Devoir d'informatique numéro 2

Correction

Étude d'un tensiomètre électronique (PT 2016)

III Traitement numérique des informations

III.1 Numérisation du signal

Q18. 1350 est environ 1/3 de plus que 1013, donc 1350 HPa correspond à environ 1000 mmHg.
Il faut stocker des valeurs entre 0 et 1000 espacées de 0,02, donc $1000/0{,}02 = 100000/2 = 5.10^4$ valeurs.
Or 12 bits donnent $2^{12} = 4 \times 1024 < 5.10^4$, et 16 bits donnent $2^{16} = 64 \times 1024 > 6.10^4$.

Seul le CAN ayant une résolution de 16 bits permet de respecter le critère de précision.

Q19. Chaque mesure occupe 16 bits. La fréquence d'échantillonnage est de 1000 Hz : il y a 1000 mesures par secondes. Un mesures peut durer jusqu'à 60 secondes : il y a donc jusqu'à 6.10^4 mesures de 16 bits à stocker.
On peut stocker ce que donne le CAN : un entier non signé sur 16 bits, c'est-à-dire 2 octets. Il faut donc, dans ce cas

12.10^4 octets (120ko) de mémoire

Si on stocke les pression en mmHg (donc en multipliant par 0,02), on obtient un flottant, codé sur 32 bits (16 bits n'est ni standard ni proposé), donc il faudrait 24.10^4 octets.

III.2 Filtrage numérique du signal

Q20. On discrétise l'équation différentielle, c'est-à-dire qu'on transforme les fonctions $f(t)$ (ici U_f et ses dérivées $\dot{U}_f$ et $\ddot{U}_f$) en suites $f_i = f(iT_e)$. L'énoncé impose de se placer au pas de temps $i+1$, donc f_{i+1}.
L'équation différentielle discrétisée s'écrit

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

Puis en remplaçant les expressions de $U_{f,i+1}$ et $\dot{U}_{f,i+1}$ par leurs expressions en fonction de $\ddot{U}_{f,i+1}$ et les valeurs des suites au rang i :

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

Puis en rassemblant les termes pour exprimer $\ddot{U}_{f,i+1}$:

$$\Big(\underbrace{\frac{1}{\omega^2} + \frac{2z\gamma T_e}{\omega} + T_e^2\beta}_{=1/B_1}\Big)\ddot{U}_{f,i+1} + \Big(\underbrace{\frac{2z(1-\gamma)T_e}{\omega} + T_e^2(\frac{1}{2}-\beta)}_{=-B_2}\Big)\ddot{U}_{f,i} + \Big(\underbrace{\frac{2z}{\omega} + T_e}_{=-B_3}\Big)\dot{U}_{f,i} + U_{f,i} = U_{e,i+1}$$

Ainsi, $\ddot{U}_{f,i+1} = B_1\Big(U_{e,i+1} + B_2\ddot{U}_{f,i} + B_3\dot{U}_{f,i} + B_4U_{f,i}\Big)$, avec

$$\boxed{B_2 = -\frac{2z(1-\gamma)T_e}{\omega} - T_e^2(\frac{1}{2} - \beta),\ B_3 = -\frac{2z}{\omega} - T_e \text{ et } B_4 = -1}$$

Q21. Une analyse physique du problème montre que la solution de l'équation différentielle doit converger vers une limite finie lorsque le temps t tend vers $+\infty$.
Graphiquement, la solution pour $T_e = 0.2$ diverge, le schéma d'intégration est alors dit « instable ». Pour $T_e = 0.175$, la solution reste bornée mais ne converge pas : on est en limite de stabilité. En conclusion,

$$\boxed{\text{Le schéma est stable pour } T_e < 0.175}$$

Ce que confirme le tableau qui suit.

Q22. En considérant les trois points (T_e, η) donnés, on peut conjecturer que $\eta = 5T_e = O(T_e)$, et donc que

$$\boxed{\text{L}}\text{a convergence est linéaire.}$$

Il n'y a que trois valeurs car c'est un exercice de concours en temps limité. Évidemment, lors d'une vrai analyse scientifique, il faut plus de trois points...

Q23. T_e est l'inverse de la fréquence, dont $T_e = 1/1000 = 0.001$. Pour cette valeur, le schéma converge rapidement, et l'erreur $\eta = 5.10^{-3}$.

Q24. On se contente de traduire la formule donnée à la question 20.

```
1  def newmark(gamma, beta, omega, z, e, Te):
2      """retourne le signal filtré de la liste e, par le schéma de Newmark
           """
3      n = len(e)
4      #valeurs initiales de uf, uf' et uf" :
5      uf, ufp, ufpp = [0], [0], [0]
6      for i in range(n-1):
7          #Equa diff :
8          ufpp.append(B1 * (e[i+1]
9                            + B2 * ufpp[i]
10                           + B3 * ufp[i]
11                           + B4 * uf[i]))
12         #Schema de Newmark :
13         ufp.append(ufp[i]
14                    + (1-gamma) * Te * ufpp[i]
15                    + gamma * Te * ufpp[i+1])
16         uf.append(uf[i]
17                   + Te * ufp[i]
18                   + Te * Te * (0.5-beta) * ufpp[i]
19                   + Te * Te * beta * ufpp[i+1])
20     return uf
```

J'insiste encore une fois sur la rédaction du code : des noms de variables explicites sans être trop long, un code aéré. Un programme est fait pour être relu et modifié. Prenez de bons réflexes.
On peut couper une ligne à l'intérieur d'une parenthèse.

Q25. Dans le calcul des termes d'une suite récurrente, on peut se contenter de garder juste ce qu'il faut pour calculer u_{n+p} et stocker le reste dans des variables intermédiaires que l'on décale.

```
1  def newmarkGlissant(gamma, beta, omega, z, e, Te):
2      """retourne le signal filtré de la liste e, par le schéma de Newmark
           """
3      n = len(e)
4      #valeurs initiales de uf :
```

```
    uf = [0]
    #valeur de uf' et uf" au rang i+1
    ufpi1 , ufppi1 = 0, 0
    for i in range(n-1):
        #On décale les valeurs de uf' et uf"
        ufpi = ufpi1
        ufppi = ufppi1
        #Equa diff :
        ufppi1 = B1 * (e[i+1]
                       + B2 * ufppi
                       + B3 * ufpi
                       + B4 * uf[i])
        #Schema de Newmark :
        ufpi1 = (ufpi
                 + (1-gamma) * Te * ufppi
                 + gamma * Te * ufppi1)
        uf.append(uf[i]
                  + Te * ufpi
                  + Te * Te * (0.5-beta) * ufppi
                  + Te * Te * beta * ufppi1)
    return uf
```

La question est d'autant plus rapide que le code précédent est correctement présenté et écrit.

III.3 Détermination des pressions systolique, diastolique

Q26. On peut aussi regarder les pentes entre $i-1$ et i, et entre i et $i+1$.

```
def isMaxLoc(L):
    """Teste si L[1] est un maximum strict dans la liste L de longueur 3
       """
    return L[1] > max(L[0], L[2])

def isMinLoc(L):
    """Teste si L[1] est un minimum strict dans la liste L de longueur 3
       """
    return L[1] < min(L[0], L[2])

def pression_systolique(data):
    minLoc = data[0]
    maxLoc = data[0]
    for i in range(1, len(data)-1):
        if isMinLoc(data[i-1:i+2]):
            minLoc = data[i]
        if isMaxLoc(data[i-1:i+2]):
            maxLoc = data[i]
            if (maxLoc - minLoc) / minLoc > 4 * 10**(-5):
                return maxLoc
```

Q27. Attention, on demande l'instant où le critère est vérifié pour la dernière fois, ce qui n'est a priori pas la même chose que l'instant avant le premier instant où le critère n'est plus vérifié.
Avec les fonctions auxiliaires précédentes :

```
 pression_diastolique(data):
    minLoc = data[-1]
    maxLoc = data[-1]
```

```
    for i in range(len(data)-2, 0, -1):
        if isMinLoc(data[i-1:i+2]):
            minLoc = data[i]
        if isMaxLoc(data[i-1:i+2]):
            maxLoc = data[i]
        if maxLoc > 40 and (maxLoc - minLoc) / minLoc > 4 * 10**(-3):
            return maxLoc
```

IV Stockage et analyse des résultats

Q28. SELECT datetime, pdias, psyst, pouls FROM mesures WHERE (datetime >= time1) AND (datetime <= time2)

Q29. SELECT datetime, pdias, psyst, pouls FROM mesures WHERE (datetime >= time1) AND (datetime <= time2) AND (pid = id_patient) AND (type = 'tension')

Q30. On suppose que le résultat est stocké dans un tableau NumPy et on extrait les temps (la première colonne : toutes les lignes et la colonne 0) : `x = resultat_requete[:,0]` ; et ainsi de suite. Les bibliothèques NumPy et matplotlib sont respectivement chargées sous les abréviations `np` et `plt`.

```
x = resultat_requete[:, 0]
#Pression systolique, trait bleu plein, marqueur cercle
plt.plot(x, resultat_requete[:, 1],'bo-')
#Pression diastolique, trait pointillé rouge, marqueur étoile
plt.plot(x, resultat_requete[:, 2],'r*--')
#Pouls, trait mixte vert, marqueur diamant
plt.plot(x, resultat_requete[:, 3],'gD-.')
plt.xlabel('Temps (s)')
plt.legend(['P systolique','P diastolique','Pouls'])
plt.show()
```

Q31.

```
def analyse(valeurs):
    return (min(valeurs), max(valeurs), sum(valeurs)/len(valeurs))
```

Q32. Bon. Avec la version proposée à la question précédente, il n'y a aucune complexité à calculer. A la rigueur, $O(n)$ pour chaque valeur à calculer, vu qu'il faut au moins un (et un seul suffit) parcourt de la liste pour les calculer. On peut reprogrammer à la main maximum, minimum et moyenne :

```
def analyse(valeurs):
    mini = valeurs[0]
    maxi = valeurs[0]
    somme = 0
    for k in valeurs:
        if k > maxi:
            maxi = k
        if k < mini:
            mini = k
        somme += k
    moyenne = somme / len(valeurs)
    return mini, maxi, moyenne
```

Dans ce cas, il y a n passages dans la boucle, avec pour chaque passage dans la boucle exactement 2 comparaisons et une addition. En sortie de boucle, il y a une division, donc au total il y a $3n+1$ comparaisons et opérations, que ce soit dans le meilleur ou le pire cas.
Si l'énoncé nous avait demandé de compter les affectations, dans ce cas il y a une différence entre le meilleur et le pire cas. En ne comptant que les affectations, dans le cas d'une liste constante, il y a $2+n+1 = n+3$

affectations au total. Dans le cas d'une liste croissante (ou décroissante), il y a $2+2n+1 = 2n+3$ affectations au total.
Ces deux cas représentent les meilleurs et pires cas.

Q33. Un quicksort :

```
1  def quicksort(L):
2      if len(L) <= 1:
3          return L
4      else:
5          pivot = L[0]
6          plus_grand = []
7          plus_petit = []
8          for x in L[1:]:
9              if x >= pivot:
10                 plus_grand.append(x)
11             else:
12                 plus_petit.append(x)
13         return quicksort(plus_petit) + [pivot] + quicksort(plus_grand)
14
15 def mediane(valeurs):
16     return quicksort(valeurs)[len(valeurs)//2]
```

La médiane n'est pertinente que sur une grande liste de valeurs : distinguer les cas pairs et impairs ne sert à rien en pratique, à part dans des exercices scolaires artificiels.

Q34. À chaque passage dans la fonction, si la taille n de la liste est strictement plus grande que 1, il y a n comparaisons.
Dans le pire cas, la complexité du quicksort est en $O(n^2)$: c'est le cas par exemple d'un liste déjà triée, les listes `plus_petit` et `plus_grand` sont déséquilibrées, l'une étant vide et l'autre de taille $n-1$. L'appel récursif sur la liste vide se termine avec une seule comparaison et sans appel supplémentaire. En notant c_n le nombre de comparaisons pour une liste triée de taille n, on a

$$c_n = n + 1 + c_{n-1}$$

Donc $\boxed{c_n = \sum_{k=0}^{n} k + 1 = \frac{(n+1)(n+2)}{2} = O(n^2)}$
Dans le meilleur cas, chaque liste `plus_petit` et `plus_grand` est de taille $n/2$. Calculons la complexité a_n pour $n = 2^k$:

$$a_{2^k} = 2^k + 1 + 2a_{2^{k-1}} = 2^k + 1 + 2^k + 2 + 2^k + 2^2 + 2^3 a_{2^{k-3}}$$

Donc $a_{2^k} = \sum_{i=0}^{k}(2^k + 2^i) = (k+1)2^k + \frac{1-2^{k+1}}{1-2} = (k+1)2^k + 2^{k+1} - 1 = (k+3)2^k - 1 = O(k2^k)$
Or $k = \ln(n)/\ln(2) = O(\ln n)$, donc, dans le cas $n = 2^k$, la complexité dans le meilleur cas est en

$$O(n \ln n)$$

La méthode est efficace pour une liste qui n'est pas déjà triée : les meilleurs tris sont en $O(n \ln n)$. Par contre, si la liste est déjà triée ou presque, la méthode est inefficace (un tri par insertion vérifierai que la liste est triée en $O(n)$). On peut rendre le pivot aléatoire pour éviter que le pire cas soit celui, relativement fréquent, où la liste est déjà ordonnée ou presque ordonnée.

Q35. SELECT nom, prenom, telephone FROM patients WHERE id = (SELECT pid FROM mesures WHERE psyst>160 AND pdias>110 AND pouls>100 AND pouls<150)