 U_{e,i+1} - 2z\omega\dot{U}_i - 2z\omega(1-\gamma)T_e\ddot{U}_i - 2z\omega\gamma T_e\ddot{U}_{i+1} - \omega^2 U_i - \omega^2 T_e\dot{U}_i - \omega^2 T_e^2(1/2-\beta)\ddot{U}_i - \omega^2 T_e^2\beta\ddot{U}_{i+1}$

$(2z\omega\gamma T_e + \omega^2 T_e^2\beta + 1)\ddot{U}_{i+1} = \omega^2 U_{e,i+1} + (-2z\omega(1-\gamma)T_e - \omega^2 T_e^2(1/2-\beta))\ddot{U}_i - (2z\omega + \omega^2 T_e)\dot{U}_i - \omega^2 U_i$

$$\ddot{U}_{i+1} = \frac{\omega^2}{2z\omega\gamma T_e + \omega^2 T_e^2\beta + 1}\left(U_{e,i+1} + \left(-2\tfrac{z}{\omega}(1-\gamma)T_e - T_e^2(1/2-\beta)\right)\ddot{U}_i - \left(2\tfrac{z}{\omega} + T_e\right)\dot{U}_i - U_i\right)$$

Il s'ensuit que

$$B_1 = \frac{\omega^2}{1 + 2z\omega\gamma T_e + \omega^2 Te^2\beta}, \quad B_2 = -\left(Te^2(1/2-\beta) + \tfrac{2z}{\omega}(1-\gamma)T_e\right), \quad B_3 = -\left(T_e + \tfrac{2z}{\omega}\right), \quad B_4 = -1$$

Q22. La figure 7 représente le résultat de la simulation de la réponse du filtre à un échelon de tension de 1 V.

L'équation différentielle régissant ce filtre est de la forme $\ddot{s} + 2z\omega_0\dot{(s)} + \omega_0^2 s = \omega_0^2 e$

Le discriminant de l'équation caractéristique est $\Delta = 4z^2\omega_0^2 - 4\omega_0^2 = 4\omega_0^2(z^2-1)$.

Ici, $z = 0,7$ donc $z^2 - 1 < 0$ donc le régime est pseudo-périodique.

On sait que le coefficient d'amortissement α (noté z dans l'énoncé) vérifie $\alpha < 1$ alors les solutions de l'équation différentielle sont pseudo-périodiques de la forme $e^{at}(A\cos(bt) + B\sin(bt))$ et tendent rapidement vers une limite finie car $b < 0$ et cette limite est la valeur de U_e.

On constate que pour $T_e > 0,175$ les simulations numériques ne tendent pas vers une limite finie.

La simulation n'est donc pas correcte pour $T_e > 0,175$.

Pour $T_e < 0,13$, la simulation donne un résultat qui semble cohérent avec la solution attendue.

Q23. L'erreur η est proportionnelle au pas de temps $\eta \simeq 5T_e$. On dit que la convergence est linéaire ou d'ordre 1.

Q24. Avec la fréquence d'échantillonnage de 1000 Hz, on a $T_e = 0,001$ alors on obtient une bonne stabilité car $0,001 < 0,175$ et une erreur raisonnable car l'erreur maximale est $\eta \simeq 0,005$ (en Volts).

Q25. L'algorithme attendu par le jury du concours utilise des listes :

```
def newmark(gamma, beta, omega, z, e, Te):
    N = len(e)
    U = [0]
    Up = [0]    # U'
    Upp = [0]   # U''
    for n in range(N-1):
        Upp.append(B1*(e[n+1] + B2*Upp[n] + B3*Up[n] + B4*U[n]))
        Up.append(Up[n] + (1-gamma)*Te*Upp[n] + gamma*Te*Upp[n+1])
        U.append(U[n] + Te*Up[n] + Te**2*(1/2 - beta)*Upp[n] + Te**2*beta*Upp[n+1])
    return U # toutes les valeurs de U
```

Une version utilisant **numpy**

```
def newmark(gamma, beta, omega, z, e, Te):
    N = e.shape[0]
    U, Up, Upp = np.zeros(N) , np.zeros(N) , np.zeros(N)
    for n in range(N-1):
        Upp[n+1] =  B1*(e[n+1] + B2*Upp[n] + B3*Up[n] + B4*U[n])
        Up[n+1] = Up[n] + (1-gamma)*Te*Upp[n] + gamma*Te*Upp[n+1]
        U[n+1] = U[n] + Te*Up[n] + Te**2*(1/2 - beta)*Upp[n] + Te**2*beta*Upp[n+1]
    return list(U) # toutes les lignes et la colonne 2
```

Q26. On ne conserve que les valeurs de U, U', U'' pour n (valeur courante) et $n+1$ (valeur suivante)

```
def newmark(gamma, beta, omega, z, e, Te):
    N = len(e)
    U = [0]
    upp_courant, up_courant, u_courant  = 0,0,0     # notations upp, up, u pour U'', U', U
    upp_suivant, up_suivant, u_suivant = 0, 0, 0    # valeurs au rang suivant
    for n in range(N-1):
        upp_suivant = B1(*e[n+1] + B2*upp_courant + B3*up_courant + B4*u_courant)
        up_suivant = up_courant + (1-gamma)*Te*upp_courant + gamma*Te*upp_suivant
        u_suivant = u_courant + Te*up_courant + Te**2*(1/2 - beta)*upp_courant + Te**2*beta*upp_suivant
        U.append(u_suivant)
        upp_courant,up_courant,u_courant = upp_suivant,up_suivant,u_suivant
    return U
```

Q27. On parcourt la liste, par temps croissants, en cherchant le motif : maximum local suivi d'un minimum local.

On commmence par chercher un temps t où la pression augmente : `data[k] <= data[k+1]`.

Puis, on détecte un maximum local dès que deux valeurs ne sont plus dans l'ordre croissant `data[k]> data[k+1]`.

La suite des valeurs sont alors décroissantes. On détecte un minimum local dès que deux valeurs successives ne sont plus dans l'ordre décroissant.

Alors, on teste le critère. On poursuit le parcours jusqu'à ce que le critère soit vérifié.

```
def pression_systolique(data):
    k = 0
    while data[k] >= data[k+1]:       # on cherche une phase de croissance de la  pression
        k += 1
    while True:                       # l'énoncé garantit qu'on s'arrête avant la fin de la liste
        while data[k] <= data[k+1]:   # recherche d'un maximum local
            k += 1
        maxi = data[k]                # fin de la croissance : on a un maximum
        while data[k] >= data[k+1]:   # recherche d'un minimum local
            k += 1
        mini = data[k]                # fin de la décroissance : on a un minimum
        if (maxi - mini)/mini > 4e-5: # critère pour la pression systolique
            return maxi               # on arrête dès que le critère est vérifié
```

Q28. Pour déterminer le dernier instant où le critère est vérifié, il faut soit parcourir toute la liste par temps croissants et ne renvoyer que le dernier, soit partir de la fin et trouver le premier instant où le critère est vérifié.

On utilise la deuxième méthode.

```
def pression_diastolique(data):
    Pmin = 40                       # 40 mm Hg d'après le début de la partie
    k = len(data) -2
    while data[k] >= data[k+1]:     # on cherche une phase de croissance de la  pression
        k -= 1
    while True:                      # l'énoncé garantit qu'on s'arrête avant le début de la liste
        while data[k-1] <= data[k] :  # on recherche d'abord un minimum local
            k = k-1                  # on remonte dans le temps
        mini = data[k]               # fin de la croissance : on a un minimum local
        while data[k-1] >= data[k] :  # on recherche le maximum local suivant
            k = k-1                  # on remonte dans le temps
        maxi = data[k]               # fin de la décroissance : on a um maximum
        if  maxi >= Pmin and (maxi - mini)/mini > 4e-3:   # critère pour la pression diastolique
            return maxi              # on arrête dès que le critère est vérifié
```

Q29.

Une clé primaire identifie de manière unique chaque tuple d'une relation. Elle garantit à la fois l'unicité de chaque tuple et l'accès à un tuple donné.

Le numéro de sécurité sociale n'est pas une clé primaire raisonnable car les étrangers et les enfants n'en ont pas. Aucun des autres attributs n'est unique.

Q30.

```
SELECT dateheure, pdias, psyst, pouls
FROM mesures
WHERE dateheure>time1 AND dateheure<time2
```

Q31.

```
SELECT dateheure, pdias, psyst, pouls, pid
FROM mesures
WHERE dateheure>time1
    AND dateheure<time2
    AND pid=id_patient
    AND type_m="tension"
```

Q32.

```
import matplotlib.pyplot as plt
temps = resultat_requete[:,0]    # colonne 0
psyst, pdias, pouls  = resultat_requete[:,1], resultat_requete[:,2],  resultat_requete[:,3]
plt.plot(temps,psyst,"-b", label ="pression systolique") # ligne continue bleue
plt.plot(temps,pdias,".k", label ="pression diastolique") # ligne pointillée noire
plt.plot(temps,pouls,"xr", label ="pouls")   # croix rouges
plt.legend()
plt.title("Évolution des constantes du patient")
plt.xlabel("heure de mesure")
plt.show()
```

Q33. Version SQL du calcul : On n'ajoute pas de GROUP BY car on a sélectionné un seul patient

```
SELECT AVG(pdias) AS "moyenne pression diastolique", MIN(pdias), MAX(pdias)
FROM mesures
WHERE dateheure>time1 AND dateheure<time2
    AND pid=id_patient
    AND type_m="tension"
```

Version avec numpy qui n'est pas dans l'esprit de l'énoncé :

```
def analyse(valeurs):
    tab = np.array(valeurs)
    return np.min(tab), np.max(tab),  np.mean(tab)
```

Version naïve dans l'esprit de l'énoncé :

```
def analyse(valeurs):
    min, max, som = valeurs[0], valeurs[0], 0
    for val in valeurs:
        if val<min:
            min = val
        if val>max:
            max = val
        som += val
    return min, max, som/len(val)
```

Q34. La complexité en coût simplifié est dans tous les cas $1+n$ soit $\Theta(n)$.

Q35. On utilise une fonction **triInsertion()** qui trie en place une liste par insertion.

```
def triInsertion(L):
    for i in range(1,len(L)):
        u = L[i]                #  élément à insérer dans la partie triée L[0.i]
        k = i
        while k>0 and L[k-1]>u:  # on remonte dans la liste
            L[k] = L[k-1]        # on décale les éléments trop grands
            k = k-1
        L[k] = u                # on place l'élément à insérer
```

Puis on écrit la fonction **mediane()** qui trie la liste puis renvoie la valeur du milieu de la liste si le nombre de valeurs est impair ou la moyenne des éléments du milieu de la liste si le nombre de valeurs est pair.

```
def mediane(valeurs):
    tri(valeurs) # trie en place
    n = len(valeurs)
    if n%2 == 0: # cas pair : on retourne
                 #  la moyenne des valeurs du milieu
        return (valeurs[n//2-1] + valeurs[n//2])/2
    else:        # cas impair : on retourne
                 #  la valeur du milieu de la liste
        return valeurs[n//2]
```

Q36. Dans le meilleur cas, la liste `valeurs` est déjà triée dans l'ordre croissant alors on n'effectue qu'une comparaison par itérations dans la boucle **for** du tri : $T(n)=n$.

Dans le pire cas la liste est triée par ordre décroissant et le nombre de comparaisons est $T(n)=1+2+3+\ldots+(n-1)=\dfrac{n(n-1)}{2}$

On a donc $T(n)=\mathcal{O}(n^2)$ donc la méthode utilisée est peu efficace.

On pourrait plutôt utiliser le tri fusion qui a une complexité en $\mathcal{O}(n\log(n))$.

Q37. On peut effectuer une jointure entre les 2 tables et on ajoute la clause `DISTINCT` pour ne voir apparaître chaque patient qu'une seule fois (sinon ils apparaissent autant de fois qu'il y a eu des mesures extrêmes).

```
SELECT DISTINCT P.nom, P.prenom, P.telephone
FROM patients AS P
JOIN mesures AS M  ON P.id=M.pid
WHERE M.psyst>160 AND M.pdias>110 AND M.pouls>100 AND M.pouls<150
```

On peut également effectuer une sous-requête pour trouver la table des identifiants des patients concernés et utiliser cette table dans une autre requête :

```
SELECT DISTINCT nom, prenom, telephone
FROM patients
WHERE id IN ( SELECT pid FROM mesures
              WHERE psyst>160 AND pdias>110 AND pouls>100 AND pouls<150 )
```