CONCOURS COMMUNS POLYTECHNIQUES

ÉPREUVE SPÉCIFIQUE - FILIÈRE TSI

INFORMATIQUE

Jeudi 3 mai : 14 h - 17 h

N.B. : le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction. Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

Les calculatrices sont interdites

Le sujet est composé de 6 parties qui peuvent être traitées indépendamment.
Il est néanmoins recommandé de les aborder dans l'ordre.

Le sujet comporte :
- le texte du sujet, 13 pages,
- l'annexe, 3 pages,
- le document-réponse à rendre, 12 pages.

Dans tout le sujet, on considérera à chaque étape que les fonctions définies lors des questions précédentes sont utilisables pour la suite du sujet.

Vous devez répondre directement sur le document réponse, soit à l'emplacement prévu pour la réponse lorsque celle-ci implique une rédaction, soit en complétant les différents programmes en langage Python. Si vous manquez de place dans les cadres prévus, vous pouvez compléter sur la dernière page en précisant la question.

OPTIMISATION D'UN CORRECTEUR P.I.D. PAR UN ALGORITHME GÉNÉTIQUE

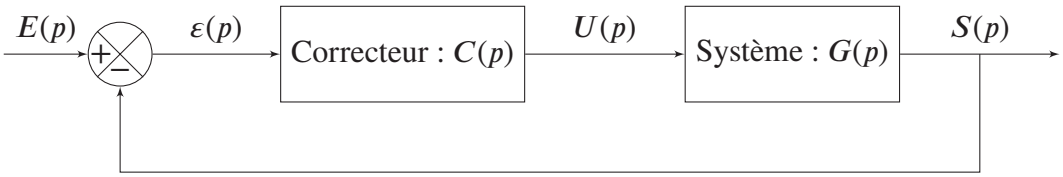
Partie I - Problématique

La correction d'un système asservi linéaire, en vue de répondre aux exigences de précision, rapidité et d'oscillations, n'est jamais aisée et est souvent réalisée à partir de méthodes pratiques plus ou moins empiriques.

Le correcteur Proportionnel, Dérivateur, Intégrateur (P.I.D.) est fréquemment employé afin d'améliorer le comportement des systèmes asservis. On se propose de mettre en œuvre une procédure d'optimisation du correcteur P.I.D. basée sur les algorithmes génétiques.

I.1 - Présentation du problème

On considère le système asservi à retour unitaire décrit par le schéma-bloc ci-dessous :



- on note p la variable de Laplace, les grandeurs seront notées en minuscule dans le domaine temporel et en majuscule dans le domaine de Laplace tel que $\mathscr{L}(e(t)) = E(p)$;
- le correcteur est un correcteur P.I.D. (parfait) de fonction de transfert $C(p)$;
- le système à réguler est connu par sa fonction de transfert $G(p) = \dfrac{N_G(p)}{D_G(p)}$ où $N_G(p)$ et $D_G(p)$ sont deux polynômes avec $\deg(N_G(p)) = k < \deg(D_G(p)) = \ell$. On pose :
  - $N_G(p) = n_0 + n_1 \cdot p + n_2 \cdot p^2 + \cdots + n_k \cdot p^k$,
  - $D_G(p) = d_0 + d_1 \cdot p + d_2 \cdot p^2 + \cdots + d_\ell \cdot p^\ell$.

Afin de déterminer la réponse temporelle du système corrigé, nous avons besoin de définir la fonction de transfert en boucle ouverte $BO(p)$, puis la fonction de transfert en boucle fermée $BF(p)$.

On rappelle que dans le cadre d'un système asservi à retour unitaire, on a :

$$BO(p) = C(p) \cdot G(p) = \frac{N_O(p)}{D_O(p)} \text{ et } BF(p) = \frac{BO(p)}{1 + BO(p)} = \frac{N_F(p)}{D_F(p)} = \frac{N_O(p)}{N_O(p) + D_O(p)}.$$

On a alors

$$S(p) = BF(p) \cdot E(p)$$

et la réponse temporelle $s(t)$ s'obtient en recherchant la transformée inverse de Laplace de $S(p)$.

Pour réaliser cette étude, nous allons utiliser la librairie **scipy.signal** qui permet, à partir de la fonction de transfert d'un système :

- d'obtenir la réponse temporelle à un échelon unitaire ;
- d'obtenir les racines d'un polynôme.

L'**annexe A.1** des pages 14 et 15, montre un exemple d'utilisation et précise les quelques commandes utiles de cette librairie dans le cadre de ce sujet.

I.2 - Système à réguler

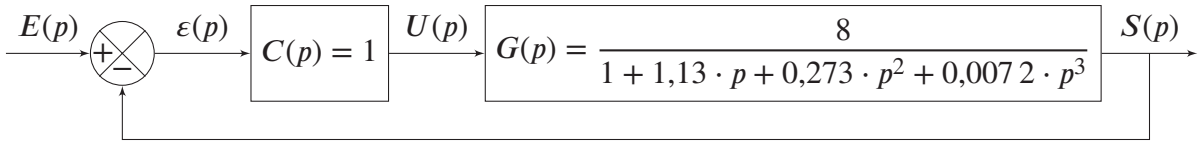
Le système que nous allons étudier est décrit par la fonction de transfert :

$$G(p) = \frac{8}{1 + 1{,}13 \cdot p + 0{,}273 \cdot p^2 + 0{,}0072 \cdot p^3} .$$

On note **numG** et **denG** les listes associées au numérateur $N_G(p)$ et au dénominateur $D_G(p)$ de la fonction de transfert $G(p)$.

Q1. À l'aide de l'**annexe A.1**, page 14 et de l'expression de $G(p)$, compléter les lignes 1 et 3 du **listing de la page DR 1 du document réponse.**

Afin d'optimiser la réponse temporelle on réalise un asservissement. Le schéma-bloc ci-dessous représente cet asservissement.



Pour ce premier essai, on considère $C(p) = 1$. La réponse temporelle $s(t)$ à un échelon unitaire $e(t)$ de ce système est tracée sur la **figure 1**.

On constate que la réponse temporelle du système asservi (**figure 1**) présente à la fois un défaut de précision (la valeur de consigne n'est pas atteinte), un temps de réponse d'environ 1,8 s, des oscillations et un premier dépassement relativement important.

Il semble nécessaire de choisir un correcteur qui améliore notablement la réponse du système.

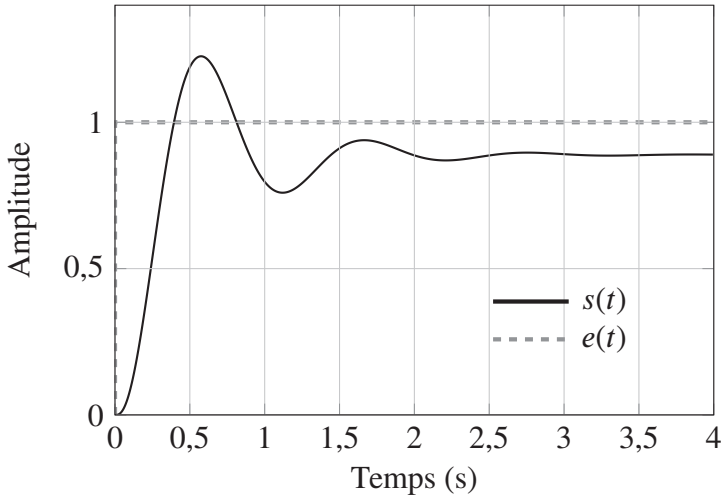


Figure 1 – Réponse temporelle à un échelon unitaire du système corrigé avec $C(p) = 1$

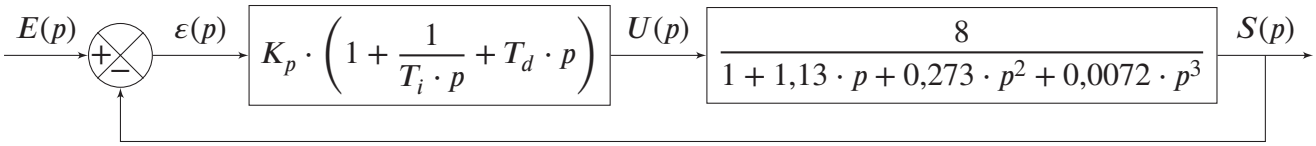
Partie II - Correction du système

II.1 - Définition du correcteur P.I.D.

On installe dans le système un correcteur P.I.D. :

$$C(p) = K_p \cdot \left(1 + \frac{1}{T_i \cdot p} + T_d \cdot p\right) .$$

Le système devient :



Q2. Mettre le correcteur sous la forme $C(p) = \dfrac{N_C(p)}{D_C(p)}$ avec $N_C(p)$ et $D_C(p)$ deux polynômes à définir. Puis, sur le **listing page DR 1**, écrire la fonction **correcteur(Kp,Ti,Td)** qui renvoie les listes **numC** et **denC** associées au numérateur et au dénominateur du correcteur P.I.D.

Le numérateur et le dénominateur du correcteur seront nommés respectivement **numC** et **denC** pour la suite.

II.2 - Détermination des fonctions de transfert en boucle ouverte et en boucle fermée

La détermination de la fonction de transfert en boucle ouverte $BO(p)$ et de la fonction de transfert en boucle fermée $BF(p)$ nécessite de développer deux fonctions : une fonction qui réalise le produit de deux polynômes et une autre qui en fait la somme.

Soit la fonction **multi_listes(P,Q)** (**listing page DR 2**) qui prend comme argument deux listes **P** et **Q**.

Q3. Décrire l'exécution de la fonction **multi_listes(P,Q)** lorsque les deux listes sont **P=[a,b,c]** et **Q=[r,s]** en précisant sur le **document réponse page DR 2** dans le cadre correspondant à cette question, le contenu des variables suivantes :

- de la liste Q1 avant la boucle for ;
- de la liste R pour $k = 2$ et $i = 1$;
- de la liste R à la fin de l'exécution.

Que fait la fonction **multi_listes(P,Q)** ?

Afin que le résultat du produit des deux listes corresponde au produit de deux polynômes, tels qu'ils sont définis par la librairie **scipy.signal**, il est nécessaire de ranger les termes de ce résultat dans l'ordre inverse.

Pour la suite, chaque fois qu'une liste sera l'image d'un polynôme, on considérera que les coefficients sont rangés dans l'ordre décroissant des puissances de p (voir la définition de la fonction **lti()** en **annexe A.1**).

Q4. Écrire la fonction **inverse(liste)** qui prend comme argument une liste **liste** et renvoie une liste **liste_r** en inversant les coefficients sur le **listing page DR 2** dans le cadre correspondant à cette question.

Soient $F_1(p) = \dfrac{N_1(p)}{D_1(p)}$ et $F_2(p) = \dfrac{N_2(p)}{D_2(p)}$ deux fonctions de transfert avec $N_i(p)$ et $D_i(p)$ des polynômes en p. On cherche à déterminer $F_1(p) \times F_2(p)$.

On note **num1**, **num2**, **den1**, **den2** les quatre listes représentant les polynômes $N_1(p)$, $N_2(p)$, $D_1(p)$ et $D_2(p)$.

Q5. Écrire la fonction **multi_FT(num1,den1,num2,den2)** du **listing page DR 3** dans le cadre correspondant à cette question qui prend comme argument quatre listes représentant les polynômes du numérateur et du dénominateur de deux fonctions de transfert et renvoie deux listes représentant le produit de ces deux fonctions de transfert rangées comme des polynômes. Pour cela, vous utiliserez les deux fonctions **multi_listes(P,Q)** et **inverse(liste)**.

Il reste à déterminer la fonction qui réalise la somme de deux polynômes.

Q6. Écrire la fonction **somme_poly(P,Q)** qui prend en argument deux listes représentant les coefficients de deux polynômes et renvoie une liste des coefficients de la somme des deux polynômes. Ces deux polynômes sont de dimensions différentes.

II.3 - Tracé de la réponse temporelle

Finalement, pour obtenir la réponse temporelle, il est nécessaire de compléter les fonctions définies précédemment par deux fonctions :

- **FTBF(num,den)**, fonction qui renvoie le numérateur et le dénominateur de la fonction de transfert en boucle fermée $\frac{S(p)}{E(p}$
- **rep_Temp(Kp,Ti,Td)**, fonction qui permet d'obtenir les deux listes **t** (le temps) et **s** l'amplitude de la sortie pour un échelon unitaire.

Ces deux fonctions sont définies ci-dessous.

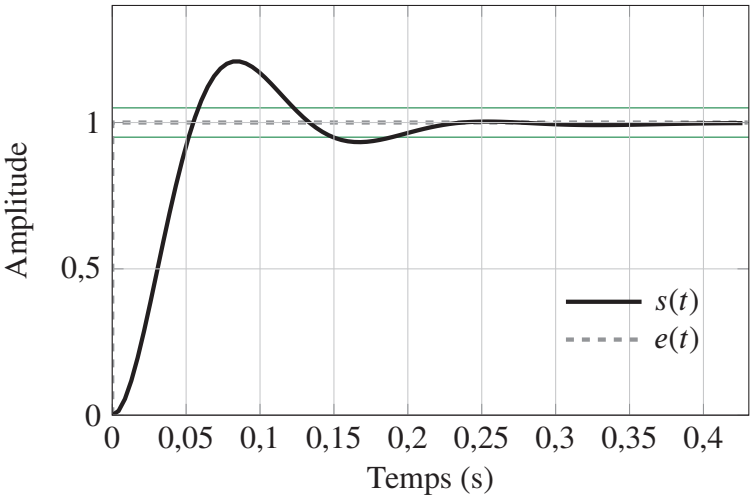


Figure 2 – Tracé de la réponse temporelle corrigée

Le listing du programme de la page 6 montre une utilisation de ces fonctions permettant de tracer la réponse temporelle pour un échelon unitaire en prenant comme paramètres du correcteur **Kp=5**, **Ti=1,1** et **Td=0,218**. La **figure 2** représente le tracé de la réponse temporelle correspondante.

Sur cette figure, on retrouve la consigne d'entrée $e(t)$ en trait pointillé, la sortie corrigée $s(t)$ en trait fort et les deux limites à $\pm$ 5 % autour de la valeur finale en tracés fins.

Définition des fonctions FTBF() et rep_Temp()

```
1  def FTBF(num,den) :
2          num_bf=num
3          den_bf=somme_poly(num,den)
4          return num_bf,den_bf
5
6  def rep_Temp(Kp,Ti,Td) :
7          numC,denC=correcteur(Kp,Ti,Td)
8          num_BO, den_BO=multi_FT(numG,denG,numC,denC)
9          num_BF,den_BF=FTBF(num_BO,den_BO)
10         BF=lti(num_BF, den_BF)
11         temps, sortie = step(BF)
12         temps, sortie = step(BF, T = linspace(min(temps), temps[−1], 500))
13         return temps,sortie
```

Programme de tracé de la réponse temporelle

```
numG = [ ..... ]
denG = [.....] #Question 1
#définition du correcteur (valeurs quelconques)
Kp,Ti,Td=5,1.1,0.218
t,s= rep_Temp(Kp,Ti,Td)
plt.plot(t[:60], s[:60])
plt.title('Réponse temporelle à un échelon')
plt.xlabel('Temps (s)')
plt.ylabel('Amplitude')
plt.hlines(1, min(t), t[60], colors='r')
plt.hlines(0.95, min(t), t[60], colors='g')
plt.hlines(1.05, min(t), t[60], colors='g')
plt.hlines(0, min(t), t[60])
plt.grid()
plt.show()
```

Partie III - Détermination des critères d'optimisation

III.1 - Réglage optimal pratique

Dans le cas qui nous intéresse, la présence d'une intégration garantit que l'erreur indicielle sera nulle à la condition que le système reste stable. Une méthode de réglage pratique (méthode de compensation des pôles) a permis de déterminer les valeurs suivantes pour le correcteur :
$K_{p0} = 2{,}33$,
$T_{i0} = 1{,}1$ s,
$T_{d0} = 0{,}218$ s.

Le tracé (**figure 3**) montre une réponse temporelle qui semble optimisée.

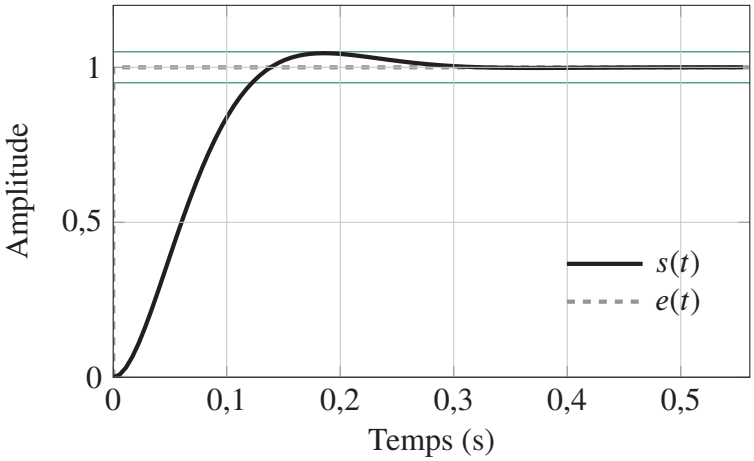


Figure 3 – Réponse temporelle optimisée obtenue par la méthode de compensation des pôles

L'objet de l'étude qui suit est de déterminer un réglage du correcteur P.I.D. qui améliore encore la réponse temporelle.

III.2 - Stabilité

Avant de définir les différents critères d'optimisation de la réponse temporelle, il est nécessaire de vérifier que la réponse temporelle est convergente. La réponse est convergente si le système est stable. On rappelle la définition de la **stabilité** : une fonction de transfert est stable si toutes les racines du dénominateur (les pôles) sont à parties réelles strictement négatives.

On cherche donc à vérifier que la fonction de transfert obtenue est stable et ce, pour différentes valeurs de K_p, T_i et T_d.

Q7. Écrire la fonction **stabilite(P)** qui prend comme argument une liste représentant un polynôme et renvoie la valeur binaire **True** si la fonction de transfert, dont le dénominateur est $P(p)$, est stable et **False** dans l'autre cas. Vous utiliserez les fonctions **roots(...)** et **real(...)** de la librairie **numpy** (voir l'**annexe A.2**, page 15).

III.3 - Critères usuels d'optimisation

Les critères usuels de réglage et d'optimisation d'un système asservi généralement retenus sont le temps de réponse à 5 %, l'amplitude des dépassements et une erreur indicielle nulle. On tente souvent d'avoir une réponse temporelle analogue à la réponse d'un système du second ordre avec un temps de réponse minimal et un dépassement limité à 5 %.

On rappelle ici les définitions :

Temps de réponse à 5 % : le temps de réponse à 5 % est le temps mis par le système pour que la réponse temporelle atteigne la valeur finale à 5 % près ;

Dépassement relatif : le dépassement relatif est défini par $D_1 = \dfrac{s_{max} - s_\infty}{s_\infty}$ avec s_{max} la valeur maximale de la fonction et s_∞ la valeur finale en régime permanent.

Q8. Justifier que la fonction **Temps_reponse(s,t)** (**listing page DR 4**) qui prend comme argument deux listes, la réponse temporelle et le temps, renvoie une valeur approchée majorant le temps de réponse à 5 %.

Q9. Proposer une ré-écriture de la fonction **Temps_reponse(s,t)** en utilisant une seule boucle *while* à la place de la boucle *for* et du test.

Q10. Écrire la fonction **depassement(s,t)** qui prend en argument les listes **s** et **t** et qui renvoie le dépassement relatif.

III.4 - Critère de la valeur absolue de l'intégrale de l'erreur

À ces critères, on peut en rajouter un qui va caractériser la réponse transitoire : il s'agit de l'intégrale de la valeur absolue de l'erreur (IAE) :

$$IAE = \int_0^{+\infty} |\varepsilon(t)| \mathrm{d}\, t$$

avec $s(t)$ la réponse temporelle de la sortie, $e(t) = 1$ la consigne d'entrée et $\varepsilon(t) = e(t) - s(t)$.

Ce critère permet d'évaluer l'aire de la surface entre la courbe de réponse $s(t)$ et la consigne d'entrée $e(t) = 1$ (**figure 4**).

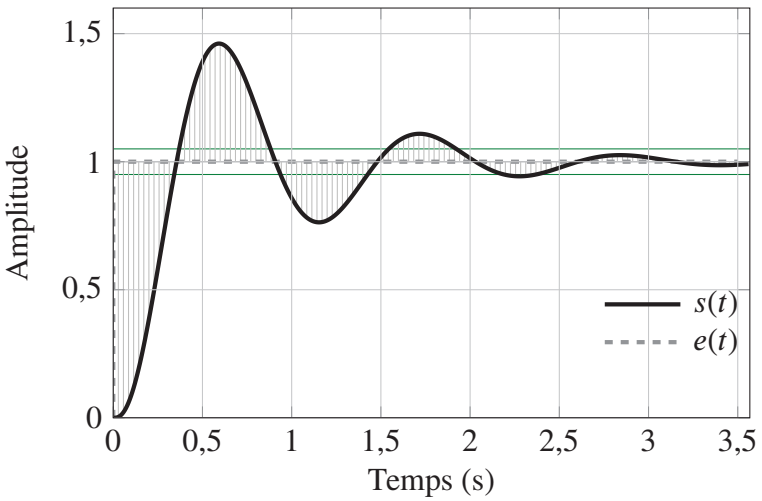


Figure 4 – Critère de la valeur absolue de l'erreur

Q11. Écrire la fonction **critere_IAE(s,t)** qui prend comme argument les listes **s** et **t** et qui renvoie l'intégrale de la valeur absolue de l'erreur. Vous préciserez la méthode utilisée.

III.5 - Fonction coût

Si on applique ces trois critères à la fonction réglée par la méthode de compensation des pôles (**figure 3**) avec $K_{p0} = 2{,}33$, $T_{i0} = 1{,}1$ s et $T_{d0} = 0{,}218$ s, on obtient respectivement pour les critères de temps de réponse ($T_{5\%}$), de dépassement relatif (D_1) et d'écart (IAE) :

$$\begin{aligned} T_{50} &= 0{,}123\,\text{s}, \\ D_{10} &= 5\,\% = 0{,}05, \\ IAE_0 &= 0{,}0733\ \text{USI}. \end{aligned}$$

Ces trois critères sont ceux que notre algorithme doit permettre d'améliorer. On définit ainsi une fonction **ponderation_cout(T5,D1,IAE)** (**listing page DR 5**) qui renvoie le « coût » global associé à ces critères. Le réglage optimal du correcteur doit alors permettre de minimiser cette fonction.

Q12. Dans quel intervalle doit être la valeur retournée par **ponderation_cout(T5,D1,IAE)** afin que la configuration soit meilleure que la réponse de référence ?

Nous allons chercher le meilleur triplet (K_p, T_i, T_d) qui minimise la fonction coût.

Les domaines de définition de K_p, T_i et T_d sont :
- $K_p \in [0{,}01, 100]$,
- $T_i \in [0{,}01\,\text{s}, 100\,\text{s}]$,
- $T_d \in [0\,\text{s}, 50\,\text{s}]$.

Afin de limiter l'étude, nous considérerons que chacune des trois variables ne peut prendre que 2^{16} valeurs réparties uniformément dans l'intervalle de définition :

$$K_p = (K_{pmax} - K_{pmin}) \cdot \frac{I_p}{2^{16} - 1} + K_{pmin}, \tag{1}$$

$$T_i = (T_{imax} - T_{imin}) \cdot \frac{I_i}{2^{16} - 1} + T_{imin}, \tag{2}$$

$$T_d = (T_{dmax} - T_{dmin}) \cdot \frac{I_d}{2^{16} - 1} + T_{dmin}. \tag{3}$$

La fonction **calcul_coef_correcteur(Ip,Ii,Id)** renvoie les coefficients du correcteur K_p, T_i et T_d.

Q13. Compléter la fonction **calcul_coef_correcteur(Ip,Ii,Id)** qui détermine les coefficients du correcteur à partir des trois valeurs entières I_p, I_i et I_d. Estimer le nombre de combinaisons possibles pour le correcteur.

La fonction **calcul_cout(Ip,Ii,Id)** doit permettre, pour une combinaison des trois valeurs **(Ip,Ii,Id)** dans leur domaine de définition, de déterminer le coût de la combinaison. Pour cela, il faut :
- déterminer les coefficients du correcteur P.I.D. ;
- déterminer le numérateur et le dénominateur de la fonction de transfert en boucle ouverte ;
- déterminer le numérateur et le dénominateur de la fonction de transfert en boucle fermée ;
- si la fonction de transfert en boucle fermée :
  - est stable, alors déterminer le temps de réponse, le dépassement et le critère IAE ;
  - n'est pas stable, alors prendre pour le coût une valeur supérieure aux valeurs possibles (ici coût = 100).

Finalement, la fonction doit renvoyer le coût.

Rappel des fonctions précédemment définies à utiliser :

stabilite(P), **Temps_reponse(s,t)**, **critere_IAE(s,t)**,
rep_Temp(Kp,Ti,Td), **depassement(s,t)**, **ponderation_cout(T5,D1,IAE)**.

Q14. Compléter la fonction **calcul_cout(Ip,Ii,Id)** en utilisant les fonctions précédemment définies.

Q15. Sachant que le temps de calcul de la fonction **calcul_cout(Ip,Ii,Id)** pour une combinaison est de 10,3 ms, est-il envisageable de les tester toutes ?

Pour la suite, nous utiliserons la fonction **calcul_cout(Ip,Ii,Id)** qui, à partir des trois valeurs entières I_p, I_i et I_d renvoie le coût de la combinaison. C'est cette quantité que nous chercherons à minimiser dans la partie suivante.

Partie IV - Résolution par un algorithme génétique

IV.1 - Principe

Les algorithmes génétiques sont des processus d'optimisation itératifs évolutionnistes. Ils permettent d'approcher la solution d'un problème d'optimisation (ici de recherche d'un minimum) en reproduisant des mécanismes de « sélection naturelle ». Le principe, décrit **figure 5**, consiste à initialiser un groupe de candidats possibles dont on va sélectionner les meilleurs éléments. Ces derniers sont conservés et croisés pour obtenir de nouveaux candidats. Ainsi, à chaque génération, la population se conserve et les caractéristiques du groupe s'améliorent jusqu'à converger vers un optimum.

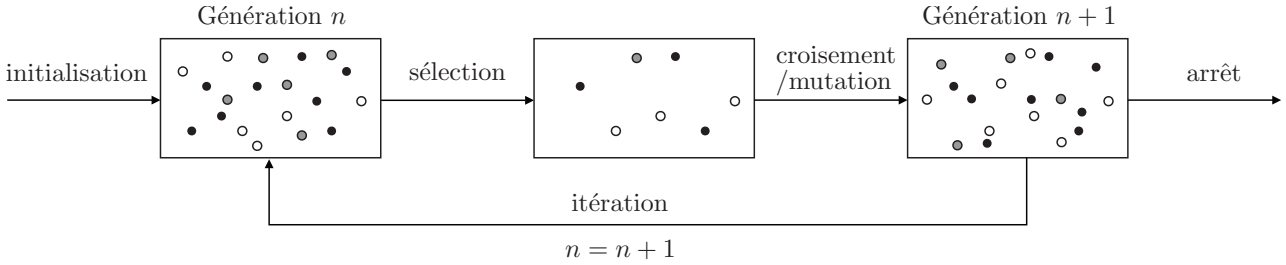


Figure 5 – Principe d'un algorithme génétique

Nous nous intéressons ici au programme, présenté dans son ensemble dans l'**annexe A.3**, page 16, qui doit permettre de trouver une combinaison des paramètres I_p, I_i et I_d qui présente un optimum vis-à-vis de la minimisation de la fonction **calcul_cout(Ip,Ii,Id)** définie dans la partie précédente.

Nous utiliserons une population de 100 candidats. À chaque génération, seuls les 20 meilleurs sont sélectionnés et croisés de façon à générer 80 nouveaux candidats et conserver une population de 100 candidats. Afin d'éviter de converger trop rapidement vers un minimum local, les nouveaux candidats subissent une mutation aléatoire pour apporter de la diversité génétique.

Ce cycle est répété jusqu'à ce qu'un critère d'arrêt vienne l'interrompre. Pour le moment nous supposerons que 50 cycles permettent d'obtenir une bonne convergence. Le meilleur individu de la dernière génération servira alors pour le réglage des paramètres I_p, I_i et I_d.

IV.2 - Initialisation des candidats

Chaque candidat possède trois gènes correspondant à chacun des paramètres I_p, I_i et I_d qui peuvent prendre 2^{16} valeurs distinctes (voir les équations (1), (2) et (3) page 8). Pour faciliter la manipulation des gènes et leurs croisements, ceux-ci sont codés sous la forme d'une chaîne de 16 caractères **'0'** ou **'1'** correspondant à un nombre binaire.

Q16. Définir le rôle de la fonction **genererGene(n)** et le type du paramètre **gen2** renvoyé. Quelle valeur de **n** faut-il alors employer pour créer un gène ?

Chaque candidat est pourvu de 3 gènes stockés sous la forme d'une liste **Candidat** de 3 chaînes de 16 caractères, par exemple :

Candidat=['1000100010001000','1000101010000100','1000111001101000'].

Q17. Compléter la fonction **generer_liste_initiale(n)** de façon à générer une liste **CandidatS** de 100 candidats aléatoires sur le **listing page DR 7**.

Par ailleurs, pour employer la fonction **calcul_cout(Ip,Ii,Id)** définie page 8, il est nécessaire de traduire la valeur décimale associée au gène codé en binaire à l'aide d'une fonction **décodage()**. Soit un gène **b2** codé sur n caractères binaires, la commande **float(b2[0])** renvoie la valeur du bit de poids le plus faible (le premier) et **float(b2[n−1])** renvoie la valeur du bit de poids le plus fort (le n-ième).

Q18. Calculer le nombre décimal associé à **b2='1001001000000000'**.

Q19. Compléter la fonction **decodage(b2)** qui prend en argument une chaîne de caractère **b2** et qui renvoie le nombre décimal **b10** associé.

IV.3 - Tri et sélection des candidats

Parmi les candidats triés dans l'ordre croissant de leur performance, ne sont sélectionnés et stockés que les 20 meilleurs candidats dans la liste **CandidatS_top** (**listing page DR 8**). Cette liste doit permettre d'une part de stocker ces candidats dans une base de données et d'autre part de constituer et générer la génération suivante.

Q20. Quel est l'algorithme de tri utilisé dans la fonction **tri(L)** du **listing page DR 8** pour trier **L** contenant les candidats ? Donner sa complexité dans le meilleur et le pire des cas.

IV.4 - Croisement des candidats

La fonction **croisement(P1,P2)** doit permettre de générer deux nouveaux individus à partir de deux parents **P1** et **P2**. Les gènes des deux parents sont découpés, croisés et recombinés 2 à 2 de façon à générer deux nouveaux gènes.

Chacun des trois gènes des deux parents est donc découpé en trois morceaux (**figure 6**). La partie centrale allant du 5e au 12e bit est échangée de façon à créer les gènes de deux enfants. Ainsi, le couple de parents (**P1**, **P2**) ci-dessous génère, avant mutation, les deux enfants (**E1**, **E2**) :

P1=['1111111111111111','0000000000000000','1111111100000000'],
P2=['0000000000000000','1111111111111111','0000000011111111'],

E1=['1111000000001111','0000111111110000','1111000011110000'],
E2=['0000111111110000','1111000000001111','0000111100001111'].

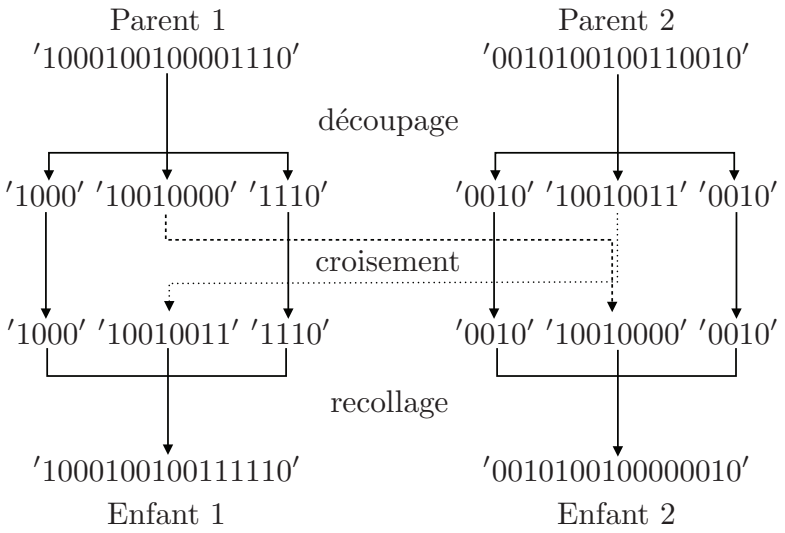


Figure 6 – Principe du croisement

Q21. Compléter le **listing** de sorte que la fonction **croisement(P1,P2)** renvoie les deux enfants **E1, E2** issus de deux parents **P1, P2**.

Avant d'être intégré à la nouvelle génération et afin de limiter le risque d'apparition de clone, chaque enfant subit une mutation sur l'un de ces gènes.

Q22. Pour **Candidat=['1111111111111111','0000000000000000','1111111100000000']**, donner ce que renvoie la commande **mutation(Candidat,1,12)** du **listing page DR 9**.

L'étape de croisement des 20 meilleurs candidats, qui sont conservés, doit permettre de générer 80 nouveaux candidats de façon à revenir à une population de 100 individus :

1) 20 sont issus de 10 croisements du meilleur des 20 meilleurs candidats avec un des 19 autres candidats (première boucle **for** du **listing page DR 9**) ;
2) 60 sont issus du croisement de 30 couples aléatoirement formés des 20 meilleurs candidats excepté le meilleur (deuxième boucle **for** du **listing page DR 9**).

Q23. Compléter la fonction **nouvGeneration(L)** du **listing page DR 9** de façon à traduire le point 2) ci-dessus.

Il est malgré tout possible que des clones apparaissent à chaque génération, d'autant plus que les candidats risquent de se ressembler au fur et à mesure des itérations. Pour éviter ce problème, on utilise une fonction **doublons()** qui repère les clones et qui les remplace par un candidat tiré aléatoirement.

Q24. Créer la fonction **doublons(L)** du **listing page DR 10** qui prend en argument une liste **L** de candidats et qui renvoie cette même liste débarrassée de ses clones et les remplace par un candidat tiré aléatoirement.

Partie V - Historique des candidats

Dans l'objectif d'étudier les paramètres qui permettent d'accélérer l'algorithme, nous décidons de stocker tous les meilleurs candidats de chaque itération dans une base de données. L'objectif est de déterminer un critère d'arrêt plus efficace que celui défini au-dessus.

La base de données est ainsi constituée d'une table Historique (**tableau 1**) qui permet de stocker les meilleurs candidats au fil des itérations. En plus de l'Id et des trois paramètres flottants K_p, T_i et T_d, elle permet de stocker trois autres attributs : score, apparition et disparition (**tableau 1**).

Historique
Id
gene_Kp
gene_Ti
gene_Td
score
apparition
disparition

Tableau 1 – Table Historique

L'attribut score est un flottant correspondant au résultat de la fonction coût du candidat. Les attributs apparition et disparition sont des entiers et correspondent respectivement au numéro de l'itération où le candidat est apparu et au numéro de celle où il a disparu. On se basera ici sur l'extrait de cette base de données fourni **tableau 2**.

Q25. Définir le but et le résultat de la requête SQL suivante :
SELECT min(score) FROM Historique.

Q26. Donner la requête SQL qui permet d'obtenir la longévité du meilleur candidat. Donner le résultat de cette requête et conclure sur la possibilité de stopper l'algorithme génétique avant la 50e itération.

Q27. Définir le but et le résultat de la requête SQL suivante :
SELECT Id FROM Historique WHERE 15 > apparition AND 15 < disparition.

Q28. Donner la requête SQL qui renvoie la valeur moyenne des gènes Kp, Ti et Td à la 20e itération.

Partie VI - En conclusion

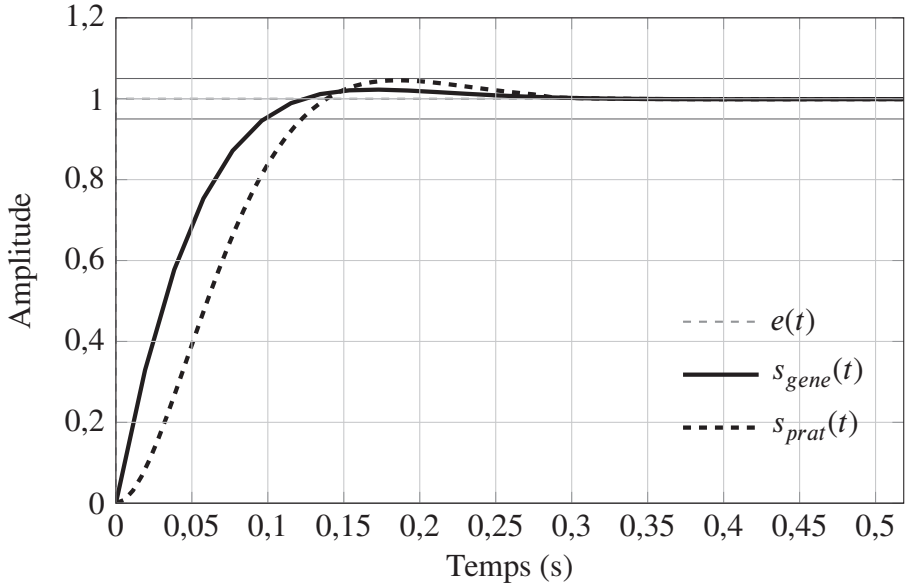


Figure 7 – Tracés comparés de la réponse temporelle déterminée par l'algorithme génétique et la réponse déterminée expérimentalement par la méthode de compensation des pôles

On retrouve sur la **figure 7** le tracé de la réponse « pratique » pour laquelle les coefficients du correcteur P.I.D. ont été déterminés expérimentalement ($s_{prat}(t)$) et le tracé de la réponse temporelle correspondant à la meilleure réponse de l'algorithme génétique ($s_{gene}(t)$).

Q29. Conclure sur l'opportunité d'utiliser un algorithme génétique.

Id	gene_Kp	gene_Ti	gene_Td	score	apparition	disparition
1	65.0542311742	25.9980929274	8.0804150454	1.1465868477	0	2
2	94.0221130694	21.8343222705	12.8343633173	1.47592141334	0	0
3	62.1003799496	12.9620883497	13.1906614786	1.53111188552	0	0
4	63.3377628748	5.39437033646	0.718699931334	1.66453693499	0	0
5	86.7534419776	39.6642320897	7.62493324178	1.76681066691	0	0
6	68.0126596475	2.17198718242	1.37788967727	0.720574832859	1	5
7	49.5983877317	6.71109224079	9.67421988251	1.23600614063	1	2
8	44.8868729686	7.2237441062	10.4859998474	1.2405098287 9	1	1
9	81.7001593042	29.2158988327	7.68596932937	1.24876332573	1	1
10	65.0542311742	25.9980929274	8.07431143664	1.16317640581	2	2
11	46.4492405585	7.2237441062	10.4921034562	1.24017983164	2	2
12	62.9593769741	4.02729869535	1.43892576486	0.750913688522	3	5
13	68.2079555962	2.17198718242	1.37788967727	0.774983006043	3	4
...	...	...	...	...	...	...
43	65.1686623941	0.417375143053	1.37865262837	0.621608931066	10	10
44	65.1808683909	0.411272144656	1.37865262837	0.620980646341	11	11
45	65.1808683909	0.412797894255	1.37865262837	0.621000002184	11	11
46	65.2296923781	0.402117647059	1.37865262837	0.618622779634	12	16
47	65.2296923781	0.42652964065	1.37865262837	0.618936823201	12	13
48	65.1869713893	0.414323643854	1.37865262837	0.620742325779	12	12
49	65.1823941405	0.402117647059	1.37865262837	0.620793058622	12	12
50	65.2357953765	0.414323643854	1.37865262837	0.618499911309	13	18
51	65.2296923781	0.414323643854	1.37865262837	0.618782759248	13	13
52	65.1884971389	0.408220645457	1.37865262837	0.620595007049	13	13
53	65.2327438773	0.414323643854	1.37865262837	0.618641425052	14	15
54	65.2296923781	0.408220645457	1.37865262837	0.618703560070	14	14
55	65.2312181277	0.414323643854	1.37865262837	0.618712114622	14	14
56	65.2342696269	0.414323643854	1.37865262837	0.618570690595	15	16
57	65.2312181277	0.408220645457	1.37865262837	0.61863287033	15	15
58	65.2357953765	0.406694895857	1.37865262837	0.618400625578	16	18
59	65.2373211261	0.414323643854	1.37865262837	0.618429087253	16	18
60	65.2373211261	0.402117647059	1.37865262837	0.618268651270	17	49
61	65.2373211261	0.406694895857	1.37865262837	0.618329745253	17	20
62	65.2373211261	0.405169146258	1.37865262837	0.618309616353	19	49
63	65.2357953765	0.402117647059	1.37865262837	0.61833956633	19	20
64	65.2373211261	0.408220645457	1.37865262837	0.618349663168	19	20
65	65.2373211261	0.400591897459	1.37865262837	0.618248479198	21	49
66	65.2373211261	0.403643396658	1.37865262837	0.618289307302	21	49
67	65.2357953765	0.400591897459	1.37865262837	0.618319406010	21	49

Tableau 2 – Historique partiel des meilleurs candidats

FIN

ANNEXE

A.1 Librairie scipy.signal

Le code ci-dessous illustre l'utilisation de cette librairie pour tracer la réponse temporelle d'un système du second ordre :

$$G(p) = \frac{K}{1 + 2 \cdot z \cdot \frac{p}{\omega_n} + \frac{p^2}{\omega_n^2}}.$$

Exemple d'utilisation de la librairie scipy.signal

```
from numpy import min, max,zeros
from scipy import linspace
from scipy.signal import lti, step, bode
from matplotlib import pyplot as plt
z=0.3
wn=10
K=1.3
d0=1
d1=2*z/wn
d2=1/wn**2
#Numérateur
num = [K]
#Dénominateur
den = [d2, d1, d0]
# Fonction de transfert
G = lti(num, den) #déclaration de la fonction de transfert
t, s = step(G)
# Ici, on appelle de nouveau la fonction afin de préciser
# le nombre de points à afficher
t, s = step(G, T = linspace(min(t), t[-1], 1000))
#Affichage
plt.plot(t, s)
plt.title('Réponse␣temporelle')
plt.xlabel('Temps(s)')
plt.ylabel('Amplitude')
plt.hlines(1, min(t), max(t), colors='r') #tracé de l'échelon unitaire
plt.hlines(0, min(t), max(t))
plt.xlim(xmax = max(t))
plt.legend(('Réponse␣temporelle␣à␣un␣échelon',), loc=0)
plt.grid()
plt.show()
```

Nous utiliserons dans ce sujet les deux fonctions définies sur la page suivante.

lti(num,den) : cette fonction (ligne 16 au-dessus) permet de définir la fonction de transfert, elle prend comme argument deux listes, l'une comportant les coefficients du numérateur, l'autre comportant les coefficients du dénominateur (lignes 12 et 14).

Les coefficients des deux listes sont rangés dans l'ordre décroissant.
Si les polynômes du numérateur et du dénominateur sont notés :

$$N(p) = n_0 + n_1 \cdot p + n_2 \cdot p^2 + \cdots + n_k \cdot p^k$$
$$D(p) = d_0 + d_1 \cdot p + d_2 \cdot p^2 + \cdots + d_\ell \cdot p^\ell$$

alors les deux listes représentant les polynômes utilisables par la fonction **lti (num,den)** sont notées :

$$num = [n_k, n_{k-1}, \ldots, n_2, n_1, n_0],$$
$$den = [d_\ell, d_{\ell-1}, \ldots, d_2, d_1, d_0].$$

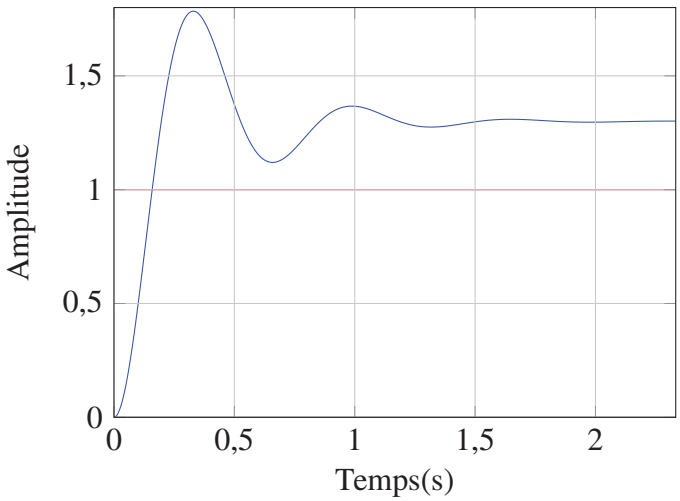


Figure 8 – Exemple d'utilisation de la librairie scipy.signal

step(G) : cette fonction de la librairie **scipy.signal** renvoie deux listes, le temps et la réponse temporelle pour une entrée en échelon unitaire de la fonction de transfert G. Si la fonction est appelée sans préciser la durée de simulation et le nombre de points, ceux-ci sont déterminés automatiquement par la librairie. Il est d'usage de l'appeler deux fois, la première sans paramètres pour laisser la librairie calculer le temps de simulation, puis une deuxième fois en reprenant le temps calculé et en imposant le nombre de points. La représentation temporelle est tracée sur la **figure 8**.

A.2 Librairie numpy

min(liste), **max(liste)** : ces deux fonctions qui prennent une liste comme argument, retournent respectivement le minimum et le maximum d'une liste.

roots(P) : cette fonction qui prend comme argument une liste dont les termes sont les coefficients d'un polynôme, renvoie une liste dont les termes sont les racines du polynômes.

```
p = [1,-3,-2,-1] # le polynôme P(x)=x^3-3x^2-2x-1
racines=roots(p)
print(racines)
```

Le script affiche : $[3.62736508 + 0.j, -0.31368254 + 0.42105281j, -0.31368254 - 0.42105281j]$

real(Q) : fonction dont l'argument est une liste et qui renvoie une liste constituée de la partie réelle de chaque terme de la liste passée en argument.

```
print(real(racines))
```

Le script affiche : $[3.62736508, -0.31368254, -0.31368254]$

zeros(n) : retourne une liste de n termes nuls.

A.3 Programme d'optimisation génétique

Cette annexe présente la structure générale du programme d'optimisation génétique et les différentes fonctions que vous aurez à commenter ou à définir. Ce programme doit permettre de trouver un optimum à la fonction **calcul_cout(Ip,Ii,Id)** à l'aide d'algorithme génétique.

Programme d'optimisation par algorithme génétique

```
#------------Paramètres  globaux----------------
n=_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
cycle=50
#-----------------Fonctions--------------------
def genererGene(n):
        # Détermination d'un gène
        return gen2

def generer_liste_initiale(n):
        # Génération de la liste initiale
         return liste_initiale

def decodage(b2):

        return b10

def perf(Li): # Li est la liste des attributs d'un candidat
        # Détermination de la perfomance d'une combinaison
        return calcul_cout(Ip,Ii,Id)

def tri(L): # L est la liste de tous les candidats
        # Tri croissant des différents candidats selon la performance
        return L

def croisement(P1,P2):
        # Fonction réalisant le croisement génétique
        return

def mutation(E,i,j):
        # Fonction réalisant une mutation
        return E

def nouvGeneration(L): # L la liste actuelle des candidats
        # Fonction déterminant la nouvelle génération
        return L_new

def doublons(L):
        # Fonction chargée d'éliminer les doublons

# Finalement le programme principal
#-----------------Main-------------------------
CandidatS=Generer_liste_initiale(n)
for i in range(cycle):
        CandidatS_top=tri(CandidatS)[0:20]
        CandidatS=nouvGeneration(CandidatS_top)
        CandidatS=doublons(CandidatS)
solution=CandidatS[0]
```

DANS CE CADRE

Académie : Session :

Examen ou Concours : **Concours Communs Polytechniques** Série* :

Spécialité/option : **FILIÈRE TSI** Repère de l'épreuve :

Épreuve/sous-épreuve : **INFORMATIQUE**

NOM :
(en majuscules, suivi, s'il y a lieu, du nom d'épouse)

Prénoms :

Né(e) le

N° du candidat

(le numéro est celui qui figure sur la convocation ou la liste d'appel)

NE RIEN ÉCRIRE

Examen ou Concours : **Concours Communs Polytechniques** Série* :

Spécialité/option : **FILIERE TSI**

Repère de l'épreuve : **INFORMATIQUE**

Épreuve/sous-épreuve :
(Préciser, s'il y a lieu, le sujet choisi)

Si votre composition comporte plusieurs feuilles, numérotez-les et placez les intercalaires dans le bon sens.

Note : /20

Appréciation du correcteur :*

* Uniquement s'il s'agit d'un examen.

DOCUMENT RÉPONSE

TSIIN07

Pour l'ensemble du sujet, les bibliothèques usuelles sont chargées avec la syntaxe suivante :

Librairies

```
1  from numpy import min, max, zeros, roots, real
2  from scipy import linspace
3  from scipy.signal import lti, step
4  from matplotlib import pyplot as plt
5  from random import random
```

Question 1

```
1  numG = [ ...
2
3  denG = [ ...
4
5  G = lti(numG, denG)
```

Question 2

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

```
1  def correcteur(Kp,Ti,Td) :
2          ...
3          ...
4          ...
5          ...
6
7
8
9
10
11
12
13         return num,den
14 numC,denC=correcteur(Kp,Ti,Td) # appel de la fonction, numC et denC les deux listes de coefficients.
```

NE RIEN ÉCRIRE DANS LA PARTIE BARRÉE

Question 3

```
1  def multi_listes(P,Q) :
2          deg_max=(len(P)-1)+(len(Q)-1)
3          P1=zeros(deg_max+1)
4          Q1=zeros(deg_max+1)
5          #P1 et Q1 deux listes de zéros de dimension deg_max+1
6          for j in range(len(P)) :
7                  P1[j]=P[j]
8          for j in range(len(Q)) :
9                  Q1[j]=Q[j]
10         R=zeros(deg_max+1)
11         for k in range(deg_max+1) :
12                 for i in range(k+1) :
13                         R[k]=R[k]+P1[i]*Q1[k-i]
14         return R
```

Q1= .....

.....

($k = 2$ et $i = 1$) : R= .....

.....

R= ....

Que fait multi_listes ?: .....

....

Question 4

```
1  def inverse(liste) :
2          ...
3          ...
4          ...
5          ...
6          ...
7          ...
8          ...
9          ...
10         ...
11         ...
12         ...
13         return liste_r
```

NE RIEN ÉCRIRE DANS LA PARTIE BARRÉE

Question 5

```
def multi_FT(num1,den1,num2,den2) :
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    return ... ... ... ...
```

Question 6

```
def somme_poly(P,Q) :
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    return somme
```

Question 7

```
def stabilite(P) :
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    ...
    return ...
```

NE RIEN ÉCRIRE DANS LA PARTIE BARRÉE

Question 8

```
1  def Temps_reponse(s,t) :
2          T5=0
3          s_fin=s[-1]
4          for tt in range(len(s)) :
5                  j=len(t)-tt-1
6                  if ((s[j]>s_fin*0.95 and s[j-1]<s_fin*0.95)or(s[j]<s_fin*1.05 and s[j-1]>s_fin*1.05)) :
7                          T5=t[j]
8                          break
9          return T5
```

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Question 9

```
1  def Temps_reponse(s,t) :
2          ...
3          ...
4          ...
5          ...
6          ...
7          ...
8          ...
9          ...
10         ...
11         ...
12         ...
13         return T5
```

Question 10

```
1  def depassement(s,t) :
2          ...
3          ...
4          ...
5          ...
6          ...
7          ...
8          ...
9          ...
10         ...
11         ...
12         ...
13         return D1
```

NE RIEN ÉCRIRE DANS LA PARTIE BARRÉE

Question 11

```
1  def critere_IAE(s,t) :
2          ...
3          ...
4          ...
5          ...
6          ...
7          ...
8          ...
9          ...
10         ...
11         return IAE
```

Question 12

```
1  T50=0.123
2  D10=0.05
3  IAE0=0.0733
4  def ponderation_cout(T5,D1,IAE) :
5          k1=1
6          k2=1
7          k3=1
8          cout=1/(k1+k2+k3)*(k1*T5/T50+k2*D1/D10+k3*IAE/IAE0)
9          return cout
```

NE RIEN ÉCRIRE DANS LA PARTIE BARRÉE

Question 13

```
def calcul_coef_correcteur(Ip,Ii,Id) :
        ...
        ...
        ...
        ...
        ...
        ...
        ...
        ...
        ...
        ...
        ...
        ...
        return Kp,Ti,Td
```

.....

.....

.....

.....

.....

.....

.....

.....

Question 14

```
def calcul_cout(Ip,Ii,Id)  :
        Kp,Ti,Td=coef_correcteur(Ip,Ii,Id)
        numC,denC=correcteur(Kp,Ti,Td)
        num_BO, den_BO=multi_FT(numG,denG,numC,denC)
        num_BF,den_BF=FTBF(num_BO,den_BO)
        stable=...
         ...
         ...
         ...
         ...
         ...
         ...
         ...
         ...
         ...
         ...
         ...
         ...
         ...
         ...
         ...
         ...
         return cout
```

NE RIEN ÉCRIRE DANS LA PARTIE BARRÉE

Question 15

.....

.....

.....

.....

.....

.....

.....

.....

Question 16

```
1  n=....
2  def genererGene(n) :
3          b2=''
4          for i in range(n) :
5                  b2=b2+str(int(random()*2))
6          return b2
```

.....

.....

.....

.....

.....

.....

.....

Question 17

```
1  def generer_liste_initiale(n) :
2          CandidatS=[]
3          for i in ...
4                  Candidat=...
5                  CandidatS.append(Candidat)
6          return CandidatS
```

Question 18

.....

.....

.....

.....

.....

.....

.....

NE RIEN ÉCRIRE DANS LA PARTIE BARRÉE

Question 19

```
1  def decodage(b2) :
2          b10=0
3          for...
4
5
6
7
8
9
10          return b10
```

Programme principal

```
1  cycle=50
2  #-----------------------Main-------------------------------
3  CandidatS=Generer_liste_initiale(n)
4  for i in range(cycle) :
5          CandidatS_top=tri(CandidatS)[0 :20]
6          CandidatS=nouvGeneration(CandidatS_top)
7          CandidatS=doublons(CandidatS)
8  solution=CandidatS[0]
```

Question 20

```
1  def perf(Li) : # Li est la liste des attributs d'un candidat
2          lp,li,ld=decodage(Li[0]),decodage(Li[1]),decodage(Li[2])
3          return calcul_cout(lp,li,ld)
4
5  def tri(L) : # L est la liste de tous les candidats
6          for i in range(1,len(L)) :
7                  if perf(L[i])<perf(L[i-1]) :
8                          p=i
9                          while p>0 and perf(L[i])<perf(L[p-1]) :
10                                 p=p-1
11                         X=L.pop(i)
12                         L.insert(p,X)
13         return L
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

NE RIEN ÉCRIRE DANS LA PARTIE BARRÉE

Question 21

```
def croisement(P1,P2) :
        E1=[]
        E2=[]
        ...
        ...
    return ...
```

Question 22

```
def mutation(E,i,j) :
        if E[i][j]=='1' :
                E[i]=E[i][:j]+'0'+E[i][j+1 :]
        else  :
                E[i]=E[i][:j]+'1'+E[i][j+1 :]
        return E
```

.....

.....

.....

.....

.....

Question 23

```
def nouvGeneration(L) : # L la liste actuelle des candidats
        L_new=L
        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 ...
                ...
                ...
                ...
                ...
        return L_new
```

NE RIEN ÉCRIRE DANS LA PARTIE BARRÉE

Question 24

```
1  def doublons ...
2          ...
3          ...
4          ...
5          ...
6
7
8
9
10
11
12         ...
```

Cadre réponse pour les questions 25 à 29

Préciser le numéro de la question, séparer les réponses.

NE RIEN ÉCRIRE DANS LA PARTIE BARRÉE

Cadre réponse pour les questions 25 à 29 (suite)

Préciser le numéro de la question, séparer les réponses.

NE RIEN ÉCRIRE DANS LA PARTIE BARRÉE

Cadre réponse libre

Préciser le numéro de la question, séparer les réponses.