teaching sciences for innovation

Proposition de corrigé

Concours : Concours Commun Polytechniques

Année : 2018

Filière : TSI

Épreuve : Informatique

Ceci est une proposition de corrigé des concours de CPGE, réalisée bénévolement par des enseignants de Sciences Industrielles de l'Ingénieur et d'Informatique, membres de l'UPSTI (Union des Professeurs de Sciences et Techniques Industrielles), et publiée sur le site de l'association :

https://www.upsti.fr/espace-etudiants/annales-de-concours

A l'attention des étudiants

Ce document vous apportera des éléments de corrections pour le sujet traité, mais n'est ni un corrigé officiel du concours, ni un corrigé détaillé ou exhaustif de l'épreuve en question.

L'UPSTI ne répondra pas directement aux questions que peuvent soulever ces corrigés : nous vous invitons à vous rapprocher de vos enseignants si vous souhaitez des compléments d'information, et à vous adresser à eux pour nous faire remonter vos éventuelles remarques.

Licence et Copyright

Informez-vous !

Retrouvez plus d'information sur les Sciences de l'Ingénieur, l'orientation, les Grandes Ecoles ainsi que sur les Olympiades de Sciences de l'Ingénieur et sur les Sciences de l'Ingénieur au Féminin sur notre site : www.upsti.fr

L'équipe UPSTI

CORRECTION INFORMATIQUE
CCP TSI 2018

PARTIE 1 : PROBLEMATIQUE

```python
from numpy import min, max, zeros, roots, real
from scipy import linspace
from scipy.signal import lti, step
import matplotlib.pyplot as plt
from random import random
import os
import sqlite3, numpy as np
```

Question 1:

```python
numG=[8]
denG=[0.0072,0.273,1.13,1]
G=lti(numG,denG)
```

PARTIE 2 : CORRECTION DU SYSTEME

Question 2: On réduit au même dénominateur

$$C(p) = K_p \frac{T_i p + 1 + T_d T_i p^2}{T_i p} = \frac{K_p + K_p T_i p + K_p T_d T_i p^2}{T_i p}$$

```python
def correcteur(Kp,Ti,Td):
    num=[Kp*Td*Ti,Kp*Ti,Kp]
    den=[Ti,0]
    return(num,den)

numC,denC=corrrecteur(Kp,Ti,Td)
```

Question 3: Multiplication de listes

```python
def multi_listes(P,Q):
    deg_max=(len(P)-1)+(len(Q)-1)
    P1=zeros(deg_max+1)
    Q1=zeros(deg_max+1)
    for j in range(len(P)):
        P1[j]=P[j]
    for j in range(len(Q)):
        Q1[j]=Q[j]
    R=zeros(deg_max+1)
    for k in range(deg_max+1):
        for i in range(k+1):
            R[k]=R[k]+P1[i]*Q1[k-i]
    return(R)
```

Cette fonction permet de calculer les coefficients du produit de 2 polynômes et les stocke dans la variable `R`. Si les 2 polynômes d'entrées `P` et `Q` sont donnés par ordre décroissants de puissances, la sortie est une liste des coefficients par ordre décroissants des puissances.

Avant la première boucle for la liste Q1 vaut: `Q1=[0,0,0,0]`

Pour k=2 et i=1 on a : `R=[a.r,a.s+b.r,c.r,0]`

A la fin de la fonction : `R=[a.r,a.s+b.r,c.r+b.s,c.s]`

Question 4: Fonction inverse :

```
def inverse(liste):
    taille=len(liste)
    liste_r=[]
    for k in range(taille):
        liste_r.append(liste[taille-1-k])
    return(liste_r)
```

Question 5: Fonction multi_FT

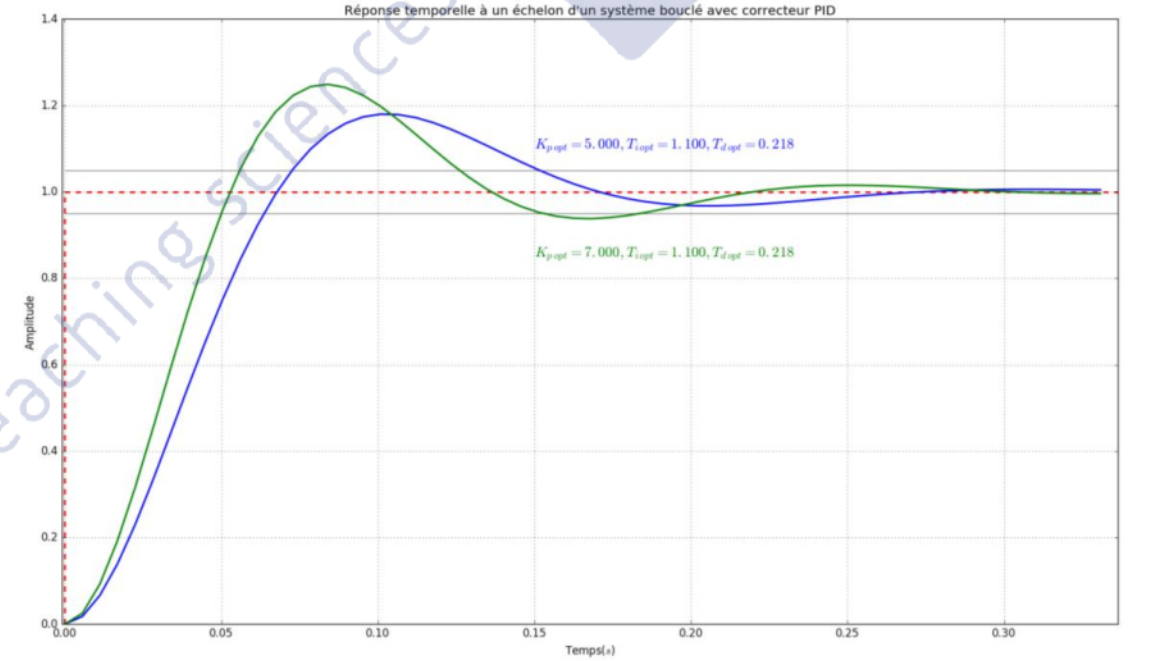
```
def multi_FT(num1,den1,num2,den2):
    num=multi_listes(num1,num2)
    den=multi_listes(den1,den2)
    return(num,den)
```

Contrairement à ce qui est dit dans l'énoncé il est inutile d'utiliser la fonction inverse.

Question 6: Somme de deux polynômes

```
def somme_poly(P,Q):
    temp=zeros(max([len(P),len(Q)]))
    if len(P)<len(Q):
        temp[len(Q)-len(P):]=P
        somme=temp+Q #pris comme somme de 2 arrays car temp est un array
    else:
        temp[len(P)-len(Q):]=Q
        somme=temp+P #pris comme somme de 2 arrays car temp est un array
    return(somme)
```

Remarque: Le tracé figure 2 de l'énoncé ne correspond pas aux valeurs des coefficients du correcteur données. Ce graphe correspond à un gain proportionnel de l'ordre de 7 et non 5. On donne ci-dessous les graphes pour ces deux séries de coefficients :



PARTIE 3: DETERMINATION DES CRITERES D'OPTIMISATION

Question 7: Stabilité

```python
def stabilite(P):
    rep=True
    racines=roots(P)
    for k in range(len(racines)):
        if real(racines[k])>0:
            rep=False
    return(rep)
```

Autre solution sans parcourir toutes les racines:

```python
def stabilite(P):
    rep=True
    racines=roots(P)
    k=0
    while rep==True and k<len(racines):
        if real(racines[k])>0:
            rep=False
        k=k+1
    return(rep)
```

Question 8: Temps de réponse

```python
def temps_reponse(s,t):
    T5=0
    s_fin=s[-1]
    for tt in range(len(s)):
        j=len(t)-tt-1
        if ((s[j]>s_fin*.95 and s[j-1]<s_fin*.95) or (s[j]<s_fin*1.05 and
        s[j-1]>s_fin*1.05)):
            T5=t[j]
            break
    return(T5)
```

On parcourt la liste des réponses dans l'ordre inverse. La sortie T5 est un majorant de $t_{r5\%}$ car T5 est telle que s[T5]>0,95s_fin ou s[T5]<1.05s_fin. Il est inutile de tester `s[j]` et `s[j-1]`.

Question 9: Autre écriture :

```python
def temps_reponse2(s,t):
    s_fin=s[-1]
    j=len(t)-1
    while (s[j]>s_fin*.95 and s[j]<s_fin*1.05):
        j=j-1
    return(t[j+1])
```

Question 10: Dépassement

```python
def depassement(s,t):
    s_fin=s[-1]
    return((max(s)-s_fin)/s_fin)
```

Question 11: Critère de la valeur absolue de l'erreur : on propose deux solutions méthodes des rectangles (`critere_IAE_rect`) et méthode des trapèzes (`critere_IAE_trap`)

```python
def critere_IAE_rect(s,t):
    IAE=0
    for k in range(1,len(t)):
        IAE+=abs(1-s[k])*(t[k]-t[k-1])
    return(IAE)
```

```python
def critere_IAE_trap(s,t):
    IAE =0
    for k in range(1,len(t)):
        IAE +=(abs(1-s[k])+abs(1-s[k-1]))*(t[k]-t[k-1])/2
    return(IAE)
```

Question 12: La fonction qui pondère le poids de chaque critère renvoie 1 pour les valeurs T_{50}, D_{10} et IAE_0. Pour qu'une configuration soit meilleure que la solution de référence, il faut que la fonction renvoie une valeur inférieure à 1.

Question 13: Coefficients du correcteur

```python
def calcul_coef_correcteur(Ip,Ii,Id):
    Kpmin,Kpmax=.01,100
    Timin,Timax=.01,100
    Tdmin,Tdmax=0,50
    Kp=(Kpmax-Kpmin)*Ip/(2**16-1)+Kpmin
    Ti=(Timax-Timin)*Ii/(2**16-1)+Timin
    Td=(Tdmax-Tdmin)*Id/(2**16-1)+Tdmin
    return(Kp,Ti,Td)
```

Le nombre de combinaisons possibles pour le triplet est: $nb = 2^{16} \cdot 2^{16} \cdot 2^{16} = 2^{48}$

Question 14: Calcul du cout d'une combinaison

```python
def calcul_cout(Ip,Ii,Id):
    Kp,Ti,Td=calcul_coef_correcteur(Ip,Ii,Id)
    numC,denC=correcteur(Kp,Ti,Td)
    num_BO,den_BO=multi_FT(numG,denG,numC,denC)
    numBF,den_BF=FTBF(num_BO,den_BO)
    stable=stabilite(den_BF)
    if stable:
        temps_BF,sol_BF=rep_Temp(Kp,Ti,Td)
        Tr=temps_reponse(sol_BF,temps_BF)
        D=depassement(sol_BF,temps_BF)
        IAE=critere_IAE_trap(sol_BF,temps_BF)
        cout=ponderation_cout(Tr,D,IAE)
    else:
        cout=100
    return(cout)
```

Question 15: Tester toutes les combinaisons possibles n'est pas envisageable.

Il faudrait $(2^{16})^3 \cdot 0{,}0103 \approx 2{,}89 \cdot 10^{12} s \approx 92 \cdot 10^3 ans$

Sans calculatrice : $2^{48} = 2^8 \cdot 2^{40} \approx 256 \cdot 10^{12}$

PARTIE 4 : RESOLUTION PAR UN ALGORITHME GENETIQUE

Question 16: Le fonction genererGene(n) permet de générer une possibilité de gène sous la forme d'une chaîne de n caractères aléatoires composée de 0 et de 1. On initialise donc n à 16.

```python
n=16
def genererGene(n):
    b2=''
    for i in range(n):
        b2=b2+str(int(random()*2))
    return(b2)
```

Question 17: Initialisation d'une population initiale

```python
def generer_liste_initiale(n):
    CandidatS=[]
    for i in range(100):
        Candidat=[genererGene(n),genererGene(n),genererGene(n)]
        CandidatS.append(Candidat)
    return(CandidatS)
```

Question 18: On a: $(1001001000000000)_2 = (37376)_{10}$

Question 19: Passage binaire decimal

```python
def decodage(b2):
    b10=0
    for k in range(len(b2)):
        b10+=int(b2[len(b2)-1-k])*2**k
    return(b10)
```

Question 20: La fonction Tri utilise le tri par insertion dont la complexité est en O(n) au meilleur des cas et en $O(n^2)$ au pire des cas avec n=len(L).

Question 21: Croisement

```python
def croisement(P1,P2):
    E1,E2=[],[]
    for i in range(3):
        E1.append(P1[i][0:4]+P2[i][4:12]+P1[i][12:])
        E2.append(P2[i][0:4]+P1[i][4:12]+P2[i][12:])
    return(E1,E2)
```

Question 22: Pour le candidat ['1111111111111111','0000000000000000','1111111100000000'] la fonction mutation(Candidat,1,12) renvoie le candidat ['1111111111111111','0000000000001000','1111111100000000']

Question 23: Nouvelle génération

```python
def nouvGeneration(L):
    L_new=L.copy() #Attention PB Enoncé sinon CandidatS_top modifié à
    la fin de la fonction
    for i in range(10):
        E1,E2=croisement(L[0],L[int(random()*19)+1])
        L_new.append(mutation(E1,int(random()*3),int(random()*16)))
        L_new.append(mutation(E2,int(random()*3),int(random()*16)))
    for i in range(30):
        E1,E2=croisement(L[int(random()*19+1)],L[int(random()*19+1)])
        L_new.append(mutation(E1,int(random()*3),int(random()*16)))
        L_new.append(mutation(E2,int(random()*3),int(random()*16)))
    return(L_new)
```

Question 24: Gestion des doublons :

```python
def doublons(L):
    for k in range(len(L)-1):
        for j in range(k+1,len(L)):
            if L[j]==L[k]:
                L[j]=[genererGene(n),genererGene(n),genererGene(n)]
    return(L)
```

PARTIE 5 : BASE DE DONNEES

Les réponses à cette partie sont basées sur l'extrait de la base de données de l'énoncé. Il est étrange que lors de simulation informatique, la base générée par nos simulations ne donne pas du tout le même résultat.

Question 25: Cette requête permet de déterminer le coût pondéré du meilleur candidat issu de l'algorithme génétique. Ici 0.618248479198

Question 26: Longévité

```sql
SELECT disparition-apparition AS longevite
FROM Historique
WHERE score=(SELECT min(score) FROM Historique)
```

Cette requête renvoie 32.

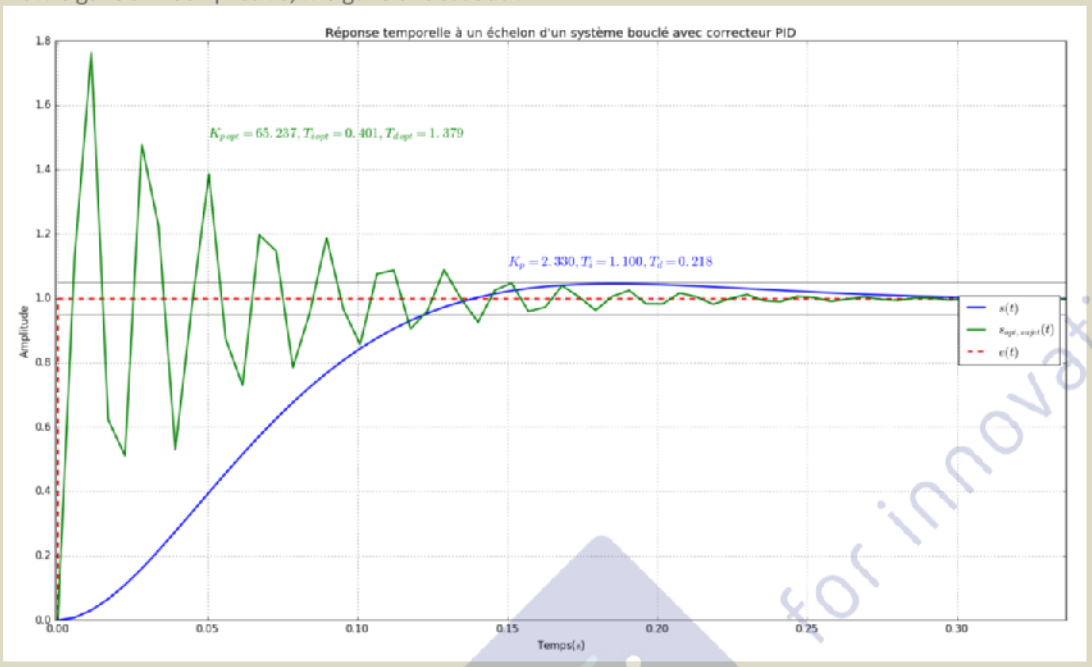
Question 27: Cette requête détermine les identifiants des candidats présents à l'itération 15. Elle renvoie 46 et 50 d'après l'extrait de la base de données de l'énoncé.

Question 28: Valeur moyenne des gains du PID :

```sql
SELECT AVG(gene_Kp) AS moyKp,AVG(gene_Ti) AS moyTi,AVG(gene_Td) AS moyTd
FROM Historique
WHERE apparition<19 AND disparition>20
```

Question 29: On constate que l'algorithme génétique permet d'obtenir un réglage du correcteur meilleur que la méthode de compensation de pôles. En effet le temps de réponse est plus faible (environ 0,1 s contre 0,14 s), le premier dépassement est plus faible (2% contre 5%) et l'intégrale de la valeur absolue de l'erreur semble plus faible également. Pour cet exemple une cinquantaine d'itérations semblent nécessaires ce qui correspond à un temps de calcul a priori réduit. A titre de comparaison

Remarque: Une simulation par matlab ou python du meilleur candidat du sujet donne un système très oscillant contredisant la figure 7. Pour preuve, la figure ci-dessous :



Cela est confirmé par le calcul du « coût » de cette solution qui donne 5,67492 et non pas la valeur donnée par l'énoncé de 0,61824.

Avec une simulation de l'algorithme complet nous trouvons un correcteur optimal qui prend les valeurs :

$$K_p = 92{,}16; T_i = 0{,}042\ s; T_d = 48{,}637\ s$$

Le « coût » de ce correcteur est de 0,567462.

Les résultats de notre simulation sont donnés dans le fichier « BDD.db ». Nous avons fait une simulation sur 200 cycles et l'algorithme a convergé en 23 itérations (au sens où l'optimal est apparu à la 23ème itérations).