BANQUE PT 2024 : CORRIGE DE LA PARTIE INFORMATIQUE DE L'EPREUVE D'INFORMATIQUE ET MODELISATION DE SYSTEMES PHYSIQUES

CONTROLE ENVIRONNEMENTAL D'UNE SALLE BLANCHE

A noter : en introduction du sujet, il est sous-entendu que des requêtes SQL seront demandées, ce qui n'est pourtant pas le cas par la suite…

C) Déterminer le nombre de FFU, leur répartition et ajustement de la vitesse de rotation

Q23 :

```
def volume(L, l, h):
    return L * l * h
```

Q24 : *Faut-il considérer n comme un entier, sachant que c'est un nombre de FFU ? Ca n'a pas l'air d'être l'esprit du sujet… Ici je considère que n n'est pas forcément entier.*

$(n\ Q)$ est le volume total renouvelé en une heure, par l'ensemble des FFU. Le volume de la salle blanche étant V, on a :

$$tx\_renew = \frac{n\ Q}{V}$$

Et donc :

$$n = \frac{tx\_renew \times V}{Q}$$

```
def nbre_ffu(V, Q, tx_renew):
    return tx_renew * V / Q
```

Q25 : *Le choix des termes « horizontale » et « verticale » est très mal choisi car a priori les dimensions L et l sont horizontales toutes les deux. Il aurait été préférable de parler de « longueur » et de « largeur ».*
L'énoncé est mal rédigé car il ne définit pas la notion de répartition équirépartie. La question elle-même est mal rédigée aussi car elle parle ***du*** *système alors qu'il y en a évidemment d'autres.*
A l'évidence, $n_x = n_y = 0$, $u = L$ et $v = l$ donne une solution mathématiquement, mais qui ne correspond sans doute pas au problème physique. La solution $n_x = n_y = 1$ convient également…
L'énoncé ne dit pas que le produit $(nx * ny)$ doit être maximal et d'ailleurs l'algorithme de Q26 ne cherche pas de solution maximale. Par exemple, pour n = 244, L = 50 m et l = 25 m, l'algorithme donne pour (nx, ny) le couple (24, 10) alors que le couple (22, 11) donne un nombre de FFU plus proche de n… Le problème est mal défini.

- Sur une « horizontale » de longueur L, il y a nx FFU (de longueur 1 m chacun), et $(nx + 1)$ espaces à gauche, à droite et entre les FFU (de longueur u chacun). On a donc :
$$L = (nx + 1) * u + nx * 1 = (nx + 1) * u + nx$$
- Sur une « verticale » de largeur l, il y a ny FFU (de longueur 1 m chacun), $(ny - 1)$ espaces entre les FFU (de longueur v chacun) et des espaces en haut et en bas (dont la somme des longueurs vaut $v + v$ ou $\frac{v}{2} + 1{,}5 * v$ selon la « verticale » considérée, c'est-à-dire $2 * v$ dans les deux cas). On a donc :
$$l = (ny - 1) * v + 2 * v + ny * 1 = (ny + 1) * v + ny$$
- Par ailleurs, nx et ny étant des entiers naturels, et le nombre de FFU n à prévoir (« *calculé précédemment* ») n'étant pas entier a priori (et si n est entier, il est peut-être premier), il est presque

illusoire d'avoir rigoureusement $n = (nx * ny)$. On s'impose donc $(nx * ny) < n$, et on augmentera légèrement le débit des ventilateurs comme expliqué ultérieurement dans l'énoncé (à la fin de la partie C). *Mais l'énoncé manque de clarté à ce stade.*

Remarque : on ne comprend pas bien pourquoi la troisième ligne est une inégalité stricte alors que $n_x * n_y = n$ semble être un cas envisageable (peu probable mais envisageable).

Q26 : *Là encore, la question pose problème. Il semble que l'initialisation se fasse avec $n_y = 1$, c'est-à-dire qu'il n'y a qu'une « ligne » de FFU, mais étant donné les décalages alternatifs des FFU, on ne peut pas dire qu'ils sont vraiment alignés.*

Si tous les FFU sont alignés sur une seule ligne, alors $ny = 1$ et nx est égal à la partie entière de n. On en déduit u et v grâce aux relations de la question précédente.

$nx = int(n)$
$ny = 1$
$u = (L - nx)/(nx + 1)$
$v = (l - ny)/(ny + 1)$

Q27 : Au départ, tous les FFU sont alignés sur une seule ligne, il est donc probable que u soit inférieur à 1 et que v soit positif. Au fur et à mesure des passages dans la boucle *while*, u va augmenter et v va diminuer. La condition d'arrêt correspond au moment où u deviendra plus grand que 1, d'où :

COND1 : $u <= 1$

A chaque fois, il faut recalculer u et v de la même manière que lors de la phase d'initialisation, d'où :

EXP2 : $(L - nx)/(nx + 1)$ et EXP3 : $(l - ny)/(ny + 1)$

On obtient finalement :

```
def repartition(L, l, n):
    nx = int(n)
    ny = 1
    u = (L - nx) / (nx + 1)
    v = (l - ny) / (ny + 1)
    while u <= 1:
        ny = ny + 1
        nx = n // ny
        u = (L - nx) / (nx + 1)
        v = (l - ny) / (ny + 1)
    return nx, ny, u, v
```

A noter que rien ne garantit dans ce programme que v reste positif…

A noter également les erreurs d'énoncé :

- *La fonction « repartition » ne doit pas retourner nx, ny, a, b (comme indiqué dans le script fourni), mais nx, ny, u, v.*
- *La fonction « repartition » a pour paramètres d'appel (L, l, n) ou (l, L, n) selon les endroits dans l'énoncé.*

D) Etude de résultats expérimentaux afin de modéliser le couple résistant ramené sur l'arbre moteur

1) Extraction des résultats expérimentaux d'un essai

A noter de nombreuses incohérences dans l'introduction de cette partie :

- *Ligne 3 : fréquence d'échantillonnage à 1000 Hz ou à 10 Hz (variable selon les endroits).*
- *Ligne 11 : 360 points ou 365 points (variable selon les endroits).*

- *Ligne 12 : 10 Hz = 0,1 s : c'est moche d'écrire ça !*
- *La 5ème ligne de données devrait commencer par 0,4 et pas par 4.*

Q28 :
En parcourant la chaîne de caractères par compréhension :

```
def remplace_virgule_par_point(chaine):
    new_chaine = ""
    for c in chaine:
        if c == ",":
            new_chaine += "."
        else:
            new_chaine += c
    return new_chaine
```

En parcourant la chaîne de caractères par les indices :

```
def remplace_virgule_par_point(chaine):
    new_chaine = ""
    for i in range(len(chaine)):
        if chaine[i] == ",":
            new_chaine += "."
        else:
            new_chaine += chaine[i]
    return new_chaine
```

Q29 : *Autre problème : la fonction « extraction_resultats_essai » a pour paramètre d'appel « chemin_fichier » ou « nom_fichier » (variable selon les endroits). De plus, la dernière ligne est mal indentée.*

ARGS1 : $ligne$
ARGS2 : $float(liste[0])$
ARGS3 : $float(liste[1])$
ARGS4 : $float(liste[2])$

A noter que ARGS5 n'est pas demandé : ARGS5 : $float(liste[3])$ *(le* `"\n"` *ne gêne pas pour la conversion).*

Q30 : *On ne comprend pas bien comment* ***compléter*** *la fonction « extraction_resultats_essai » avec un « return dico » alors qu'elle a déjà un « return ». Il faut donc comprendre qu'on ne complète pas la fonction « extraction_resultats_essai », mais qu'on* ***remplace*** *le dernier « return » par la nouvelle suite d'instructions.*

Plutôt que de renvoyer les 4 listes, on définit le dictionnaire `dico` et on stocke ces 4 listes dans le dictionnaire (ce sont les *valeurs*). Les *clés* associées (qui sont uniques) sont les chaînes de caractères `"temps"`, `"consigne"`, `"vitesse"` et `"courant"` *(première instruction)*.
Et la fonction renvoie le dictionnaire ainsi créé *(deuxième instruction)*.
L'intérêt d'un dictionnaire est que l'on accès aux valeurs via les clés, avec l'instruction `dico[clé]`. Ainsi, vu que les noms des clés sont explicites ici, on n'a pas à se préoccuper de l'ordre dans lequel la fonction *extraction_resultats_essai* renverrait les 4 listes (on accède aux différentes mesures par leur nom au lieu d'un indice).
Mais on n'utilise pas les spécificités d'un dictionnaire ici, l'intérêt d'un dictionnaire est en fait assez limité dans cette situation.

Q31 : *Y a-t-il à nouveau une erreur d'énoncé ? Faut-il s'intéresser à l'essai 1 ou à l'essai 11 ?*

```
donnees = extraction_resultats_essai("essai1.txt")
print(donnees["temps"])
```

Q32 : *La bibliothèque importée est bien sûr « matplotlib » et pas « matplot ».*

```
def affichage(donnees):
    plt.figure()                                    # crée une fenêtre de tracé vide
    plt.plot(donnees["temps"], donnees["vitesse"])  # tracé de la courbe
    plt.show()                                      # affichage de la figure
```

Ou bien si on veut embellir le graphe avec des titres, légendes, … (pas demandé ici)

```
def affichage(donnees):
    plt.figure()                                    # crée une fenêtre de tracé vide
    plt.plot(donnees["temps"], donnees["vitesse"])  # tracé de la courbe
    plt.xlabel("temps (s)")                         # légende de l'axe des abscisses
    plt.ylabel("vitesse de rotation (tr/min)")      # légende de l'axe des ordonnées
    plt.legend("vitesse réelle")                    # légende
    plt.grid()                                      # présence de la grille de fond
    plt.show()                                      # affichage de la figure
```

2) Détermination du couple résistant

Q33 :

```
def moyenne_sur_intervalle(Liste, deb, fin):
    somme = 0
    for i in range(deb, fin):
        somme += Liste[i]
    return somme / (fin - deb)
```

Ou bien, en utilisant la fonction « sum » :

```
def moyenne_sur_intervalle(Liste, deb, fin):
    return sum(Liste[deb: fin]) / (fin - deb)
```

Q34 :

```
def point_de_fonctionnement(donnees):
    N = moyenne_sur_intervalle(donnees["vitesse"], 500, 701)
    i = moyenne_sur_intervalle(donnees["courant"], 500, 701)
    Cr = 0.13 * i
    return N, Cr
```

3) Analyse de l'ensemble des essais

Q35 :

```
L_vit = []
L_Cr = []
for i in range(1, 19):
    donnees = extraction_resultats_essai("essai" + str(i) + ".txt")
    L_vit.append(point_de_fonctionnement(donnees)[0])
    L_Cr.append(point_de_fonctionnement(donnees)[1])
```

4) Modélisation du comportement du couple résistant Cr en fonction de la vitesse du moteur N par une régression linéaire

Q36 : La méthode de descente de gradient est une méthode d'apprentissage supervisé car la détermination des coefficients se base sur des données fournies :

- C'est une méthode d'apprentissage automatique car des approches mathématiques et statistiques donnent aux ordinateurs la capacité d' « apprendre » à partir de données.

- C'est une méthode d'apprentissage supervisé car elle consiste à apprendre une fonction de prédiction à partir d'exemples annotés.

On distingue les problèmes avec classification et avec régression :

- Les problèmes de prédiction d'une variable qualitative sont des problèmes de classification : un apprentissage supervisé avec classification permet d'étiqueter des données, de les ranger dans des catégories (par exemple les k plus proches voisins).

- Les problèmes de prédiction d'une variable quantitative sont des problèmes de régression : un apprentissage supervisé avec régression a pour but de trouver une relation entre différentes quantités.

Q37 : Une régression linéaire multiple est une méthode de régression mathématique étendant la régression linéaire simple pour décrire les variations d'une variable endogène (dépendante) associée aux variations de plusieurs variables exogènes (indépendantes).

Dans une régression linéaire multiple, on cherche à exprimer une quantité comme somme d'une constante et de multiples de grandeurs données.

Le modèle de comportement de Cr à rechercher serait de considérer plusieurs paramètres comme la vitesse de rotation au carré, la température, l'humidité, le nombre d'heures d'utilisation...

$$C_r = a_0 + a_1\omega^2 + a_2T + a_3H + \cdots$$

Q38 :

```
m = len(V_N2)
X = np.hstack((V_N2, np.ones([m, 1])))
```

Q39 :

```
def modele(X, theta):
    return np.dot(X, theta)
```

Q40 :

$$Q(a_0, a_1) = \sum_{k=1}^{m} \left(\left(a_1 \times x^{(k)} + a_0 \right) - y^{(k)} \right)^2$$

Q41 :

$$X.\theta - Y = \begin{bmatrix} \left(a_1 \times x^{(1)} + a_0 \right) - y^{(1)} \\ \vdots \\ \left(a_1 \times x^{(m)} + a_0 \right) - y^{(m)} \end{bmatrix}$$

$$(X.\theta - Y)^T = \begin{bmatrix} \left(a_1 \times x^{(1)} + a_0 \right) - y^{(1)} & \dots & \left(a_1 \times x^{(m)} + a_0 \right) - y^{(m)} \end{bmatrix}$$

$$(X.\theta - Y)^T.(X.\theta - Y) = \left[\sum_{k=1}^{m} \left(\left(a_1 \times x^{(k)} + a_0 \right) - y^{(k)} \right)^2 \right]$$

Le coût $Q(\theta)$ est le contenu de la matrice (1×1) suivante (ou bien directement l'expression suivante par abus de notation) :

$$(X.\theta - Y)^T.(X.\theta - Y)$$

Q42 : Première possibilité (qui ne semble pas dans l'esprit de l'énoncé) : sachant que l'opérateur * réalise un produit terme à terme :

```
def cout(X, Y, theta):
    A = np.dot(X, theta) - Y
    return np.sum(A * A)
```

Autres possibilités :

Si on manipule des tableaux Numpy de taille $(m, 1)$ (tableaux 2D) :

Tests introductifs :

```
>>> A=np.array([[1],[2],[3]])
>>> A.shape
(3, 1)
>>> B=np.transpose(A)
>>> B.shape
(1, 3)
>>> C=np.dot(B,A)
>>> C
array([[14]])
>>> type(C)
<class 'numpy.ndarray'>
>>> D=np.sum(C)
>>> D
14
>>> type(D)
<class 'numpy.int32'>
```

```
def cout(X, Y, theta):
    A = np.dot(X, theta) - Y
    B = np.dot(np.transpose(A), A)
    return np.sum(B)
```

La dernière somme sert à convertir la matrice B de taille (1×1) *en un nombre. On aurait aussi pu forcer la conversion de la matrice B de taille* (1×1) *en un nombre par l'instruction « float(B) ».*

Si on manipule des tableaux Numpy de taille $(m,)$ (tableaux 1D, « vecteurs ») :

Tests introductifs :

```
>>> A=np.array([1,2,3])
>>> A.shape
(3,)
>>> B=np.transpose(A)
>>> B.shape
(3,)
>>> C=np.dot(B,A)
>>> C
14
>>> type(C)
<class 'numpy.int32'>
>>> D=np.dot(A,A)
>>> D
14
```

```
def cout(X, Y, theta):
    A = np.dot(X, theta) - Y
    return np.dot(A, A)
```

Il est alors inutile de transposer A et de faire la somme finale.

Q43 :

$$\frac{\partial Q}{\partial a_1} = 2\sum_{k=1}^{m}\left(\left(\left(a_1 \times x^{(k)} + a_0\right) - y^{(k)}\right) \times x^{(k)}\right)$$

$$\frac{\partial Q}{\partial a_0} = 2\sum_{k=1}^{m}\left(\left(a_1 \times x^{(k)} + a_0\right) - y^{(k)}\right)$$

Q44 : *Là aussi, l'énoncé laisse à désirer… Il aurait été préférable d'écrire* $Q(a_1, a_0)$ *au lieu de* $Q(a_0, a_1)$ *pour coller à la suite…*

$$\overrightarrow{grad}(Q) = \begin{bmatrix} \frac{\partial Q}{\partial a_1} \\ \frac{\partial Q}{\partial a_0} \end{bmatrix} = 2\begin{bmatrix} \sum_{k=1}^{m}\left(\left(\left(a_1 \times x^{(k)} + a_0\right) - y^{(k)}\right) \times x^{(k)}\right) \\ \sum_{k=1}^{m}\left(\left(a_1 \times x^{(k)} + a_0\right) - y^{(k)}\right) \end{bmatrix}$$

$$X.\theta - Y = \begin{bmatrix} \left(a_1 \times x^{(1)} + a_0\right) - y^{(1)} \\ \vdots \\ \left(a_1 \times x^{(m)} + a_0\right) - y^{(m)} \end{bmatrix}$$

$$X = \begin{bmatrix} x^{(1)} & 1 \\ \vdots & \vdots \\ x^{(m)} & 1 \end{bmatrix}$$

$$X^T = \begin{bmatrix} x^{(1)} & \dots & x^{(m)} \\ 1 & \dots & 1 \end{bmatrix}$$

Et donc :

$$\overrightarrow{grad}(Q) = 2.X^T.(X.\theta - Y)$$

Q45 :

```
def gradient(X, Y, theta):
    A = np.dot(X, theta) - Y
    return 2 * np.dot(np.transpose(X), A)
```

Q46 :

```
def descente_gradient(X, Y, theta_init, alpha, n):
    theta = theta_init
    for k in range(n):
        grad = gradient(X, Y, theta)
        theta[0] -= alpha * grad[1]
        theta[1] -= alpha * grad[0]
    return theta
```

Ou bien, sachant qu'avec Numpy on peut calculer directement sur les vecteurs :

```
def descente_gradient(X, Y, theta_init, alpha, n):
    theta = theta_init
    for k in range(n):
        theta -= alpha * gradient(X, Y, theta)
    return theta
```

E) Etude du comportement du moteur électrique entraînant le ventilateur

1) Etude du régime établi : recherche de la tension de commande moteur

Q47 :

```
def affichage(a0, a1):
    omega = [(10 * i) for i in range(26)]
    Cr = [(a1 * omega[i]**2 + a0) for i in range(len(omega))]
    plt.figure()                                # crée une fenêtre de tracé vide
    plt.plot(omega, Cr)                         # tracé de la courbe
    plt.title("Graphe de Cr")                   # titre du graphe
    plt.xlabel("vitesse de rotation (rad/s)")   # légende de l'axe des abscisses
    plt.ylabel("couple résistant (N.m)")        # légende de l'axe des ordonnées
    plt.legend("Couple résistant")              # légende
    plt.grid()                                  # présence de la grille de fond
    plt.show()                                  # affichage de la figure
```

Ou bien, si on veut faire plus simple (sans les titres, légendes …), et en utilisant des tableaux Numpy :

```
def affichage(a0, a1):
    omega = np.arange(0, 260, 10)               # borne supérieure exclue
    Cr = a1 * omega**2 + a0
    plt.figure()                                # crée une fenêtre de tracé vide
    plt.plot(omega, Cr)                         # tracé de la courbe
    plt.show()                                  # affichage de la figure
```

Q48 : *La question est peu claire parce que l'on ne retourne pas que omega, mais également U (qui est un paramètre d'appel…). De plus, on suppose qu'il n'y a qu'une seule solution dans l'intervalle [deb, fin] car sinon la méthode de dichotomie peut ne pas marcher.*

```
def dichotomie(U, deb, fin, e):
    while (fin - deb) > e:
        milieu = (deb + fin) / 2
        if f(milieu, U) * f(fin, U) < 0:
            deb = milieu
        else:
            fin = milieu
    return (deb + fin) / 2, U
```

2) Etude en régime dynamique : prévision des caractéristiques de la réponse du FFU à un échelon de courant

L'introduction de cette partie est peu claire, car l'énoncé a dû être tronqué.

Q49 :

$$\frac{d\omega}{dt} = \frac{k.I - b - a.\omega(t)^2}{J} = g(\omega(t), t)$$

Q50 : Formule de Taylor-Young à l'ordre 1 :

$$\omega(t+T) = \omega(t) + T.\frac{d\omega}{dt} + o(T)$$

D'où l'expression proposée.

Q51 :

$$\omega(t+T) \approx \omega(t) + T.\frac{d\omega}{dt} \approx \omega(t) + T.g(\omega(t), t)$$

Pour la fin de l'énoncé, les notations Lomega ou Lw, ainsi que omega0 ou w0, auraient été préférables (notations sans l'alphabet grec !).

Q52 : I*l semble que t0 = 0 d'après l'énoncé ? Manque de clarté…*

```
def Euler_Explicite(g, w0, t0, t1, T):
    LT = []
    Lw = []
    w = w0
    t = t0
    while t <= t1:
        LT.append(t)
        Lw.append(w)
        w += T * g(w, t)
        t += T
    return LT, Lw
```

Q53 : *Liberté est laissée au candidat de définir le temps de réponse.*
On considère la courbe proposée et on définit ici le temps de réponse comme la durée nécessaire pour que la vitesse de rotation ait évolué de 99 % (par exemple) entre sa valeur initiale et sa valeur en régime permanent.

```
omega_permanent = Lw[-1]

i = 0
while Lw[i] < w0 + 0.99 * (omega_permanent - w0):
    i += 1
temps_de_reponse = LT[i] - LT[0]    # t0 est-il nécessairement nul ? Pas clair.

print("le temps de réponse est :", temps_de_reponse)
print("la vitesse de rotation en régime permanent est :", omega_permanent)
```