MINISTÈRE
DE L'ÉDUCATION
NATIONALE
ET DE LA JEUNESSE
Liberté
Égalité
Fraternité

EAE SIN 2

SESSION 2023

AGREGATION
CONCOURS EXTERNE

Section : SCIENCES INDUSTRIELLES DE L'INGÉNIEUR

Option : SCIENCES INDUSTRIELLES DE L'INGÉNIEUR ET INGÉNIERIE INFORMATIQUE

MODÉLISATION D'UN SYSTÈME, D'UN PROCÉDÉ OU D'UNE ORGANISATION

Durée : 6 heures

Calculatrice autorisée selon les modalités de la circulaire du 17 juin 2021 publiée au BOEN du 29 juillet 2021.

L'usage de tout ouvrage de référence, de tout dictionnaire et de tout autre matériel électronique est rigoureusement interdit.

Il appartient au candidat de vérifier qu'il a reçu un sujet complet et correspondant à l'épreuve à laquelle il se présente.

Si vous repérez ce qui vous semble être une erreur d'énoncé, vous devez le signaler très lisiblement sur votre copie, en proposer la correction et poursuivre l'épreuve en conséquence. De même, si cela vous conduit à formuler une ou plusieurs hypothèses, vous devez la (ou les) mentionner explicitement.

NB : Conformément au principe d'anonymat, votre copie ne doit comporter aucun signe distinctif, tel que nom, signature, origine, etc. Si le travail qui vous est demandé consiste notamment en la rédaction d'un projet ou d'une note, vous devrez impérativement vous abstenir de la signer ou de l'identifier.
Le fait de rendre une copie blanche est éliminatoire

Tournez la page S.V.P.

INFORMATION AUX CANDIDATS

Vous trouverez ci-après les codes nécessaires vous permettant de compléter les rubriques figurant en en-tête de votre copie.

Ces codes doivent être reportés sur chacune des copies que vous remettrez.

Concours	Section/option	Epreuve	Matière
EAE	1417A	102	2680

IMPRIMERIE NATIONALE – 23 0095 – D'après documents fournis

Pilotage d'un exosquelette par une interface Cerveau-Machine

Introduction

Contexte scientifique et technique

En octobre 2019, de nombreux médias, ainsi qu'une publication scientifique majeure ont présenté une première mondiale : un homme de 30 ans, devenu paraplégique suite à un accident, équipé d'implants sur le cerveau et installé dans un exosquelette, a pu se déplacer en marchant au sein d'un laboratoire et toucher des cibles précises avec ses mains artificielles, en contrôlant le système uniquement par sa pensée, comme illustré en figure 1.

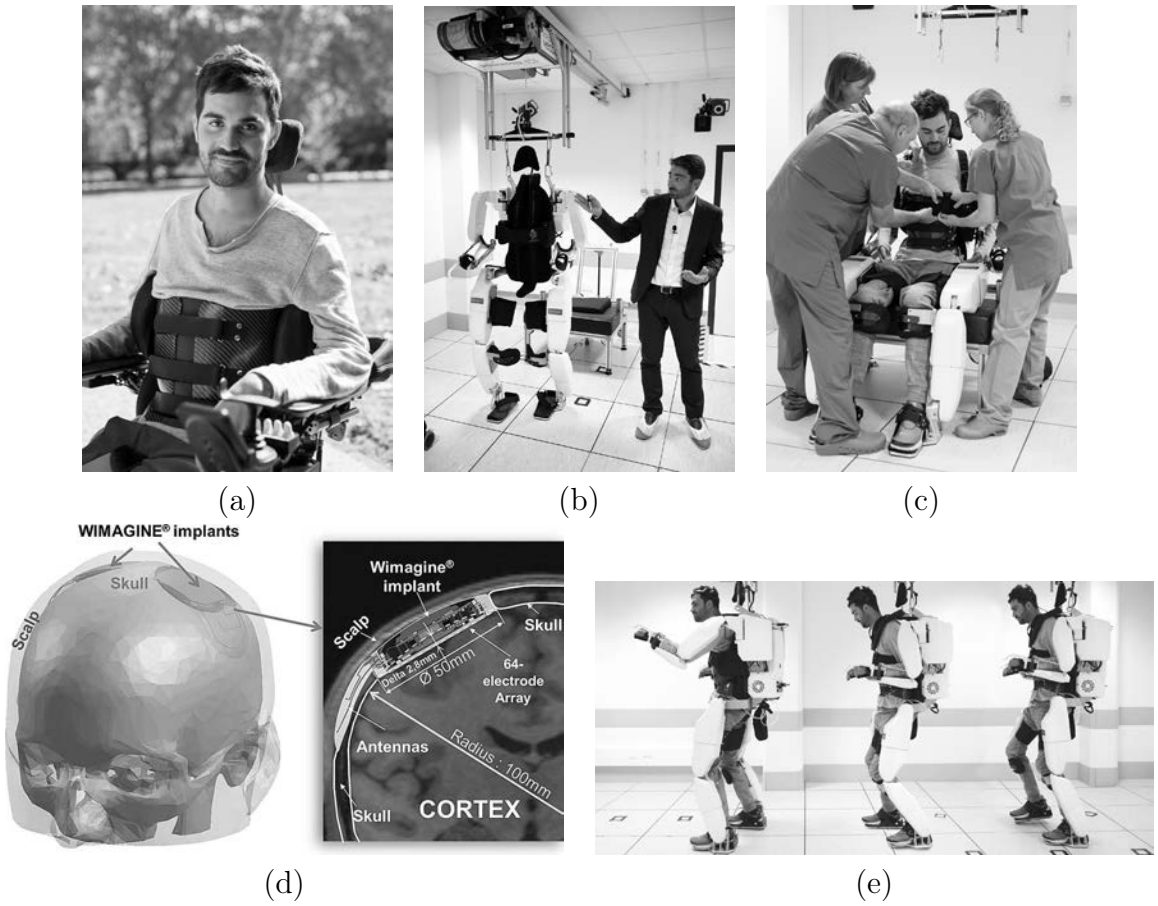


FIGURE 1 – Démonstration de mouvements autonomes contrôlés par la pensée du paraplégique : (a) patient (b) exosquelette (c) installation du patient (d) schéma et radiographie de présentation de l'implant (e) test de marche

Tournez la page S.V.P.

Cette prouesse majeure, développée au sein du projet BCI de CLINATEC, est un pas important vers une utilisation ambulatoire d'exosquelettes, pour laquelle il reste encore de nombreux défis à relever.

CLINATEC

Le centre CLINATEC, basé à Grenoble, présidé par le Pr. Benabid, est un établissement de recherche issu d'une collaboration entre le CEA (Commissariat à l'énergie atomique), le CHU de Grenoble et l'UGA (Université Grenoble Alpes), dans le but de lier étroitement recherche médicale et monde industriel. Il associe sur un même site une plateforme technique, où naissent des dispositifs technologiques de pointe, et un hôpital doté des meilleurs équipements. Une particularité qui permet de réunir au chevet des malades des équipes pluridisciplinaires regroupant roboticiens, mathématiciens, physiciens, électroniciens, informaticiens, biologistes, neurologues, chirurgiens et personnels de soins. Cette organisation innovante a pour objectif d'accélérer le transfert des innovations jusqu'au patient, en évaluant rapidement le fonctionnement et la pertinence de systèmes innovants, via la réalisation d'essais cliniques dans les meilleures conditions de sécurité. CLINATEC développe des dispositifs pour des patients souffrant de pathologies neurodégénératives, de cancers cérébraux et d'handicaps moteurs d'origine lésionnelle (tétra- ou paraplégie).

Le projet BCI (Brain Computer Interface)

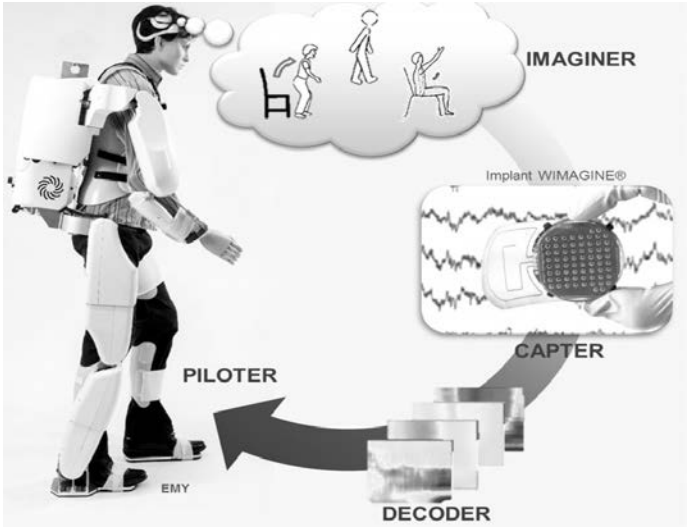
Au sein de CLINATEC, le projet BCI (Brain Computer Interface) a pour objectif de permettre aux personnes souffrant d'un handicap moteur lourd de retrouver de la mobilité grâce à un système de compensation, notamment via le pilotage d'un exosquelette à partir de signaux corticaux captés à l'aide d'un implant. Le principe du projet repose sur le fait qu'imaginer un mouvement ou l'exécuter provoque la *même activité électrique cérébrale* au niveau du cortex moteur. Deux implants permettent de capter ces signaux électriques d'électro-encéphalographie intracrânienne (*Electrocorticography* en anglais, abrégé ECoG par la suite) pour les transmettre à un système informatique, puis de les décoder, afin de piloter des objets complexes, comme bouger des systèmes mécaniques tel l'exosquelette EMY à 8 degrés de liberté.

Système étudié

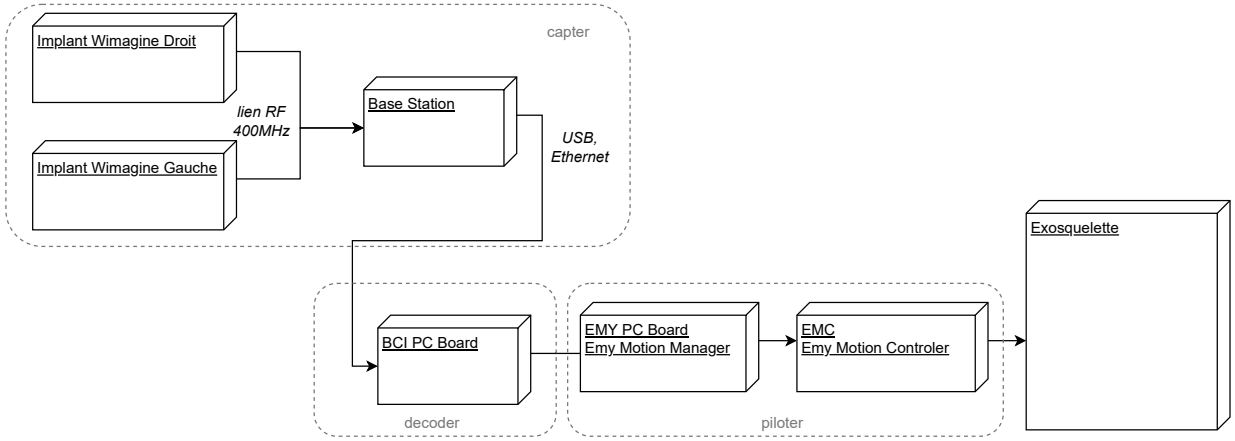
Le système étudié contient plusieurs sous-systèmes physiques permettant d'aller de l'acquisition de l'activité électrique cérébrale au pilotage des actionneurs. Une vue d'ensemble est donnée dans le diagramme de déploiement en figure 2(b). La chaîne logicielle est composée de 3 grands composants répartis sur les sous-systèmes [5] :

- la numérisation des signaux ECoG est assurée par les implants *WIMAGINE*. Lors d'une opération, deux implants sont posés à demeure dans la boîte crânienne du patient, à la place de l'os crânien et en dessous du cuir chevelu. Ils sont en contact avec le cerveau via des électrodes, pour enregistrer les hémisphères droit et gauche. Ils émettent, via des antennes, un signal reçu par un casque amovible posé sur la tête et connecté à une station de base installée dans le dos de l'exosquelette. Ce signal est synchronisé à la réception par la station de base. Les données sont découpées en lots pour le traitement de l'information en temps réel.

- Dans un seconde temps, les signaux ECoG sont traités et soumis à des modèles adaptatifs en charge d'estimer l'intention de mouvement du patient. Ce traitement est réalisé sur la cible matérielle *BCI PC Board* du diagramme de déploiement de la figure 2(b).
- Les ordres moteurs décodés sont ensuite transmis aux systèmes EMM(*EMY Motion Manager*) calculant le positionnement recherché pour l'exosquelette et EMC (*EMY Motion Controller*) gérant les commandes motrices adaptées au pilotage des actionneurs de l'exosquelette.



(a)



(b)

FIGURE 2 – (a) Présentation schématisée du projet BCI de Clinatec. Le patient souffrant d'un handicap moteur est placé dans un exosquelette et imagine le mouvement qu'il souhaite exécuter. Un implant cérébral permet de capter l'activité électrique résultante. Puis un système informatique décode ce signal et pilote l'exosquelette. (b) Diagramme de déploiement UML du système BCI.

La première partie de ce sujet se focalise sur la transmission des données d'enregistrement du signal ECoG et les possibilités de compression du signal afin de réduire le débit entre les implants et la station de base réceptionnant les données. Nous nous intéresserons ensuite au décodage des intentions dans le partie 2. Comment décoder les signaux pour prédire les mouvements ? Enfin dans la partie 3, nous étudierons l'exosquelette à 4 membres EMY, et le pilotage des axes asservis. La figure 3 présente de manière synthétique le cahier des charges

Tournez la page S.V.P.

partiel en lien avec les performances étudiées.

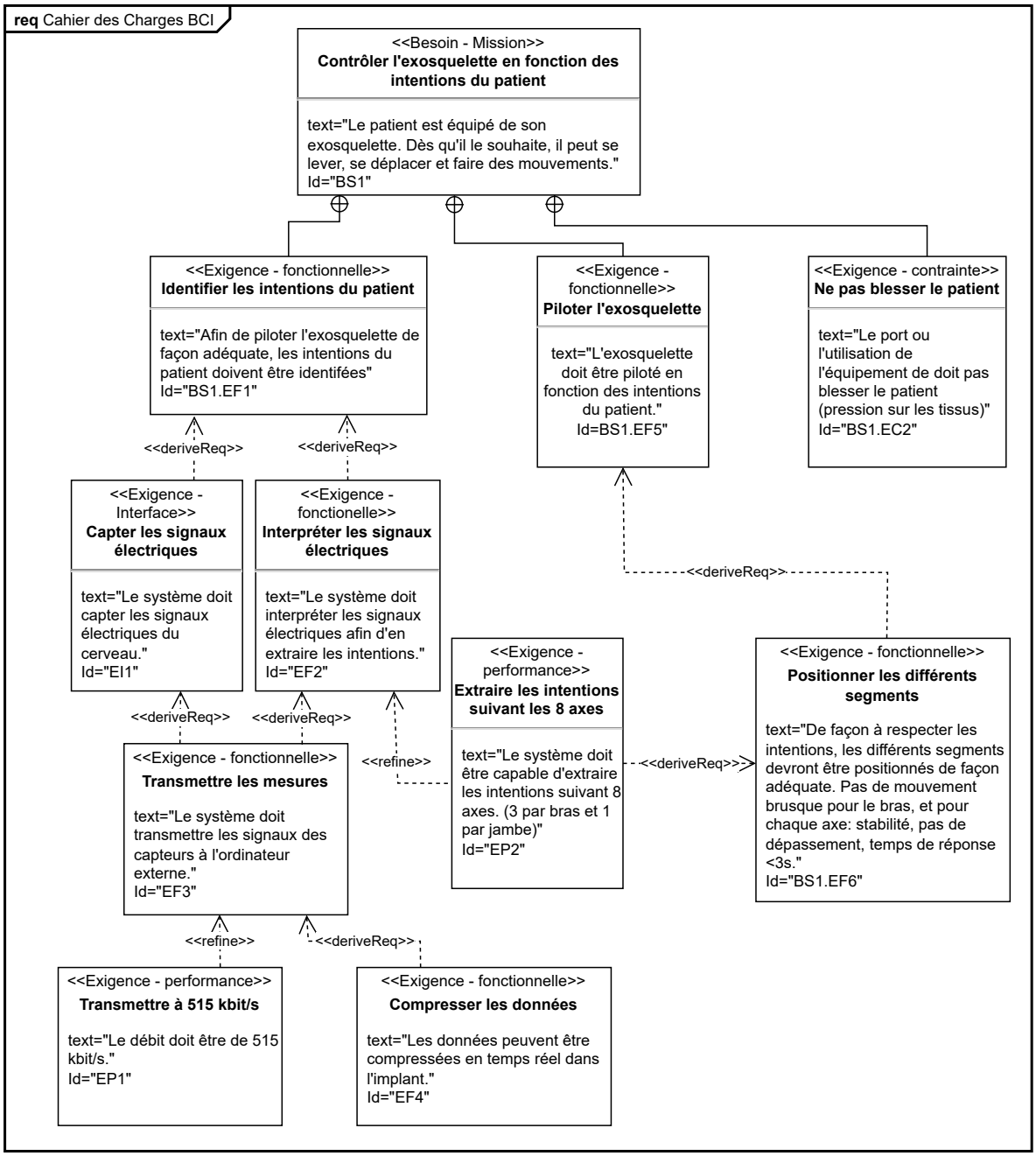


Figure 3 – Extrapolation du cahier des charges du système BCI développé par CLINATEC

Partie 1. Compression de l'information ECoG pour la transmission implant-station de base

L'enregistrement des signaux ECoG à la surface du cerveau est réalisé par les implants *WIMAGINE* présentés en figure 4 :

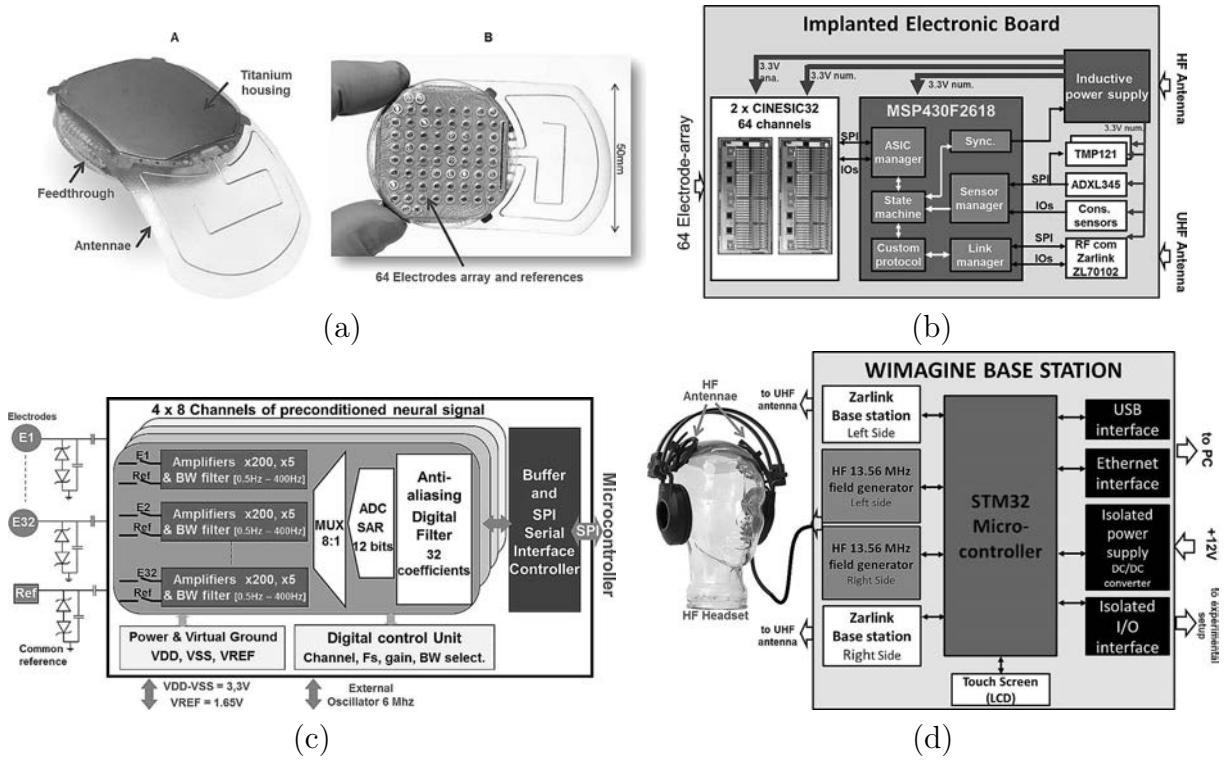


FIGURE 4 – Système implant et station de base : (a) Chaque implant est constitué d'une matrice de 64 électrodes au-dessus desquelles est positionnée l'électronique embarquée (b) Schéma synthétique de l'implant, contenant notamment un micro-contrôleur MSP420F2618 et deux ASIC CINESIC32 permettant l'enregistrement de 64 canaux en simultané (c) Schéma synthétique des ASIC CINESIC32, on peut noter la présence d'un convertisseur Analogique Numérique (SAR ADC 12 bits) (d) Schéma synthétique de la station de base : le lien UHF (400 MHz) sert à la transmission des données, le lien HF permet la transmission de puissance sans fil vers l'implant.

1.1 Contraintes à la transmission du signal ECoG

La communication entre l'implant et la station de base en charge de la réception des signaux ECoG est assurée par une liaison sans fil UHF 400 MHz conçue pour les applications médicales. Les implants utilisent un composant intégré, le *Zarlink ZL70102* dont les spécifications sont données dans le document technique DT1. Les données utilisateur transmises consistent en des paquets de 14 octets.

Question 1. L'objectif est dans un premier temps de déterminer la taille d'une mesure sur les canaux d'acquisition :

(a) Donner la taille d'un échantillon de tension ECoG pour un seul canal ?

(b) Calculer la taille de l'ensemble d'une mesure des potentiels d'ECoG ?

Tournez la page S.V.P.

(c) Calculer le nombre de *Data Blocks* définis dans la documentation constructeur (document technique DT1) nécessaire pour envoyer l'ensemble de la mesure ECoG.

Question 2. Il s'agit ici de déterminer le débit théorique et l'occupation de la bande passante par les signaux issus de l'implant. La fréquence d'échantillonnage du signal dans l'implant est de $f_e = 1\,\text{kHz}$.

(a) Déterminer le débit binaire théorique pour la transmission de l'ensemble des signaux ECoG.

(b) A partir des données constructeur en document technique DT1, quel est le débit maximum effectif que la liaison UHF peut supporter ? Conclure sur la possibilité de transmettre à priori l'ensemble des données ECoG.

(c) A $f_e = 1\,\text{kHz}$, déterminer le nombre de canaux maximum que la liaison sans fil permettra d'envoyer.

(d) L'ensemble des canaux sont échantillonnés à $f_e = 1\,\text{kHz}$ et transmis. Calculer le nombre de bits maximum sur lesquels doit être codé un échantillon ?

Soit $s_n(t)$ le signal ECoG du canal n, où $n \in [\![0, 63]\!]$ et t le temps. On note s_n^* le signal échantillonné par le convertisseur analogique numérique à une fréquence f_e, et

$$s_n^*(k) = s_n\left(\frac{k}{f_e}\right)$$

le k-ième échantillon du signal s_n. On note s_n^* la séquence d'échantillons du canal n correspondant à la numérisation du signal s_n. Ce signal est converti sur N_1 bits et les valeurs de $s_n^*(k)$ appartiennent donc à un alphabet A défini par :

$$A = \{a_0, \cdots a_{2^{N_1}-1}\} = \left\{-2^{N_1-1}+1, \cdots, -1, 0, 1, 2, \cdots, 2^{N_1-1}\right\}$$

Soit $p_{i,n}$ la probabilité d'apparition du caractère a_i dans la séquence s_n^*. On a donc de manière triviale :

$$0 \leq p_{i,n} \leq 1,$$

$\forall\,(i,n) \in [\![0, 2^{N_1-1}]\!] \times [\![0, 63]\!]$, et

$$\sum_{i=0}^{2^{N_1}-1} p_{i,n} = 1$$

$\forall n \in [\![0, 63]\!]$

L'entropie H de la séquence s_n^* est définie par :

$$\mathrm{H}(s_n^*) = -\sum_{i=0}^{2^{N_1}-1} p_{i,n} \log_2(p_{i,n})$$

où l'utilisation du logarithme en base 2 permet d'avoir un résultat en bits, et par définition $p_{i,n}\log_2(p_{i,n}) = 0$ si $p_{i,n} = 0$.

Question 3. Afin d'utiliser cette définition sur les signaux ECoG, les propriétés de base de H sont explorées ici :

(a) Montrer que $0 \leq \mathrm{H}(s_n^*)$.

(b) A quelle condition sur la séquence s_n^* l'entropie $\mathrm{H}(s_n^*)$ est-elle nulle ?

(c) Dans l'hypothèse où chaque caractère de A est équiprobable dans s_n^*, que vaut l'entropie $\mathrm{H}(s_n^*)$?

(d) Montrer que $\mathrm{H}(s_n^*) \leq N_1$.

(e) Conclure sur le sens physique à donner à la quantité d'entropie $\mathrm{H}(s_n^*)$ de la séquence s_n^*.

Question 4. Il s'agit ici de quantifier l'entropie des signaux ECoG afin d'évaluer les possibilités de compression du signal. Les calculs sont réalisés avec le langage *Python* et avec l'API *Numpy*. Les bases d'utilisation de cette API sont rappelées en document technique DT2. Une séquence s_n^* est stockée dans une variable de type *numpy.array*, où chaque élément est un échantillon enregistré en sortie du convertisseur Analogique Numérique et stocké au format *np.int* compris entre $-2^{N_1-1}+1$ et 2^{N_1-1}.

(a) A partir du prototype donné ci-dessous, écrire le code de la fonction *compute_probabilities* qui renvoie une variable itérable contenant les probabilités d'apparition des caractères de A dans la séquence s_n^*. Il est possible d'utiliser la fonction *numpy.bincount* dont l'aide est donnée en document technique DT3.

```
def compute_probabilities(sequence, N_bits=12):
  """calcule la probabilite d'apparition de chaque
  mot binaire de N_bits

  Arguments
  ---------
    sequence : numpy array
      signal ECoG brut sur une electrode en sortie
      du convertisseur Analogique-Numerique
    N_bits : int
      nombre de bits du convertisseur Analogique
  """
```

(b) A partir du prototype donné ci-dessous, écrire le contenu de la fonction *compute_entropy* qui renvoie la valeur d'entropie de la séquence s_n^*.

```
def compute_entropy(signal, N_bits):
  """calcule l'entropie d'un signal,
  numerise sur un nombre de bits donne

  Arguments
  ---------
    signal : numpy array
      signal ECoG brut sur une electrode en sortie
      du convertisseurAnalogique-Numerique
    N_bits : int
      nombre de bits du convertisseur Analogique
  """
```


Tournez la page S.V.P.

Ces fonctions de calcul de l'entropie appliquées sur une base de données de signaux d'enristrement ECoG, permettent d'obtenir des valeurs rassemblées sous la forme d'un histogramme donné en figure 5.

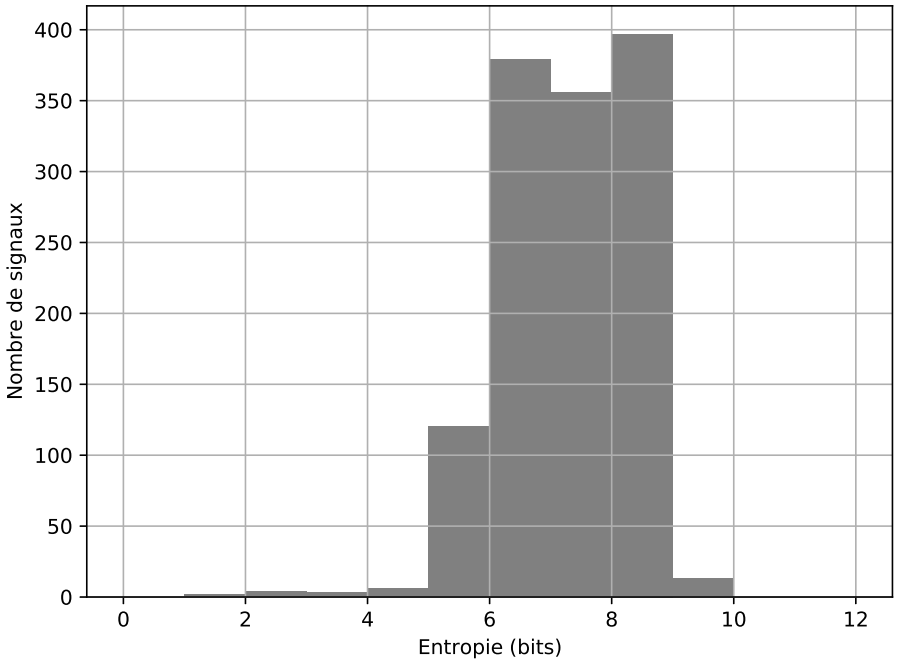


FIGURE 5 – Histogramme des entropies calculées pour une base de données de signaux ECoG

Question 5. Avec une compression sans perte, sur combien de bits est-il possible d'encoder le signal ECoG ? Ce nombre de bits peut-il permettre de transmettre l'ensemble des données ECoG sur la liaison sans fil, en respectant l'exigence EP1 du diagramme d'exigence ?

1.2 Compression spatiale par utilisation de la Transformée en Cosinus Discrète

Pour une séquence unidimensionelle s on définit la transformée en cosinus discrete S de profondeur N, abrégée DCT-1D par la suite, par :

$$S(k) = \sqrt{\frac{2}{N}} C_k \sum_{i=0}^{N-1} \cos\left(\frac{(2i+1)\,k\pi}{2N}\right) s(i)$$

$\forall k \in [\![0, N-1]\!]$, où

$$C_k = \begin{cases} \dfrac{1}{\sqrt{2}} & \text{si } k = 0 \\ 1 & \text{si } k > 0 \end{cases}$$

Il est possible d'utiliser une définition matricielle de la DCT-1D, en posant

$$\underline{S} = \begin{bmatrix} S(0) \\ \vdots \\ S(N) \end{bmatrix}, \ \underline{s} = \begin{bmatrix} s(0) \\ \vdots \\ s(N) \end{bmatrix}$$

ce qui revient à écrire

$$\underline{S} = C\underline{s}$$

où C est une matrice de taille $N \times N$ de coefficients définie par :

$$C = \begin{bmatrix} c_{0,0} & \cdots & c_{0,N-1} \\ \vdots & & \vdots \\ c_{N-1,0} & \cdots & c_{N-1,N-1} \end{bmatrix}, \ c_{k,i} = \begin{cases} \dfrac{1}{\sqrt{N}} & \text{si } k = 0, \forall i \in [\![0, N-1]\!] \\ \sqrt{\dfrac{2}{N}} \cos\left(\dfrac{(2i+1)\,k\pi}{2N}\right) & \text{si } k > 0, \forall i \in [\![0, N-1]\!] \end{cases}$$

Tournez la page S.V.P.

Question 6. Avant d'appliquer cette transformée sur le signal ECoG, les propriétés fondamentales de la DTC-1D sont explorées ici.

(a) Compléter le document réponse DR-1 avec les valeurs numériques des coefficients de C pour $N = 8$.

(b) Calculer $S(0)$ en fonction de $< s >_{0,N-1}$, la moyenne de la séquence s sur N éléments.

(c) Montrer que

$$CC^T = I_N$$

où I_N est la matrice identité de taille N. On pourra admettre que :

- N est pair,
- $\forall k \in [\![1, N-1]\!],\ \sum_{i=0}^{N-1} \cos\left(\frac{(2i+1)\,k\pi}{2N}\right) = 0$

(d) En déduire que C est orthogonale, c'est à dire que $C^{-1} = C^T$.

(e) Retrouver la DCT-1D inverse, c'est-à-dire l'expression de $s(k)$ en fonction de S.

(f) L'énergie d'une séquence est définie par

$$E_S = \| S \|^2 \triangleq \sum_{k=0}^{N-1} S(k)^2 = \underline{S}^t \underline{S}$$

montrer que la DCT-1D conserve l'énergie, c'est-à-dire que $E_S = E_s$.

Dans la suite du sujet, les données issues de la mesure ECoG sont représentées sous la forme d'un tenseur **S** à 3 dimensions comme illustré en figure 6.

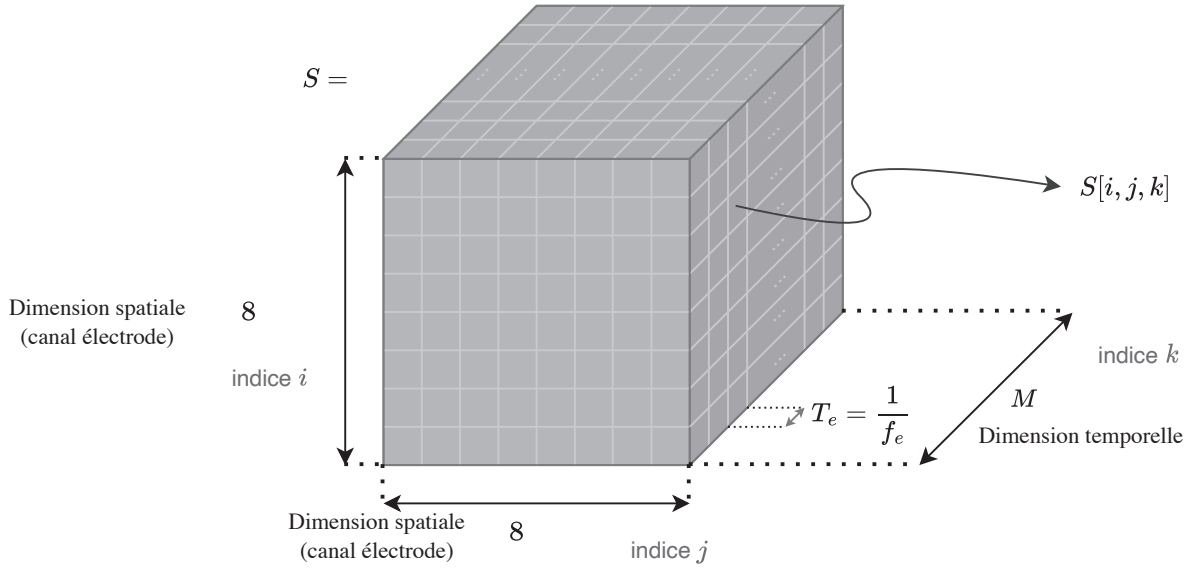


FIGURE 6 – Représentation sous forme de tenseur de dimension 3 des données ECoG, la matrice d'électrodes d'enregistrement est représentée sous forme d'une matrice carrée dont les deux premières dimensions sont spatiales, la troisième dimension représente l'évolution dans le temps

A un numéro d'échantillon temporel donné k, le signal s est donc une matrice 8×8 (= 64 canaux) qu'il est possible de traiter avec des méthodes issues du traitement de l'image. En particulier, la corrélation spatiale entre les électrodes étant forte, il est intéressant d'appliquer une Transformée en Cosinus Dicrète (*Discrete Cosine Transform (DCT)*) en 2 dimensions (DCT-2D) pour encoder le signal de manière spatiale et non temporelle. Dans le cas du signal ECoG présent, on transforme un signal contenu dans un tenseur **s**, de dimension $8 \times 8 \times M$, en un tenseur transformé par DCT-2D **S**, de dimension $8 \times 8 \times M$ avec la relation suivante :

$$\begin{aligned}\mathbf{S}\left[i,j,k\right] &= 4C_iC_j \sum_{l=0}^{7}\sum_{m=0}^{7} \cos\left(\frac{\left(2l+1\right)i\pi}{16}\right)\cos\left(\frac{\left(2m+1\right)j\pi}{16}\right)\mathbf{s}\left[l,m,k\right] \\ &= 4C_iC_j \sum_{l=0}^{7}\sum_{m=0}^{7} B_{i,j}\left[l,m\right]\mathbf{s}\left[l,m,k\right]\end{aligned}$$

où les $8 \times 8 = 64$ (indices i et j) matrices B de taille 8×8 (indices l et m) sont des fonctions de base.

La transformée inverse est donnée par la relation suivante :

$$\begin{aligned}\mathbf{s}\left[l,m,k\right] &= \frac{1}{4}\sum_{i=0}^{7}\sum_{j=0}^{7} C_iC_j \cos\left(\frac{\left(2l+1\right)i\pi}{16}\right)\cos\left(\frac{\left(2m+1\right)j\pi}{16}\right)\mathbf{S}\left[i,j,k\right] \\ &= \frac{1}{4}\sum_{i=0}^{7}\sum_{j=0}^{7} C_iC_jB_{i,j}\left[l,m\right]\mathbf{S}\left[i,j,k\right]\end{aligned}$$

Tournez la page S.V.P.

Question 7. Afin d'évaluer la sortie de la DCT-2D sur le signal ECoG, il est possible d'implémenter cette transformée en Python.

(a) Implémenter la fonction *DCT2D_matrix* prenant en argument les indices i et j et renvoyant la matrice $B_{i,j}$ dont l'entête est donné ci-dessous :

```
import numpy as np

def DCT2D_matrix(i, j, N=8):
    '''
    Construit la matrice B i,j

    Arguments:
    ----------
    i, j : int
        indices du signal temporel

    Returns:
    --------
    B : numpy.array
    '''
```

(b) Implémenter la fonction *DCT2D_all_matrix* renvoyant l'ensemble des matrices B sous la forme d'une liste de listes (lignes puis colonnes) comme indiqué dans l'entête ci-dessous :

```
import numpy as np

def DCT2D_all_matrix(N):
    '''
    Construit la liste 2D des matrices B i,j pour une profondeur
    de DCT de N

    Returns:
    --------
    mat_listoflist : list of lists
        liste (lignes) de listes (colonnes) des matrices B
        pour une profondeur de DCT de N
    '''
    mat_listoflist = []
```

(c) Implémenter la fonction *DCT2D* renvoyant une matrice correspondant à la DCT2D pour le tenseur **s** à l'échantillon k comme indiqué dans l'entête suivant :

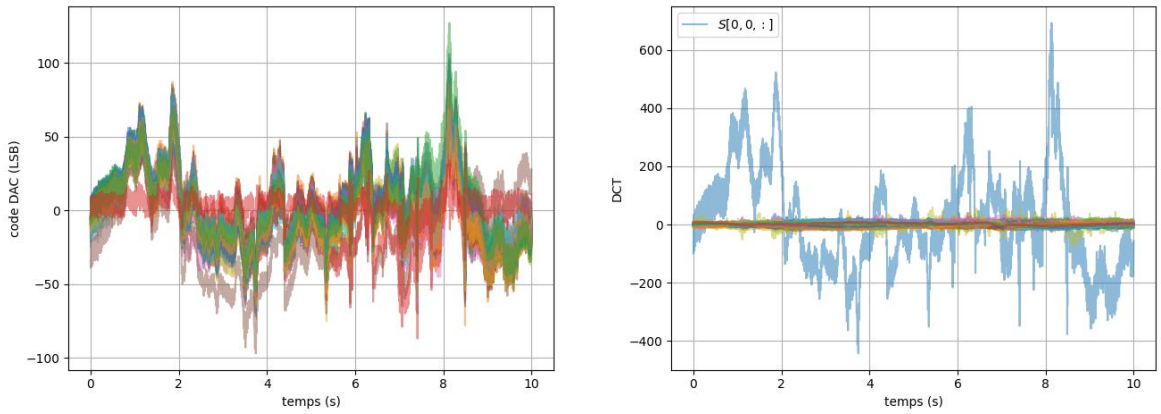
```
import numpy as np

def DCT2D(s,k,N=8):
    '''
    Calcule la DCT2D sur l'echantillon temporel
    k du tenseur s

    Arguments:
    ----------
    s : numpy array
        tenseur ECoG en domaine temporel
    k : int
        indice en temps de l echantillon
```

```
    N : int
      profondeur de DCT, par defaut a 8

    Returns:
    ----------
    S : numpy array
      image de s[:,:,k] par la DCT2D
    '''
```

Question 8. La DCT est implémentée avant l'envoi des données, donc sur le micro-contrôleur embarqué. Cette fonction n'est pas implémentée dans les faits avec le code python, cependant il est possible de se baser sur ce code pour évaluer le coût de calcul.

(a) En considérant qu'une multiplication coûte 3 périodes d'horloge, calculer le nombre de périodes d'horloge nécessaires pour le calcul de la transformée d'une image spaciale de **S**.

(b) Le signal ECoG étant toujours échantillonné à $f_e = 1\,\text{kHz}$, calculer la fréquence d'horloge minimale pour le micro-contrôleur en ne considérant que le calcul de la DCT.



(a) (b)

FIGURE 7 – Exemple de tenseur du signal ECoG brut et après application de la DCT-2D. (a) tracé de 10 secondes des canaux ECoG (un canal par trace) brut en aval de la conversion analogique numérique (b) même signal, après application de la DCT. Le canal se démarquant correspond au tracé à travers le temps pour les indices $(i, j) = (0, 0)$, l'ensemble des autres indices $(i, j) \neq (0, 0)$ apparaissent groupés et de plus faibles amplitudes. Pour ce signal et sur la base de données considérée, les valeurs de DCT pour les indices $(i, j) = (0, 0)$ sont bornées entre -4095 et 4096. Les valeurs de DCT pour les indices $(i, j) \neq (0, 0)$ sont bornées entre -127 et 128.

La figure 7 présente un exemple de tenseur ECoG **s** et le résultat **S** de l'application de la DCT-2D. En répétant l'acquisition, il est à remarquer que les valeurs de DCT-2D pour les indices $(i, j) = (0, 0)$ sont bornées entre -4095 et 4096. Les valeurs de DCT-2D pour les indices $(i, j) \neq (0, 0)$ sont bornées entre -127 et 128.

Tournez la page S.V.P.

Question 9. La DCT-2D permet de transformer l'information, les données dans le micro-contrôleur sont converties en *int* (la partie décimale n'est pas conservée) pour l'ensemble des valeurs issues du calcul de la DCT.

(a) Sur combien de bits devrait être codé le canal $(i, j) = (0, 0)$, après l'application de la DCT-2D ?

(b) Sur combien de bits devraient être codés les canaux $(i, j) \neq (0, 0)$ après l'application de la DCT-2D ?

(c) Calculer le nombre de bits moyens par échantillon.

(d) Conclure sur la possibilité du système de transmettre l'ensemble des informations ECoG enregistrables dans l'implant (exigence EF4 du diagramme d'exigence), et proposer éventuellement des solutions permettant d'améliorer l'embarquabilité du calcul de la DCT-2D dans l'implant.

Partie 2. Décodage des trajectoires à partir des enregistrements ECoG

L'objectif de cette partie est de vérifier l'extraction des intentions de mouvement (EP2 dans le diagramme d'exigence). Les données ont été envoyées par les implants *WIMAGINE* à la station de base, qui restitue les données au '*BCI PC Board*' (cf Fig. 2 (b)). Ce dispositif contient un système appelé *Online Cerebral Decoder* en charge de décoder le signal ECoG et de générer les commandes des mouvements pour le pilotage de l'exosquelette.

La notation sous forme de tenseur utilisée dans la partie précédente sera toujours utilisée. Les données ECoG sont mises sous la forme d'un tenseur non compressé et converties en μV et non plus en codes issus des convertisseurs Analogiques Numériques, il n'est pas nécessaire de considérer les unités des canaux ECoG dans cette partie. Les données $\mathbf{s}$ d'enregistrement ECoG constituent le signal d'entrée du décodage. Dans la suite du sujet, nous nous limiterons à l'enregistrement de l'activité de l'hémisphère gauche par l'implant.

On note $\mathbf{y} \in \mathbb{R}^M$ le mouvement à générer. Sur le système étudié, une première phase de *pre-processing* et d'extraction de caractéristiques (ou *features*) est appliquée, générant un vecteur noté $\mathbf{x} \in \mathbb{R}^N$. Le décodeur est déduit de données ECoG et de mouvements enregistrés sur des sujets valides [6] lors d'une première phase d'apprentissage comme illustré en Figure 8. Dans une seconde phase, les données arrivant en temps réel du tenseur ECoG passent par l'étape de *pre-processing* et d'extraction de *features*, puis sont appliquées au modèle identifié et servent de commande pour l'exosquelette après une étape de *post-processing*.

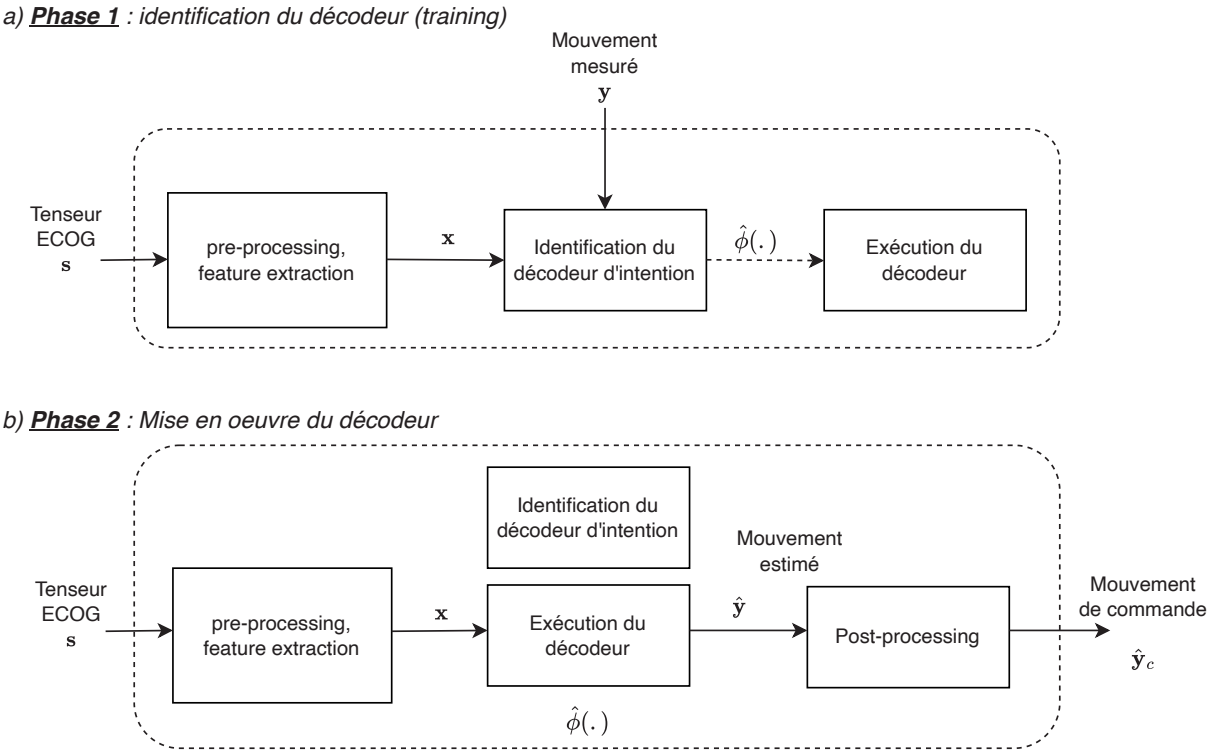


FIGURE 8 – (a) Principe de l'identification d'un décodeur à partir d'enregistrements de signaux ECoG et de mouvements mesurés (b) application du décodeur pour la prédiction d'intention de mouvement.

Tournez la page S.V.P.

2.1 Identification des *features*

Pour former un tenseur de caractéristiques, chaque signal ECoG est cartographié dans l'espace temps-fréquences par transformée en ondelettes continue (CWT). Si on considère un signal $s_n(t)$ d'une électrode n, la CWT est définie par :

$$S_{n,w}(s,\tau) = \frac{1}{\sqrt{|s|}} \int_{-\infty}^{\infty} s(t)\, \overline{\psi}\left(\frac{t-\tau}{s}\right) dt$$

où ψ est une fonction de $\mathbb{R}$ dans $\mathbb{R}$ ou $\mathbb{C}$ appelée ondelette mère ; s et τ sont respectivement les échelle et translation de l'ondelette fille. L'opération $\overline{\cdot}$ désigne la conjugaison complexe. Nous utiliserons dans la suite l'ondelette de Morlet complexe définie par :

$$\psi(t) = \frac{1}{\sqrt[4]{\pi}} e^{\jmath\omega t} e^{-\frac{t^2}{2}}$$

où $\jmath^2 = -1$, et ω est la pulsation en $\mathrm{rad} \cdot \mathrm{s}^{-1}$. La notation $*$ pourra désigner le produit de corrélation.

Question 10. Écrire une fonction *Python* nommée *morlet*, prenant en argument un vecteur t et ω par défaut à la valeur 5, et renvoyant le calcul de l'ondelette mère utilisée.

Question 11. Dans un premier temps, les propriétés nécessaires à l'utilisation de l'ondelette de Morlet sont étudiées :

(a) Calculer $\Psi(f)$ la transformée de Fourier de $\psi(t)$.

(b) Identifier la fréquence centrale de l'ondelette mère.

(c) Pour un facteur d'échelle s, quelle sera la fréquence centrale de l'ondelette fille centrée en 0 ($\tau = 0$) : $\tilde{\psi}\left(\frac{t-\tau}{s}\right) = \frac{1}{s}\psi\left(\frac{t}{s}\right)$?

La CWT introduite ci-dessus est dans le cas présent appliquée à un signal échantillonné à la période $T_e = \frac{1}{f_e}$. La formule intégrale précédente est couramment discrétisée sous la forme :

$$S_{n,w}[s,\tau] = \frac{1}{\sqrt{|s|}} \sum_{n=0}^{N-1} s_n[n]\, \overline{\tilde{\psi}}\left[\frac{\tau - n}{s}\right]$$

soit sous la forme d'un produit de convolution discret. Cette opération peut être réalisée avec la fonction *numpy.convolve* dont la documentation est donnée dans le document technique DT4.

Question 12. Calculer la complexité algorithmique en temps de la CWT discrétisée.

Question 13. Compléter le code de la fonction de la transformée en ondelette *cwt_convolve* du document réponse DR2 :

- cette fonction prend en entrée un signal échantillonné *sig*, la période d'échantillonnage T_e et un vecteur (*array_like*) contenant les fréquences où l'on cherche à identifier les *features*,
- le nombre de valeurs et les valeurs de translations τ sont identiques au nombre d'échantillons et valeurs de temps des échantillons du signal d'entrée,
- il est possible d'utiliser la fonction *morlet*, définie précédemment.

Question 14. Soit N le nombre d'échantillons et L le nombre de fréquences sur lesquelles sont extraites les *features*. Exprimer la complexité algorithmique en temps de la fonction précédente.

Les fonctions précédentes sont enregistrées dans une bibliothèque nommée *WaveletTransform*. Une bibliothèque *ecog* permet de charger un signal ECoG dans un objet. Le code suivant est exécuté :

```
import WaveletTransform as wt
import ecog
import numpy as np

# ouverture d un enregistrement ECoG de la base de donnees
# d identification du decodeur (Patient numero 1)
enregistrement = ecog.raw_ECoG('./DB/Patient_1')
T_sample = 1e-3

print(enregistrement.ecog_tensor.shape)
print(type(enregistrement.ecog_tensor[0,0,0]))

# parametres
nsec = 51            # temps de depart
tensor_width = enregistrement.ecog_tensor.shape[0]
start = int((nsec)/T_sample)
stop = int((nsec+1)/T_sample)
freqs = np.linspace(10,150,15)  # frequences pour la CWT

# extraction des features entre les secondes 51 et 52
feature_vector = np.zeros((tensor_width**2, stop-start, len(freqs)))
for i in range(enregistrement.tensor_width**2):
  sig = enregistrement.get_electrode_signal(i)[start:stop]
  cwt = np.log10(np.abs(wt.cwt_convolve(sig, freqs, T_sample)))
  feature_vector[i,:,:] = np.transpose(cwt)
```

L'exécution de ce code renvoie la sortie de console suivante :

```
(8, 8, 1045828)
<class 'numpy.float64'>
```

Question 15. Quelles sont les tailles en nombre d'éléments, puis en octets :

(a) de la variable *sig*,

(b) de la variable *cwt*,

Tournez la page S.V.P.

(c) de la variable *feature_vector*

La fonction de convolution utilisée est en majeure partie responsable du temps d'exécution de la transformée en ondelette. Les prochaines questions traitent d'une optimisation possible de ce calcul.

Soit f et g deux signaux échantillonnés. f_N désigne le signal f périodisé à période $N \in \mathbb{N}$ tel que :

$$f_N[kN+n] = f[n], k \in \mathbb{Z} \text{ et } n \in [\![0, N-1]\!]$$

La convolution cyclique est définie par :

$$(f *_N g)[n] = \sum_{m=0}^{N-1} f_N[m]\, g_N[n-m]$$

L'implémentation suivante permet la convolution cyclique en python :

```
import numpy as np

def cyclic_convolution(f,g):
  N = len(f)
  conv = np.zeros(N)

  for n in range(N):
    for m in range(N):
      if n-m >= 0:
        conv[n] += f[m]*g[n-m]
      else:
        conv[n] += f[m]*g[N+(n-m)]
  return conv
```

Question 16. Dans un premier temps, cette fonction est testée :

(a) Calculer la sortie du code suivant :

```
import numpy as np

f = np.array([1,3,2])
g = np.array([3,6,4])

print(cyclic_convolution(f,g))
```

(b) Quelle est la complexité algorithmique en temps du calcul de convolution cyclique ?

La Transformée de Fourier Discrète (abrégée DFT) est définie pour un signal échantillonné f par :

$$DFT(f)[k] = \sum_{n=0}^{N-1} f[n]\, e^{\frac{-2\jmath\pi}{N}kn}$$

Cette opération est en particulier réalisée par la FFT (*Fast Fourier Transform*) avec une complexité algorithmique en temps de $\mathcal{O}(N \log N)$. La transformée inverse, ou IDFT, est réalisée par la IFFT (*Inverse FFT*) avec la même complexité.

Question 17. Cette question vise à évaluer l'utilisation de la FFT pour calculer la convolution cyclique :

(a) Montrer que $\forall m \in [\![0, N-1]\!]$:

$$DFT\left(f_N\right)[k] = \sum_{n=0}^{N-1} f_N[n-m]\, e^{\frac{-2\jmath\pi}{N}k(n-m)}$$

(b) Montrer que

$$DFT\left(f *_N g\right)[k] = DFT\left(f_N\right)[k] \cdot DFT\left(g_N\right)[k]$$

(c) En déduire une méthode de calcul de $(f *_N g)$.

(d) Quelle sera la complexité de cette méthode de calcul ?

Question 18. Il est donc possible d'implémenter une autre méthode de calcul de la convolution cyclique :

(a) Proposer une implémentation du calcul de convolution cyclique à partir de l'entête de code suivante :

```
import numpy.fft.fft as fft
import numpy.fft.ifft as ifft

def FFT_cyclic_convolution(f,g):
```

(b) Calculer explicitement la sortie du code suivant :

```
import numpy as np

f = np.array([1,3,2])
g = np.array([3,6,4])

print(FFT_cyclic_convolution(f,g))
```

Comme rappelé dans la documentation *Numpy*, la définition première de la convolution discrète impose d'avoir des signaux définis pour $n \in \mathbb{Z}$:

$$(f * g)[n] = \sum_{m=-\infty}^{+\infty} f[m]\, g[n-m]$$

Dans le problème traité, les signaux f ont des échantillons pour $n \in [\![0, N-1]\!]$. Soit f_P le signal défini par *zero padding* :

$$f^P[n] = \begin{cases} f[n] & \text{si } 0 \leq n < N-1 \\ 0 & \text{si } N \leq n < P-1 \end{cases}$$

avec $P \in \mathbb{N}$ et $n \in [\![0, P-1]\!]$

Question 19. Calculer la valeur minimum de P telle que

$$\forall n \in [\![0, P-1]\!], \left(f^P * g^P\right)[n] = \left(f^P *_P g^P\right)[n]$$

Tournez la page S.V.P.

Question 20. L'utilisation de ce résultat permet de remplacer l'appel coûteux en temps à la fonction *numpy.convolve* :

(a) Écrire le code de la fonction *fast_ convolution* dont l'entête est donné ci-dessous, en réutilisant la fonction de convolution cyclique avec FFT :

```
1 import numpy as np
2 import numpy.fft.fft as fft
3 import numpy.fft.ifft as ifft
4
5 def fast_convolution(f,g):
```

(b) Calculer la complexité algorithmique en temps de cette fonction.

(c) Le code suivant est implémenté et exécuté :

```
1 import numpy as np
2
3 f = np.array([1,3,2])
4 g = np.array([3,6,4])
5
6 print(np.convolve(f,g,'same'))
7 print(np.convolve(f,g,'full'))
8 print(np.fast_convolution(f,g))
```

Les sorties correspondant aux lignes 6 et 7 sont :

```
1 [3 15 28 24  8]
2 [15 28 24]
```

Calculer la sortie de la ligne 8

(d) En fonction de N la taille d'origine de f et g comment indexer le résultat de *fast_ convolution* pour obtenir un vecteur de taille N ?

Question 21. Compléter le code de calcul de la transformée en ondelette du document réponse DR3.

Question 22. Toujours avec N le nombre d'échantillons et L le nombre de fréquences sur lesquelles sont extraites les *features*., exprimer la complexité algorithmique en temps de la fonction précédente. Comparer cette complexité à la première implémentation de la CWT et conclure sur le gain apporté par le changement d'implémentation au regard du volume des données traitées.

2.2 Décodage des intentions de mouvement

Comme décrit en figure 8, le décodage des intentions de mouvement consiste à identifier dans une phase de *training* une fonction $\hat{\phi}$ capable de reconstruire au mieux un mouvement mesuré $\mathbf{y} \in \mathbb{R}^M$ (vecteur d'observation) à partir des *features* extraites $\mathbf{x} \in \mathbb{R}^N$. On peut écrire :

$$\hat{\mathbf{y}} = \hat{\phi}(\mathbf{x})$$

où $\hat{\mathbf{y}} \in \mathbb{R}^M$ est le mouvement estimé, et $\hat{\phi}$ l'estimation de la fonction ϕ. Pour estimer la fonction ϕ lors de la phase d'entrainement, les mesures permettent de disposer de n vecteurs $\mathbf{x}$ et $\mathbf{y}$ mis sous forme de matrices notées respectivement $\mathbf{X} \in \mathbb{R}^{n \times N}$ et $\mathbf{Y} \in \mathbb{R}^{n \times M}$.

Le vecteur de *features* $\mathbf{x}$, correspond à une durée (ou *epoch*) de signal ECoG de 1 seconde (toujours échantillonné à 1 kHz). Ces données sont ensuite passées en entrée de la CWT étudiée à la partie précédente avec pour bande de fréquence 10 Hz à 150 Hz et avec un pas de fréquence de $\delta f = 10$ Hz. Le résultat de la CWT étant un nombre complexe, la valeur du module est prise pour résultat. Les données fréquentielles sont ensuite sous-échantillonnées d'un facteur 100 par moyennage. L'étape de filtrage des artefacts de mouvement visible dans la figure 9 n'a pas de conséquence sur la taille des données en entrée et ne sera pas considérée dans ce sujet.

Les n vecteurs $\mathbf{x}$ sont générés sur une période de *training* de 1400 s en déplaçant successivement la fenêtre de l'*epoch* d'un pas $\delta t = 0.1$ s.

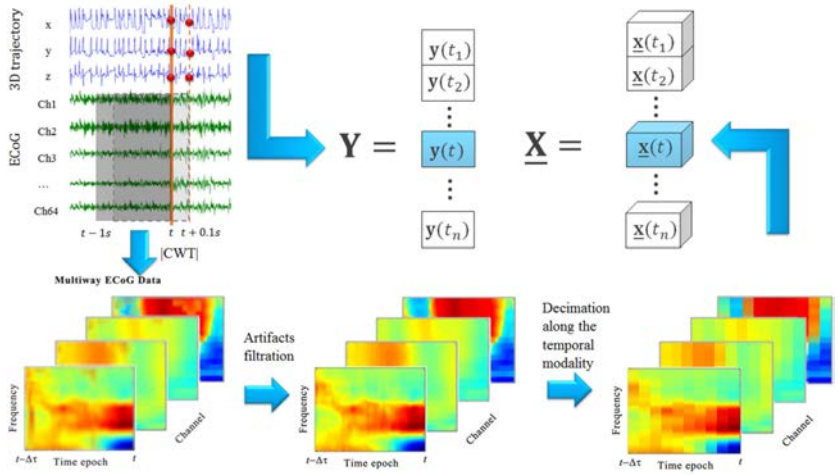


FIGURE 9 – *Mapping* spatial, fréquentiel et temporel des données d'enregistrement ECoG pour le décodage des intentions de mouvement [7].

Question 23. L'objectif de cette question est d'estimer les tailles des vecteurs d'entrée lors de la phase de *training*.

(a) Calculer la taille d'un vecteur de *features* $\mathbf{x}$.

(b) A partir de la taille d'une *epoch* et le pas δt, calculer le nombre n de vecteurs de *features*.

Tournez la page S.V.P.

(c) En déduire la taille puis le nombre d'éléments de la matrice $\mathbf{X}$ pour la phase de *training*.

(d) Les données sont constituées de *float64*. Calculer le poids mémoire de la matrice $\mathbf{X}$ lors de la phase de *training*.

Question 24. Le tenseur du signal brut ECoG est mis comme attribut d'un objet de type *ecog* défini en *Python*. Un diagramme de la classe ainsi que les prototypes des méthodes disponibles sont donnés en document technique DT5.

(a) Écrire une méthode *get_ecog_epoch* qui renvoie un tenseur (tableau à 3 dimensions) pour l'*epoch* commençant au temps $'t'$.

(b) Écrire une méthode *compute_features* qui pour un temps $'t'$ calcule et renvoie le vecteur de *feature* $\mathbf{x}$. Il est possible d'utiliser la fonction *Numpy.ravel* comme rappelé dans le document technique DT2 pour transformer le tenseur en vecteur.

Les modèles les plus simples permettant une régression des données sont linéaires. Dans ce cas, la fonction ϕ peut s'écrire :

$$\phi(\mathbf{X}) = \mathbf{BX} = \sum_{i=1}^{n} \beta_i \mathbf{x_i}$$

avec $\mathbf{B} = (\beta_1, \cdots, \beta_n)^T$ respectivement une matrice ou des vecteurs de poids, et $\mathbf{x_i}$ un vecteur de *feature* indexé i parmi les n constituant $\mathbf{X}$.

Identifier le modèle revient à trouver la matrice $\mathbf{B}$ dans ce cas. Une première méthode par les moindres carrés ordinaires, ou *Ordinary Least Square* (noté OLS dans la suite du sujet) permet l'évaluation de cette matrice par minimisation de la quantité $\mathbf{J}$ définie par :

$$\mathbf{J} = \|\mathbf{Y} - \phi(\mathbf{X})\|_2 = (\mathbf{Y} - \mathbf{BX})^T (\mathbf{Y} - \mathbf{BX})$$

Question 25. Pour la méthode OLS :

(a) Montrer que

$$\frac{\partial \mathbf{J}}{\partial \mathbf{B}} = 2(\mathbf{Y} - \mathbf{XB})^T (-\mathbf{X})$$

(b) En déduire une solution analytique $\hat{\mathbf{B}}$ permettant d'évaluer le modèle au sens des moindres carrés.

La méthode OLS pose cependant deux problèmes majeurs : les données d'observation sont particulièrement grandes et également fortement corrélées entre elles. La régression sur un modèle linéaire se fait dans ce cas en utilisant des moindres carrés partiels ou *Partial Least Squares* (notés PLS dans la suite du document). Il est possibe de décomposer les matrices $\mathbf{X}$ et $\mathbf{Y}$:

$$\begin{cases} \mathbf{X} = & \sum_{f=1}^{F} \mathbf{t}_f \mathbf{p}_f^T + \mathbf{E} = \mathbf{TP}^T + \mathbf{E} \\ \mathbf{Y} = & \sum_{f=1}^{F} \mathbf{u}_f \mathbf{q}_f^T + \mathbf{F} = \mathbf{UQ}^T + \mathbf{F} \end{cases}$$

où les F vecteurs $\mathbf{p}$ et $\mathbf{q}$ sont les F composantes principales de $\mathbf{X}$ et $\mathbf{Y}$, et les vecteurs $\mathbf{t}$ et $\mathbf{u}$ sont les F projections de $\mathbf{X}$ et $\mathbf{Y}$ sur ces composantes. Cette projection s'écrit de manière matricielle avec $\mathbf{T}$ et $\mathbf{U}$ les matrices de score, $\mathbf{P}$ et $\mathbf{Q}$ les matrices de charge et $\mathbf{E}$ et $\mathbf{F}$ les matrices de résidu de respectivement $\mathbf{X}$ et $\mathbf{Y}$. Cette décomposition est illustrée en figure 10. Les matrices $\mathbf{T}$ et $\mathbf{U}$ sont choisies afin de maximiser leur covariance et ainsi de projeter les *features* et observations dans une base permettant de faciliter la régression.

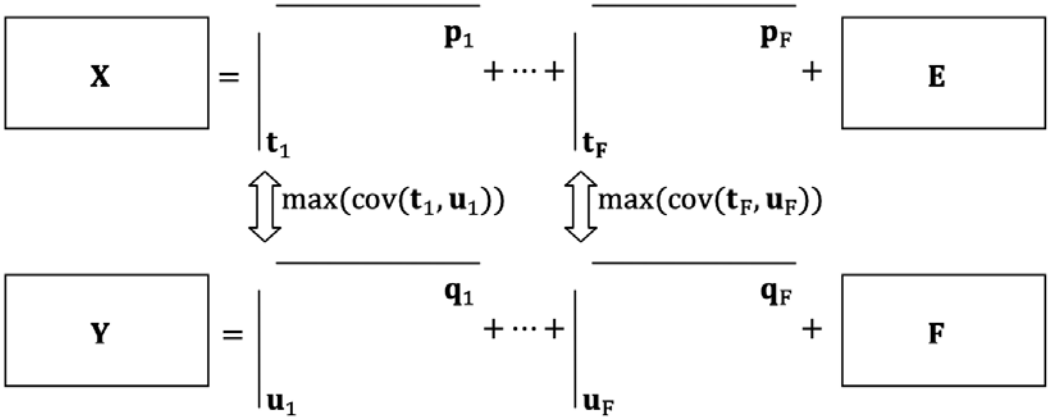


FIGURE 10 – Représentation schématisée des matrices $\mathbf{X}$ et $\mathbf{Y}$ comme étant la somme de scores ($\mathbf{t}$ et $\mathbf{u}$) sur des vecteurs de composantes principales ($\mathbf{p}$ et $\mathbf{q}$), ici au nombre de F et de matrices de résidus $\mathbf{E}$ et $\mathbf{F}$. La décomposition pour les PLS, contrairement à une décomposition par composantes principales simple a pour but de maximiser la corrélation entre les scores $\mathbf{t}$ et $\mathbf{u}$ pour dégager des composantes explicatives du modèle ϕ. Cette figure et les notations sont tirées de [7], attention la notation F désigne un entier, et les vecteurs $\mathbf{q}$ sont indicés de 1 à F, $\mathbf{F}$ désigne une matrice.

Le modèle peut alors s'écrire :

$$\mathbf{Y} = \phi(\mathbf{X}) = \mathbf{TBQ}^T + \mathbf{F}$$

Question 26. Si les matrices $\mathbf{T}$, $\mathbf{U}$, $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{E}$ et $\mathbf{F}$ sont identifiées, exprimer la solution analytique permettant d'identifier $\mathbf{B}$ au sens des moindres carrés.

Tournez la page S.V.P.

La décomposition est réalisée avec un algorithme itératif développé pour l'extraction de composantes principales nommé NIPALS (*Non linear Iterative PArtial Least Square*). Cet algorithme est synthétisé en figure 11. Afin de simplifier l'écriture, on notera $\mathbf{Y}_k$ la k-ième colonne de la matrice $\mathbf{Y}$ et de manière similaire $\mathbf{X}_k$ la k-ième colonne de la matrice $\mathbf{X}$ avec $k \in [\![1, n]\!]$.

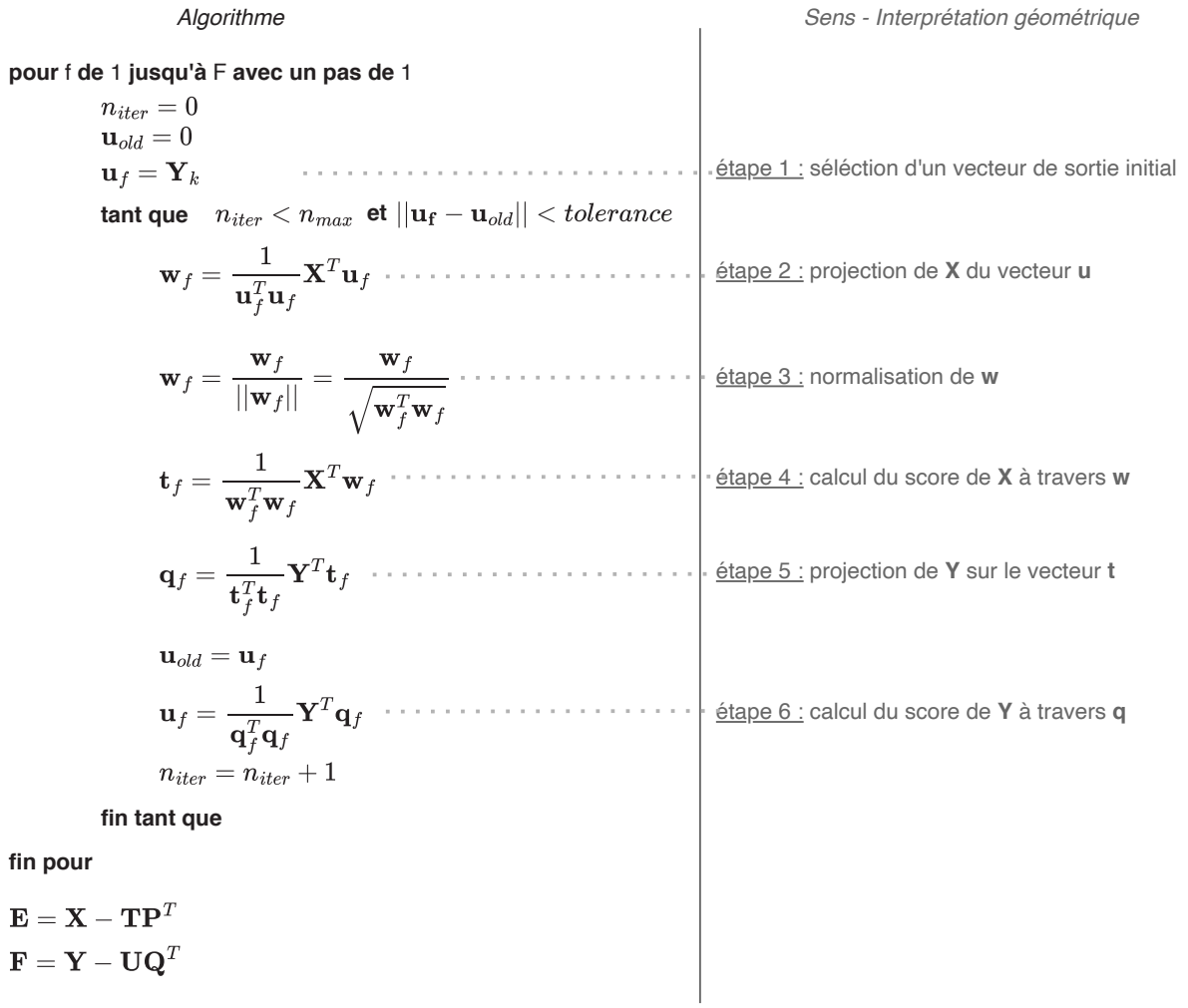


FIGURE 11 – Pseudo-code et éléments d'interprétation géométrique de l'algorithme NIPALS

Question 27. Compléter le tableau correspondant au diagramme du document réponse DR4 à partir de l'algorithme NIPALS. Les numéros d'étapes permettant de répondre sont notés en figure 11.

Question 28. Une classe *PLS* destinée à faire la décomposition et trouver la matrice $\mathbf{B}$ est implémentée. Cette classe possède une méthode *fit* qui intègre l'algorithme NIPALS et identifie les matrices $\mathbf{T}$, $\mathbf{U}$, $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{E}$, $\mathbf{F}$ et $\mathbf{B}$. Le code, incomplet est donné en document technique DT6. Les parties à compléter sont écrites dans le document réponse DR5.

(a) Compléter les calculs des variables dans la boucle d'itération à partir de l'algorithme et du diagramme vu en question précédente. (lignes 15, 17, 21, 24, 27 du DR5)

(b) Compléter le calcul de la matrice $\mathbf{B}$ à la ligne 48 du DR5.

Question 29. Proposer le code *Python* d'une méthode *eval* qui pour un vecteur de *features* $\mathbf{x}$ calcule le mouvement estimé $\hat{\mathbf{y}}$. Cette méthode sera utilisée dans la phase de mise en oeuvre du décodeur d'intention de mouvement.

Le modèle PLS et d'autres modèles plus complexes, non étudiés en détail ici, ont été mis en oeuvre dans le cadre du développement de l'exosquellette. Le document technique DT7 montre des résultats obtenus par CLINATEC pour l'estimation des coordonnée de mouvement du poignet. Les modèles utilisés sont :

- un filtre de Kalman,
- PLS,
- *Multiway (N-way) PLS* : NPLS,
- *Penalized regression* : SNPLS,
- *Polynomial Penalized PLS* : PNPLS.

Question 30. À partir des résultats quantitatifs présentés en document technique DT7 et d'élements qualitatifs (bruit, rapidité), quels modèles apparaissent comme performants et quelles sont les limites de l'algorithme PLS étudié en détail ?

Les données prédites par le modèle sont ensuite soumises à une étape de *post-processing* avant de servir de signal de contrôle de l'exosquelette, comme étudié dans la partie suivante.

Tournez la page S.V.P.

Partie 3. Commande de l'exosquelette

L'objectif de cette partie est de positionner adéquatement les segments de l'exosquelette (objectif DB1.EF6 du diagramme d'exigences). Pour le pilotage de l'exosquelette, deux systèmes sont utilisés : l'EMM (*EMY Motion Manager*) et l'EMC (*EMY Motion Controller*). L'EMM s'occupe de la gestion haut-niveau de l'exosquelette. Ceci est nécessaire car l'exosquelette doit marcher seul, une fois qu'il a reçu l'ordre du patient d'avancer ou de s'arrêter. L'EMC s'occupe quant à lui de l'asservissement de chaque axe. La consigne angulaire est calculée par l'EMM, et devient une entrée de l'EMY, qui gère l'asservissement des axes.

3.1 Motion manager

Les jambes et les bras du robot constituent une structure série (tant que le pied ou la main ne touche rien) avec des liaisons rotoïdes. Le paramétrage de chaque liaison, ou articulation, introduit des coordonnées articulaires. La connaissance de ces coordonnées articulaires permet de trouver la position des effecteurs de bout de chaîne (pied ou main). On parle de l'attitude (*pose*, en anglais) des différents corps les uns par rapport aux autres. Elle est décrite par un vecteur position entre un point du corps et le référentiel (3 distances), ainsi que par l'orientation spatiale (3 angles) pour obtenir les 6 degrés de liberté. Le passage des coordonnées articulaires aux coordonnées opérationnelles s'appelle le modèle géométrique direct et se calcule avec les lois de la cinématique. Des difficultés apparaissent lorsque l'on veut calculer les coordonnées articulaires nécessaires pour suivre une certaine trajectoire du pied (ou de la main) : il faut établir la fonction inverse, appelée modèle géométrique indirect.

Une généralisation des matrices de rotation, ou matrices homogènes, est classiquement utilisée, permettant une intégration de la translation à la matrice de rotation. Ceci permet de transformer l'addition vectorielle nécessaire pour décrire la translation en une multiplication combinée avec les rotations, au prix d'une augmentation de l'ordre de la matrice. Dans ce sujet nous ne travaillerons pas sur les matrices homogènes et nous nous limiterons dans notre étude uniquement à l'orientation, et plus particulièrement au paramétrage des orientations dans l'espace et à l'intérêt des quaternions par rapport aux angles d'Euler.

3.1.1 Angles d'Euler et problème de singularité.

Dans la littérature, les angles proposés par Euler en 1770 pour représenter l'orientation des solides rigides dans l'espace ne sont pas définis toujours de la même façon. Il existe les formulations *roll-yaw-roll*, *roll-pitch-roll* et *roll-pitch-yaw* (roulis-tangage-lacet). C'est cette dernière qui est plutôt utilisée en robotique, et on parle alors parfois d'angles de Cardan. On peut alors décrire la rotation totale $\mathbf{R}(\varphi, \theta, \psi)$ en utilisant des matrices de rotation de $\mathbb{R}^3$ représentant la rotation de chaque angle (aussi appelées matrices aux cosinus directeurs), par une multiplication :

$$\mathbf{R}(\varphi, \theta, \psi) = \underbrace{e^{\psi \mathbf{k}\wedge}}_{\mathbf{R}_\psi} \cdot \underbrace{e^{\theta \mathbf{j}\wedge}}_{\mathbf{R}_\theta} \cdot \underbrace{e^{\varphi \mathbf{i}\wedge}}_{\mathbf{R}_\varphi}$$

où $\mathbf{i} = (1,0,0)^{\mathrm{T}}, \mathbf{j} = (0,1,0)^{\mathrm{T}}, \mathbf{k} = (0,0,1)^{\mathrm{T}}$.

Après développement on obtient :

$$\begin{pmatrix} \cos\theta\cos\psi & -\cos\varphi\sin\psi+\sin\theta\cos\psi\sin\varphi & \sin\psi\sin\varphi+\sin\theta\cos\psi\cos\varphi \\ \cos\theta\sin\psi & \cos\psi\cos\varphi+\sin\theta\sin\psi\sin\varphi & -\cos\psi\sin\varphi+\sin\theta\cos\varphi\sin\psi \\ -\sin\theta & \cos\theta\sin\varphi & \cos\theta\cos\varphi \end{pmatrix}$$

où les colonnes correspondent respectivement à $\mathbf{i}_1|_{\mathbf{R}_0}$, $\mathbf{j}_1|_{\mathbf{R}_0}$, $\mathbf{k}_1|_{\mathbf{R}_0}$ et où les angles φ, θ, ψ sont les angles d'Euler.

Question 31. Montrer que lorsque $\cos\theta = 0$ nous avons

$$\frac{d\mathbf{R}}{d\psi} = -\frac{d\mathbf{R}}{d\varphi}.$$

Ceci correspond à une singularité, parfois appelée blocage de Cardan, indiquant que nous perdons un degré de liberté et que nous ne pouvons plus bouger dans toutes les directions de l'espace.

Plutôt que d'utiliser des matrices de rotation, il est possible d'exprimer l'orientation spatiale d'un solide dans l'espace comme la rotation d'un certain angle autour d'un vecteur de l'espace. Si l'axe de rotation est donné par un vecteur (trois paramètres), la longueur de ce vecteur est redondante. Nous avons donc quatre paramètres, le vecteur de l'axe et l'angle de rotation, et une contrainte, limitant par exemple la longueur du vecteur à 1. Nous retrouvons ainsi les trois paramètres libres de l'orientation spatiale.

Des formules pour passer de la représentation (axe/angle) à la matrice de cosinus directeurs et vice-versa sont données dans la littérature [1]. Pour une rotation d'un angle ϑ autour de l'axe $[x, y, z]^{\mathrm{T}}$ avec $\left\|[x, y, z]^{\mathrm{T}}\right\| = 1$, on obtient la matrice :

$$(1-\cos\vartheta)\begin{bmatrix} xx & xy & xz \\ xy & yy & yz \\ xz & yz & zz \end{bmatrix} + \cos\vartheta\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \sin\vartheta\begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}$$

La formule suivante nous permet l'inverse, donc de trouver l'axe de rotation et l'angle à partir d'une matrice $\mathbf{R}$, et ce plus directement que par le vecteur propre :

$$\text{avec } \mathbf{R} = \begin{bmatrix} \mathrm{a} & \mathrm{d} & \mathrm{g} \\ \mathrm{b} & \mathrm{e} & \mathrm{h} \\ \mathrm{c} & \mathrm{f} & \mathrm{i} \end{bmatrix} \text{ on obtient l'axe } \begin{bmatrix} \mathrm{x} \\ \mathrm{y} \\ \mathrm{z} \end{bmatrix} = \frac{1}{2\sin(\vartheta)}\begin{bmatrix} \mathrm{f}-\mathrm{h} \\ \mathrm{g}-\mathrm{c} \\ \mathrm{b}-\mathrm{d} \end{bmatrix} \text{ et l'angle } \vartheta$$

$$\cos(\vartheta) = \frac{1}{2}(\mathrm{tr}(R)-1)$$

$$\sin(\vartheta) = \frac{1}{2}\sqrt{(f-h)^2+(g-c)^2+(b-d)^2}$$

Question 32. Justifier pourquoi l'expression de $\sin(\vartheta)$ n'est pas unique. Que se passe-t-il pour l'expression de l'axe si $\vartheta = 0$?

Tournez la page S.V.P.

Ces deux inconvénients disparaissent de façon élégante en employant des quaternions.

3.1.2 Quaternions

Les quaternions sont une généralisation des nombres complexes à des nombres à quatre dimensions. Ces nombres hypercomplexes contiennent une partie réelle scalaire λ_0 et trois parties imaginaires $[\lambda_1, \lambda_2, \lambda_3]^{\mathrm{T}}$ qui sont interprétées comme partie vectorielle $\underline{\lambda}$. Le quaternion Q est donc le quadruple :

$$Q = \{\lambda_0, \lambda_1, \lambda_2, \lambda_3\} = \{\lambda_0, \underline{\lambda}\}$$

et peut également s'écrire :

$$Q = \lambda_0 + \imath\lambda_1 + \jmath\lambda_2 + k\lambda_3$$

avec $\imath^2 = \jmath^2 = k^2 = \imath\jmath k = -1$ les imaginaires purs. Les produits de ces imaginaires purs entre eux décrivent des rotations directes sur la partie vectorielle :

$$\begin{cases} \imath\jmath & = k \\ \jmath k & = \imath \\ k\imath & = \jmath \end{cases}$$

et des rotations indirectes :

$$\begin{cases} \jmath\imath & = -k \\ k\jmath & = -\imath \\ \imath k & = -\jmath \end{cases}$$

La direction de l'axe de rotation $[\mathrm{x}, \mathrm{y}, \mathrm{z}]^{\mathrm{T}}$ peut donc être donnée par le vecteur $\underline{\lambda} = [\lambda_1, \lambda_2, \lambda_3]^{\mathrm{T}}$.

L'angle de rotation ϑ est introduit de la façon suivante dans le quaternion Q :

$$\lambda_0 = \cos(\vartheta/2) \quad \text{et} \quad \underline{\lambda} = \sin(\vartheta/2)[x, y, z]^{\mathrm{T}}, \|\mathrm{x}, \mathrm{y}, \mathrm{z}\| = 1$$

Les rotations sont donc représentées par des quaternions unitaires :

$$\lambda_0{}^2 + \lambda_1{}^2 + \lambda_2{}^2 + \lambda_3^2 = 1$$

Une propriété intéressante des quaternions unitaires est que :

$$Q^{-1} = \bar{Q} = \lambda_0 - \imath\lambda_1 - \jmath\lambda_2 - k\lambda_3$$

Il est possible de calculer le résultat de la rotation d'angle ϑ autour d'un axe $[\mathrm{x}, \mathrm{y}, \mathrm{z}]^{\mathrm{T}}$ d'un vecteur $[\mathrm{a}, \mathrm{b}, \mathrm{c}]^{\mathrm{T}}$ vers $[\mathrm{a}', \mathrm{b}', \mathrm{c}']^{\mathrm{T}}$ en posant les quaternions :

$$\begin{cases} P &= \imath \mathrm{a} + \jmath \mathrm{b} + k \mathrm{c} \\ P' &= \imath \mathrm{a}' + \jmath \mathrm{b}' + k \mathrm{c}' \end{cases}$$

au moyen de la relation :

$$P' = Q^{-1} P Q$$

Question 33. Une classe *Quaternion* est écrite en *Python*, on souhaite y ajouter la possibilité de multiplication.

(a) Soient les quaternions

$$\begin{cases} Q_A = 1 + \jmath + k \\ Q_B = 2 + 2\imath + 3\jmath + 2k \end{cases}$$

Calculer $Q_A \cdot Q_B$ puis $Q_B \cdot Q_A$

(b) De manière plus générale, soient :

$$\begin{cases} Q_A = \lambda_{0A} + \imath\lambda_{1A} + \jmath\lambda_{2A} + k\lambda_{3A} \\ Q_B = \lambda_{0B} + \imath\lambda_{1B} + \jmath\lambda_{2B} + k\lambda_{3B} \end{cases}$$

Calculer explicitement le produit $Q_A \cdot Q_B$ en fonction des λ_{0A}, λ_{0B}, λ_{1A}, λ_{1B}, λ_{2A}, λ_{2B}, λ_{3A} et λ_{3B}.

(c) Il est possible d'ajouter en python les opérations algébriques pour des objets. Ecrire le code de la méthode *__mul__* qui a pour argument l'objet (*self*) et un autre nombre (réel, complexe ou quaternion) nommé *other*, qui sera appelée lors de l'exécution du code *self*other*.

Question 34. Une bibliothèque de code (incomplet) permettant de réaliser les rotations est proposée en document technique DT8. Compléter le diagramme UML des classes en document réponse DR6.

Question 35. Compléter le code de la méthode d'initialisation de la classe *Rotation3D* en document réponse DR7.

Question 36. Ajouter une méthode *__call__* à la classe *Rotation3D* prenant en argument les coordonnées *a, b, c* d'un vecteur et renvoyant les coordonnées résultant de sa rotation par l'angle et l'axe définis dans l'objet de type *Rotation3D*.

Tournez la page S.V.P.

3.2 Motion Controller

Une étude dynamique (équations du mouvement) a été établie pour le dimensionnement de la mécanique, la conception de l'entraînement, et finalement le contrôle du robot. Ce contrôle nécessite de mettre en place un asservissement au niveau de chaque axe, dont la consigne a été établie par le Motion Manager : nous allons étudier la modélisation de l'un de ces asservissements.

3.2.1 Modélisation du comportement fréquentiel

Nous commencons par étudier la stabilité du système, en s'intéressant au comportement fréquentiel de sa Fonction de Transfert en Boucle Ouverte (notée FTBO par la suite). Tout d'abord nous nous s'intéresserons au calcul de sa phase, pour ensuite tracer son diagramme de Black, et établir un calcul de la marge de stabilité du système. Le code permettant le tracé du diagramme de Bode est fourni dans le document technique DT9.

On remarque que la fonction angle renvoie une valeur entre -180° et +180° (ceci est dû à l'utilisation de la fonction arctangente), ce qui crée des discontinuités dans la courbe représentant la phase, comme illustré dans le graphique de la figure 12 représentant le gain et la phase brute. Un traitement permettant d'éviter les discontinuités de la phase a permis d'obtenir le 3ème graphique représentant la phase corrigée.

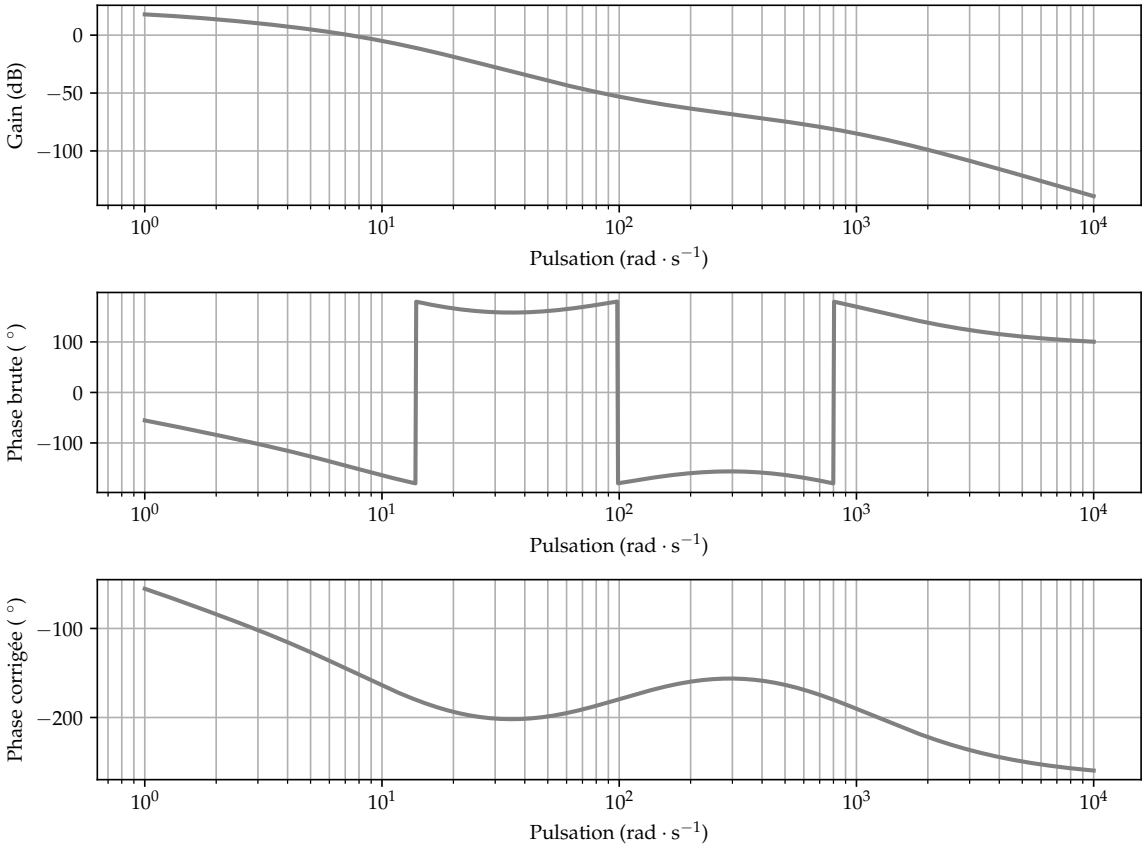


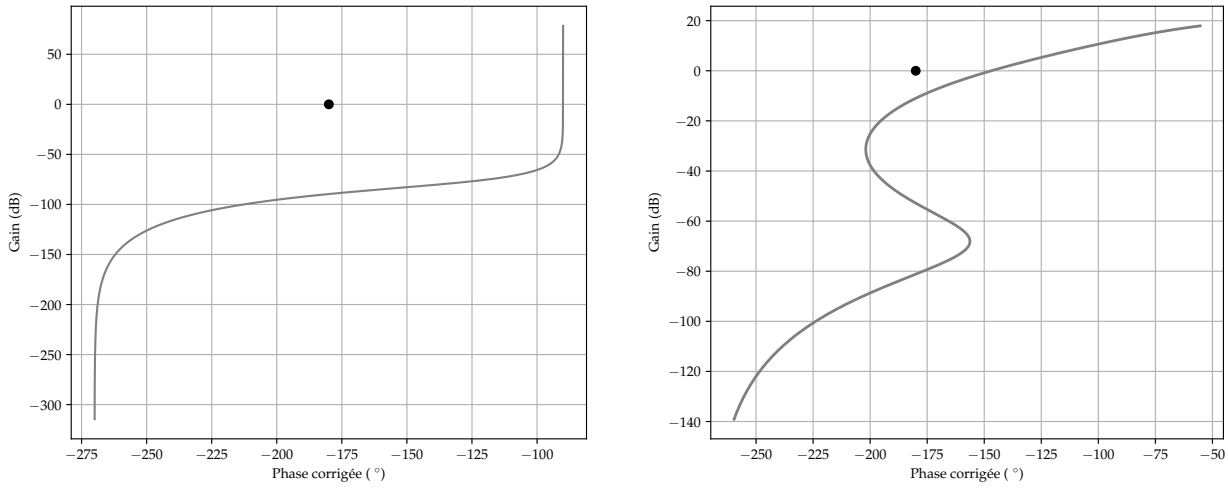
Figure 12 – Exemple de diagramme de Bode d'un système asservi présentant une discontinuité au calcul de la phase.

Question 37. Ce problème est traité par la librairie *python-control* qui implémente des fonctions d'analyse et d'aide à la conception d'asservissement, avec la fonction *unwrap* qui a été partiellement recopiée aux lignes 14 et 19 dans le code fourni dans le document technique DT9.

(a) A partir du code donné en document technique DT9, compléter les valeurs des variables demandées en document réponse DR8.

(b) A partir du code donné en document technique DT9, écrire le code de la ligne 19 permettant d'obtenir la phase sans discontinuité.

Question 38. Proposer une suite au code présenté en DT9 traçant le diagramme de Black (Gain en fonction de la phase) de la FTBO comme illustré en figure 13(a).



(a) (b)

FIGURE 13 – (a) Exemple de diagramme de Black d'un système asservi. (b) Diagramme de Black du système étudié. Le point tracé correspond au lieu d'instabilité.

Question 39. Évaluation des marges de stabilité.

(a) Écrire une fonction *dicho_tableau(F, v)* retournant les indices qui encadrent la valeur v dans un tableau trié F par dichotomie.

(b) Donner le code d'une ligne de commande permettant de trouver la marge de phase $M_\phi = Arg\left(FTBO\left(\jmath\omega_{0dB}\right)\right) + 180°$ pour le système de la figure 13 (a).

(c) Faire de même pour la marge de gain $M_G = -20\log\left|FTBO\left(\jmath\omega_{-180°}\right)\right|$.

Question 40. Le tracé de Black du système étudié par la suite est représenté figure 13 (b).

(a) Justifier qu'une méthode similaire n'est pas utilisable pour trouver une des marges.

(b) Enfin, qu'en est-il du cas où il y aurait une résonance (le gain augmente avant de redescendre) ?

Tournez la page S.V.P.

3.2.2 Simulation de la réponse temporelle et des performances dynamiques du système

On s'intéresse au réglage optimal de l'asservissement de l'articulation de l'exosquelette. La première étape est la mesure des performances. Pour dépasser les blocages précédents, on va construire une nouvelle modélisation de la commande, qui nous permettra d'évaluer la stabilité, puis nous mettrons en place des outils d'évaluation des performances et enfin des outils de réglage du correcteur.

La commande en position angulaire de l'articulation est asservie pour compenser les perturbations (effets de la gravité, chocs externes, frottements internes). Soit θ_C l'angle de consigne élaboré par le motion manager, et θ_S l'angle réellement obtenu, tous les deux en radians. C_r en N.m représente le couple résistant exercé par l'extérieur sur l'axe.

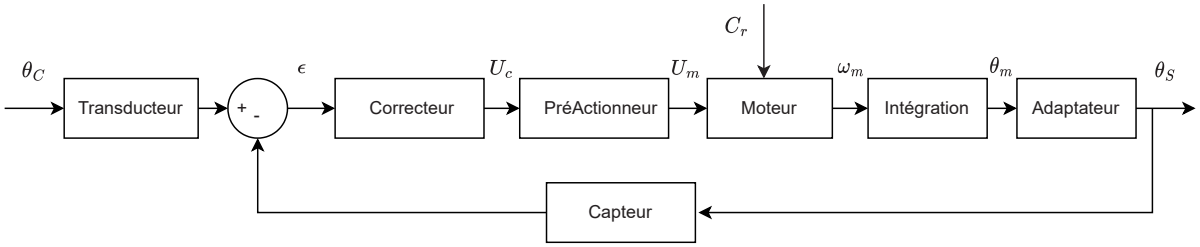


FIGURE 14 – Schéma bloc de l'asservissement

L'image de la consigne de position θ_C est comparée à l'image de la position réelle du bras mesurée par un capteur de position absolu. L'écart généré ϵ est adapté via un correcteur de fonction de transfert $C(p)$ pour donner une tension U_c :

$$C\left(p\right) = K_d \left(1 + \frac{1}{\tau_i \cdot p} + \frac{\tau_d \cdot p}{1 + \frac{\tau_d}{N} p}\right)$$

où l'action dérivée est filtrée avec un filtre passe-bas du premier ordre de constante de temps τ_d/N de façon à éviter une trop grande sensibilité aux bruits de mesure et aux fronts de la consigne, et permet d'être synthétisable.

Néanmoins, afin de simplifier le problème et parce qu'en pratique le filtre anti-repliement après le convertisseur numérique-analogique crée un effet similaire, nous nous contenterons d'un modèle théorique simplifié :

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

Pour commencer, on suppose que le gain du correcteur est de 1 et on traitera son réglage par la suite.

La tension créée par le correcteur rentre dans le pré-actionneur puis le moteur qui fournit θ_S. Il inclut un moteur électrique *brushless* avec son pré-actionneur, et une transmission constituée d'engrenages et d'un système à cabestan permettant de mettre en rotation l'axe que l'on cherche à asservir. On désignera par "moto-réducteur" ce système dans la suite de cette partie. Il reçoit le couple résistant venant de l'extérieur, noté $C_r(p)$. Le schéma bloc correspondant se trouve figure 15, où l'algèbre des schémas blocs a été utilisée pour faire apparaître un retour unitaire.

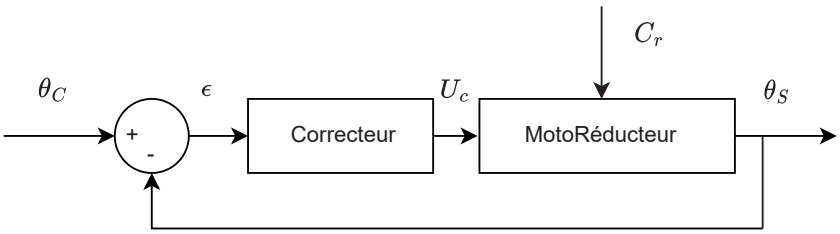


FIGURE 15 – Schéma bloc de l'asservissement

Nous allons établir une simulation numérique de ce système, pour étudier son comportement vis-à-vis de la consigne. En utilisant le théorème de superposition, on supposera donc la perturbation nulle pour la suite du sujet. La première étape est de vérifier la stabilité, en vérifiant que la partie réelle des pôles de la Fonction de Transfert en Boucle Fermée (notée FTBF par la suite) est strictement négative. Pour cela nous allons rentrer la fonction étudiée sous la forme de deux listes correspondant aux coefficients des polynômes constituant son numérateur et son dénominateur. Pour garder une écriture selon la convention de *MATLAB* ou du module *python-control*, dans toute la suite les coefficients seront rangés selon les puissances décroissantes.

Question 41. Écrire une fonction *somme_poly(P, Q)* prenant en arguments deux listes représentant les coefficients de polynômes et renvoyant une liste représentant le polynôme somme.

Question 42. Écrire une fonction *multi_poly(P, Q)* prenant en arguments deux listes représentant les coefficients de polynômes et renvoyant une liste représentant le polynôme produit.

Des fonctions sont fournies dans le DT10. La fonction *multi_FT(num1, den1, num2, den2)* permet de calculer le numérateur et le dénominateur représentant la simplification de deux fonctions de transfert en série. Elle sera utile pour regrouper le correcteur et le motoréducteur en une seule fonction de transfert appelée FTBO. La fonction *FTBF(num, den)* permet de calculer la fonction de transfert du système bouclé à retour unitaire à partir du numérateur et du dénominateur de la FTBO.

Question 43. Un système est stable si les parties réelles des racines du dénominateur de sa FTBF sont négatives.

(a) Écrire une fonction *Est_stable(P)* qui prend en argument une liste représentant un polynôme (ordre des puissances décroissantes), et renvoie *True* si les parties réelles des racines du dénominateur de sa FTBF sont négatives et *False* sinon. On pourra utiliser la fonction *np.roots* décrite à la fin du DT2.

(b) Donner le code permettant d'appeler cette fonction pour valider la stabilité du système corrigé, en prenant un correcteur de gain unitaire.

Tournez la page S.V.P.

3.2.3 Evaluation des performances

La réponse indicielle du système, obtenue avec le code fourni, suit la courbe suivante :

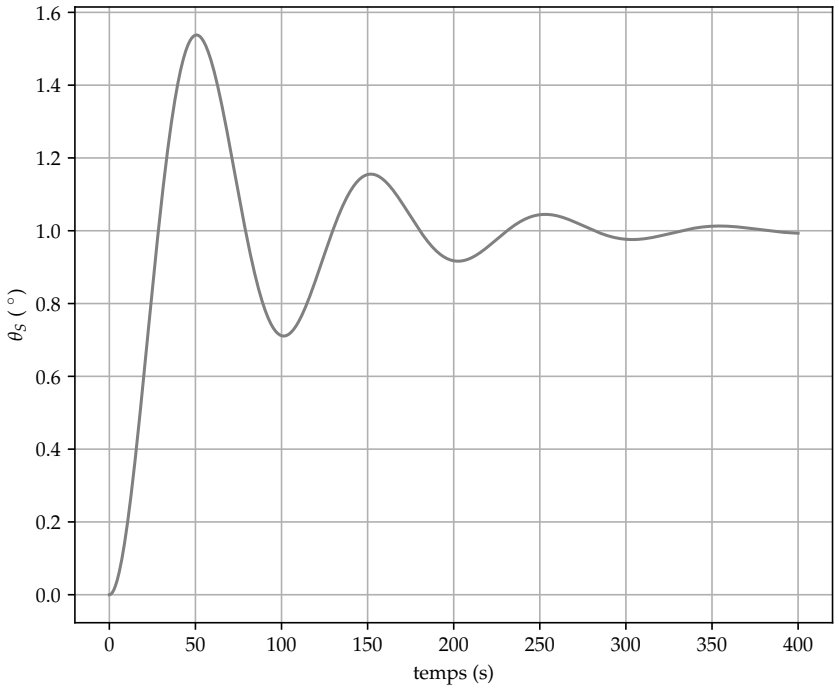


FIGURE 16 – Réponse indicielle du système non corrigé

Pour régler au mieux le système, nous allons étudier sa réponse indicielle. Pour étudier sa rapidité, nous allons calculer son temps de réponse à 5%. On suppose le système stable. On suppose que la simulation a été faite sur un temps suffisamment long pour que la sortie atteigne un régime établi quasi constant.

Question 44. Écrire une fonction *TR5(S, T)* prenant en argument les valeurs de la sortie S, et le vecteur temps correspondant T, et retournant une valeur approchée majorant le temps de réponse à 5%, c'est à dire le temps à partir duquel la sortie est comprise, et reste comprise entre + ou - 5% de sa valeur finale.

Sur le document technique DT10 est fournie la fonction *valeur_ depassement(S, T)* qui renvoie le 1er dépassement relatif, c'est-à-dire

$$D1 = \frac{s_{max} - s_\infty}{s_\infty}$$

La fonction *precision(S)* renvoie, quant à elle, l'erreur statique du système, c'est-à-dire la différence entre la valeur de consigne et s_∞. Pour améliorer la réponse en régime transitoire, on va aussi prendre en compte l'erreur dynamique, en calculant l'intégrale de la valeur absolue de l'erreur $A = \int_0^\infty |e(t) - s(t)|dt$. En pratique nous ferons le calcul sur un intervalle de temps fini.

Question 45. Écrire une fonction *Aire(S, T)*, qui renvoie l'intégrale de la valeur absolue

de l'erreur en utilisant la méthode des trapèzes. S est le tableau des valeurs de la sortie correspondant au vecteur T représentant le temps.

Afin de chercher une solution optimale, il faut combiner les différents critères de performance dans un indicateur global. On utilise pour cela la fonction *ponderation_ cout(S, T)* fournie dans le DT10 qui fait appel aux fonctions précédentes et renvoie un indicateur de performance globale (plus il est petit, meilleur c'est).

3.2.4 Détermination du meilleur correcteur possible

Approche brute-force

Dans cette partie nous cherchons un réglage optimal du correcteur en comparant 4 méthodes de réglage du correcteur PID : force brute, la méthode de Ziegler-Nichols temporelle, une approche par algorithme génétique, une approche par essaim particulaire.

On cherche donc le triplet K_p, T_i, T_d minimisant la fonction *ponderation_ cout*, en prenant $K_p \in [0.01, 100]$ car la tension d'alimentation du moteur ne peut pas être trop grande, et $T_i \in [0.01, 100]$ et $T_d \in [0, 50]$. Une première approche, par force brute, consiste à essayer le maximum de valeurs différentes.

Question 46. Évaluer la complexité de l'appel à la fonction *Indicateur*. Si cet appel prend 0.01 s, combien de temps faudrait-il pour tester tous les cas avec 2^{10} valeurs pour chaque coefficient ?

Méthode de Ziegler-Nichols

La méthode de Ziegler-Nichols temporelle utilise la réponse indicielle du processus seul (sans correcteur). Il faut déterminer le point d'inflexion de la courbe (celui correspondant au temps le plus faible). On mesure ensuite la pente p en ce point, et le retard apparent L correspondant au point d'intersection de la tangente avec l'axe des abscisses. On prend ensuite comme coefficients $K_p = 1,2/(pL)$, $T_i = 2L$ et $T_d = 0,5L$.

Question 47. Écrire le code permettant de déterminer les coefficients du correcteur avec cette méthode.

Tournez la page S.V.P.

Le système non corrigé a un score de 400, qui tombe à 43 avec cette méthode. Nous allons voir par la suite s'il est encore possible d'optimiser ce score.

Optimisation par algorithme génétique

Cet algorithme repose sur un processus d'optimisation itératif évolutionniste, reproduisant les mécanismes de la sélection naturelle. Le principe consiste à initialiser un groupe de candidats possibles dont on va sélectionner les meilleurs éléments, qui seront conservés et croisés pour obtenir de nouveaux candidats. A chaque génération, la population se conserve et les caractéristiques du groupe s'améliorent jusqu'à converger vers un optimum.

- La population est l'ensemble des solutions envisageables. C'est l'ensemble des triplets (K_p, T_i, T_d). On en considère 100.
- L'individu représente une solution. C'est un triplet (K_p, T_i, T_d).
- Le chromosome est une composante de la solution. K_p, T_i, T_d sont 3 chromosomes.
- Le gène est une caractéristique, une particularité. Ce sera les bits dans le codage en binaire de la valeur numérique d'un chromosome.

Il y a trois opérateurs d'évolution dans les algorithmes génétiques :

- La sélection : choix des individus les mieux adaptés. On conservera les 20 meilleurs.
- Le croisement : mélange par la reproduction des particularités des individus choisis. On considère qu'un enfant hérite d'un chromosome d'un parent et de deux de l'autre, aléatoirement. On créera des reproductions entre les 20 meilleurs.
- La mutation : altération aléatoire des particularités d'un individu pour éviter les minima locaux. À chaque reproduction 1 bit sur un 1 chromosome est éventuellement modifié.

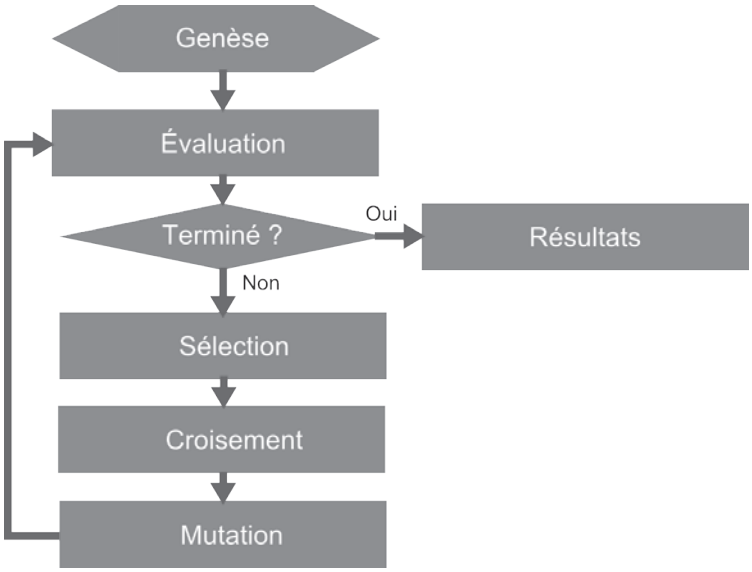


Figure 17 – Principe de l'algorithme génétique

Pour mettre en place la mutation, on a besoin de normaliser la façon dont on va représenter les valeurs numériques de K_p, T_i, T_d. On crée une bijection entre les valeurs et leur codage en binaire sur 10 bits, N_p, N_i, N_d. Pour cela nous utiliserons :

$$K_p = K_{pmin} + (K_{pmax} - K_{pmin})\frac{N_p}{2^{10} - 1}$$

$$T_i = T_{imin} + (T_{imax} - T_{imin})\frac{N_i}{2^{10} - 1}$$

$$T_d = T_{dmin} + (T_{dmax} - T_{dmin})\frac{N_d}{2^{10} - 1}$$

Pour calculer le coût, il faut pour un jeu de valeurs N_p, N_i, N_d calculer les coefficients du PID, puis vérifier que le système est stable, et si c'est le cas calculer le coût, et sinon renvoyer 10000.

Question 48. Écrire une fonction *calcul_ cout(Np, Ni, Nd)* déterminant le coût d'une solution pour un jeu de valeurs de N_p, N_i, N_d.

Question 49. Écrire une fonction *generer_ chromosome(c)* créant une chaîne aléatoire de '0' ou '1' de longueur n.

Question 50. Écrire une fonction *generer_ population_ initiale(p, n)* créant une population de p individus aléatoirement. On rappelle que chaque individu a 3 chromosomes, constitués de n gènes. Ainsi par exemple individu = ["1000101011", "0101010101", "0011001010"]. On utilise une convention gros-boutiste (poids le plus fort à gauche, *big endian*).

Question 51. Écrire une fonction *decodage(b)* prenant une chaîne de caractères représentant un nombre binaire et renvoyant le nombre décimal correspondant.

Question 52. Écrire une fonction *tri(L)* la plus rapide possible, triant les individus en fonction de leur performance (le coût le plus faible en premier). Donner le nom de votre tri ainsi que sa complexité dans le pire et dans le meilleur des cas.

Le résultat de ce tri sera utilisé pour sélectionner les 20 meilleurs individus que l'on conservera à chaque génération.

Question 53. Écrire une fonction *croisement(P1, P2)* qui permet de générer deux nouveaux individus à partir de deux parents P1 et P2. L'enfant hérite aléatoirement d'un chromosome d'un parent et de deux de l'autre.

De plus il subit une mutation d'un de ses gènes : un bit d'un de ses chromosomes change aléatoirement.

Question 54. Écrire une fonction *mutation(E)* changeant aléatoirement un bit d'un des chromosomes d'un individu E.

Tournez la page S.V.P.

Pour établir une nouvelle génération :

- sélection : on garde les 20 meilleurs candidats,
- croisement : on fait se reproduire le 1er avec les 19 suivants, puis le deuxième avec les 18 suivants, puis le 3ème avec les 17 suivants, le 4ème avec les 16 suivants, et on complète avec un tirage aléatoire de 10 individus pour arriver à 100.

Question 55. Écrire une fonction *nouvGeneration(L)* traduisant l'algorithme ci-dessus.

Question 56. Écrire la fonction *doublons(L)* qui prend en argument une population (sous forme de liste) et renvoie cette population débarrassée de ses clones, remplacés par des individus tirés aléatoirement.

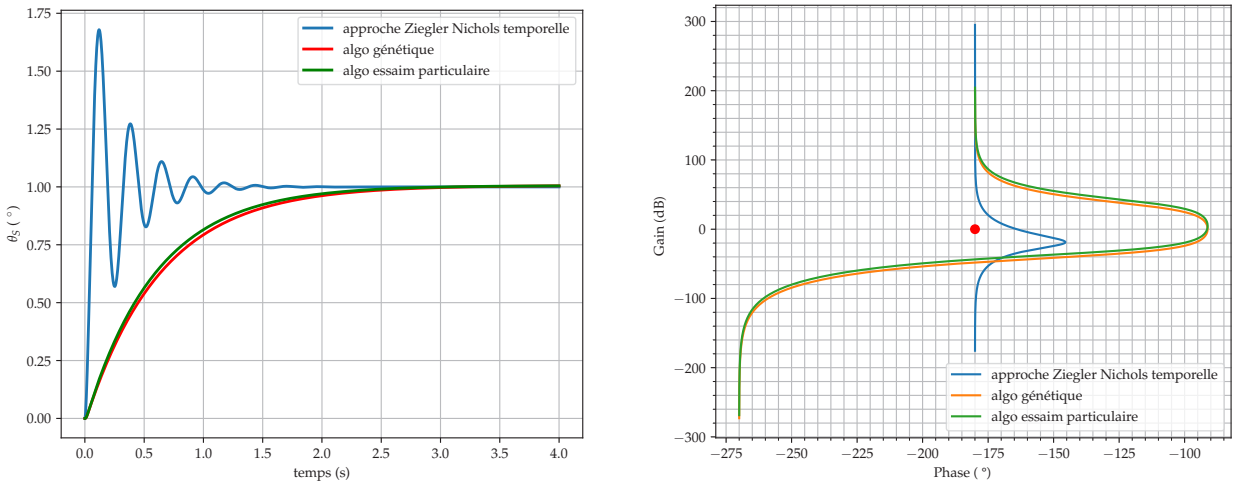
Question 57. Écrire le code permettant de trouver les valeurs optimales du correcteur après 30 générations.

Optimisation par essaim de particules

L'optimisation par essaim de particules (*Particle Swarm Optimization*, abrégée ici PSO) est une méthode d'optimisation stochastique, pour les fonctions non-linéaires, basée sur la reproduction d'un comportement social. L'origine de cette méthode vient des observations faites lors des simulations informatiques de vols groupés d'oiseaux et de bancs de poissons. Ces simulations ont mis en valeur la capacité des individus d'un groupe en mouvement à conserver une distance optimale entre eux et à suivre un mouvement global par rapport aux mouvements locaux de leur voisinage.

D'autre part, ces simulations ont également révélé l'importance du mimétisme dans la compétition qui oppose les individus à la recherche de la nourriture. En effet, les individus sont à la recherche de sources de nourriture qui sont dispersées de façon aléatoire dans un espace de recherche, et dès lors qu'un individu localise une source de nourriture, les autres individus vont chercher à reproduire son comportement. Ce comportement social basé sur l'analyse de l'environnement et du voisinage constitue alors une méthode de recherche d'optimum par l'observation des tendances des individus voisins. Chaque individu cherche à optimiser ses chances en suivant une tendance qu'il modère par son propre vécu.

Question 58. Le document technique DT11 donne un exemple d'utilisation de la fonction *pso* du module *pyswarm*. Écrire le code permettant d'utiliser l'optimisation par essaim particulaire pour trouver les valeurs optimales de K_d, T_i et T_p.



(a) (b)

FIGURE 18 – (a) Réponse indicielle du système pour différents réglages du correcteur (b) Diagramme de Black pour différents réglages du correcteur

La méthode de l'algorithme génétique avec 30 générations prend approximativement autant de temps que celle utilisant des essaims particulaires.

Question 59. Conclure quant à l'adéquation et l'efficacité de ces méthodes dans le cadre du réglage du correcteur adapté à l'exosquelette, et proposer des pistes de solutions pour améliorer les performances de l'asservissement.

Tournez la page S.V.P.

Avertissement

Ce sujet est basé sur des discussions avec des membres de l'équipe de CLINATEC, que les auteurs remercient pour leur disponibilité, ainsi que sur des informations disponibles dans la littérature scientifique et publiées entre autre par CLINATEC. Néanmoins il n'engage pas et ne représente nullement le travail de CLINATEC. Etant donné le caractère confidentiel d'un sujet de concours, il n'a pas été lu par CLINATEC avant parution, et est basé sur des extrapolations libre effectuées par les auteurs à partir de leur compréhension du projet, notamment pour les parties 1 et 3.

Bibliographie

Liste non-exhaustive des ouvrages et articles de recherche ayant permis de concevoir ce sujet :

Références

[1] M Dombre and W Khalil. Modelisation et commande des robots, hermes, 1988.

[2] Andrey Eliseyev and Tatiana Aksenova. Stable and artifact-resistant decoding of 3d hand trajectories from ecog signals using the generalized additive model. *Journal of neural engineering*, 11(6) :066005, 2014.

[3] Andrey Eliseyev and Tetiana Aksenova. Penalized multi-way partial least squares for smooth trajectory decoding from electrocorticographic (ecog) recording. *PloS one*, 11(5) :e0154878, 2016.

[4] Corinne S Mestais, Guillaume Charvet, Fabien Sauter-Starace, Michael Foerster, David Ratel, and Alim Louis Benabid. Wimagine : wireless 64-channel ecog recording implant for long term clinical applications. *IEEE transactions on neural systems and rehabilitation engineering*, 23(1) :10–21, 2014.

[5] Alexandre Moly. *Innovative decoding algorithms for Chronic ECoG-based Brain Computer Interface (BCI) for motor disabled subjects in laboratory and at home*. PhD thesis, Université Grenoble Alpes [2020-....], 2020.

[6] Kentaro Shimoda, Yasuo Nagasaka, Zenas C Chao, and Naotaka Fujii. Decoding continuous three-dimensional hand trajectories from epidural electrocorticographic signals in japanese macaques. *Journal of neural engineering*, 9(3) :036015, 2012.

[7] Andriy Yelisyeyev. *Interface cerveau-machine à partir d'enregistrement électrique cortical*. PhD thesis, Grenoble, 2011.

Documents Techniques

Document technique DT1 : Extraits de la documentation technique du ZL70102

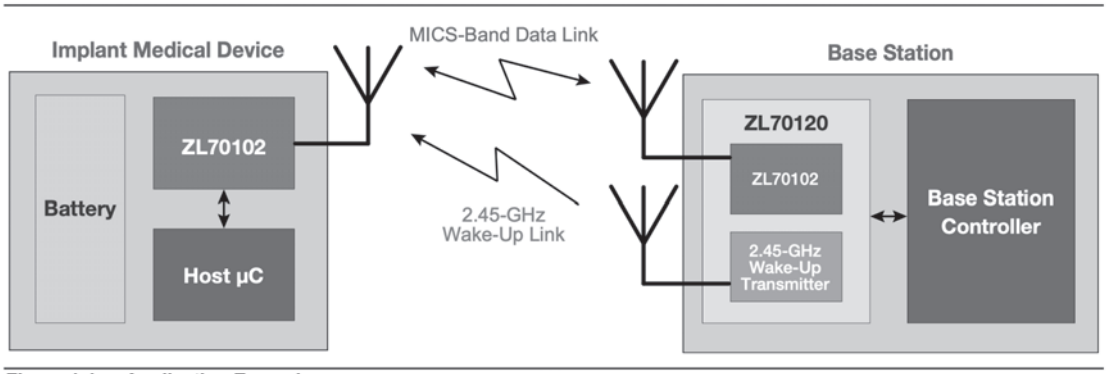


Figure 1-1 • **Application Example**

400-MHz Packet Definition

The packet definition is chosen to enable a high effective data rate. The packet header should be kept as small as possible and the payload should be as large as possible. The same packet definition is used in both the uplink and downlink. The basis for the packet definition and the link protocol is fully described in the ZL70102 Design Manual.

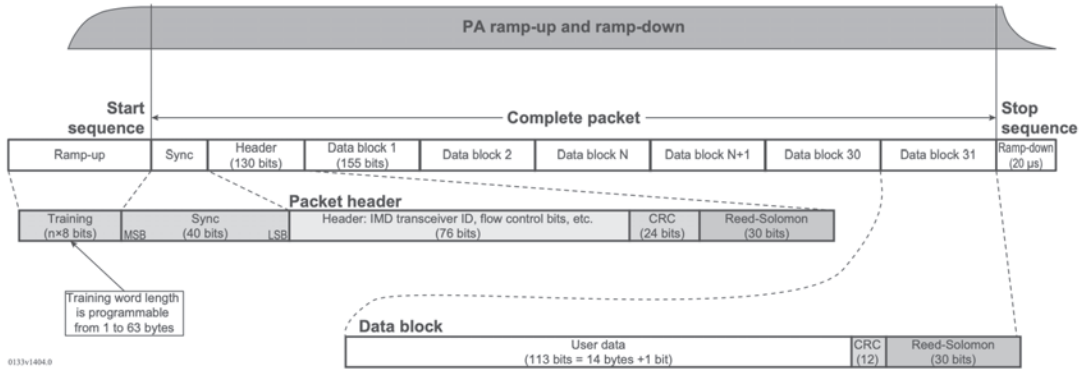


Figure 3-9 • **Packet Definition (first in time on the left side)**

Table 3-3 • **Options for Modulation Modes, Data Rates, and Receiver Sensitivity**

Modulation Mode	Maximum Raw Radio Data Rate (kbit/s)	Maximum Effective Data Rate (kbit/s)	Typical Receiver Sensitivity (Note 1)
2FSK-fallback	200	134	−98dBm
2FSK	400	265	−91dBm
4FSK	800	515 (Note 2)	−79dBm

Notes:

1. *The sensitivity is based on the application circuit in Figure 10-1 on page 10-1, at the reference point of the dual-band antenna (50ohm). This value represents a packet error rate of 10%.*
2. *Requires calibration of the RX ADC. Refer to the ZL70102 Design Manual for the calibration procedure.*

Tournez la page S.V.P.

C

Document technique DT2 : *Numpy Quickstart* et Extraits de la documentation *Numpy*

3.2.5 Trigonometric functions

sin(x, /[, out, where, casting, order, ...])	Trigonometric sine, element-wise.
cos(x, /[, out, where, casting, order, ...])	Cosine element-wise.
tan(x, /[, out, where, casting, order, ...])	Compute tangent element-wise.
arcsin(x, /[, out, where, casting, order, ...])	Inverse sine element-wise.
arccos(x, /[, out, where, casting, order, ...])	Inverse cosine, element-wise.
arctan(x, /[, out, where, casting, order, ...])	Trigonometric inverse tangent, element-wise.
hypot($x1, x2$, /[, out, where, casting, ...])	Given the "legs" of a right triangle, return its hypotenuse.
arctan2($x1, x2$, /[, out, where, casting, ...])	Element-wise arc tangent of x1/x2 choosing the quadrant correctly.
degrees(x, /[, out, where, casting, order, ...])	Convert angles from radians to degrees.
radians(x, /[, out, where, casting, order, ...])	Convert angles from degrees to radians.
unwrap(p[, discont, axis, period])	Unwrap by taking the complement of large deltas with respect to the period.
deg2rad(x, /[, out, where, casting, order, ...])	Convert angles from degrees to radians.
rad2deg(x, /[, out, where, casting, order, ...])	Convert angles from radians to degrees.

Hyperbolic functions

sinh(x, /[, out, where, casting, order, ...])	Hyperbolic sine, element-wise.
cosh(x, /[, out, where, casting, order, ...])	Hyperbolic cosine, element-wise.
tanh(x, /[, out, where, casting, order, ...])	Compute hyperbolic tangent element-wise.
arcsinh(x, /[, out, where, casting, order, ...])	Inverse hyperbolic sine element-wise.
arccosh(x, /[, out, where, casting, order, ...])	Inverse hyperbolic cosine, element-wise.
arctanh(x, /[, out, where, casting, order, ...])	Inverse hyperbolic tangent element-wise.

Rounding

around(a[, decimals, out])	Evenly round to the given number of decimals.
round_(a[, decimals, out])	Round an array to the given number of decimals.
rint(x, /[, out, where, casting, order, ...])	Round elements of the array to the nearest integer.
fix(x[, out])	Round to nearest integer towards zero.
floor (x, /[, out, where, casting, order, ...])	Return the floor of the input, element-wise.
ceil(x, /[, out, where, casting, order, ...])	Return the ceiling of the input, element-wise.
trunc(x, /[, out, where, casting, order, ...])	Return the truncated value of the input, element-wise.

Tournez la page S.V.P.

Sums, products, differences

prod(a[, axis, dtype, out, keepdims, ...])	Return the product of array elements over a given axis.
sum(a[, axis, dtype, out, keepdims,..])	Sum of array elements over a given axis.
nanprod(a[, axis, dtype, out, keepdims])	Return the product of array elements over a given axis treating Not a Numbers (NaNs) as ones.
nansum(a[, axis, dtype, out, keepdims])	Return the sum of array elements over a given axis treating Not a Numbers (NaNs) as zero.
cumprod(a[, axis, dtype, out])	Return the cumulative product of elements along a given axis.
cumsum(a[, axis, dtype, out])	Return the cumulative sum of the elements along a given axis.
nancumprod(a[, axis, dtype, out])	Return the cumulative product of array elements over a given axis treating Not a Numbers (NaNs) as one.
nancumsum(a[, axis, dtype, out])	Return the cumulative sum of array elements over a given axis treating Not a Numbers (NaNs) as zero.
diff(a[, n, axis, prepend, append])	Calculate the n-th discrete difference along the given axis.
ediff1d(ary[, to_end, to_begin])	The differences between consecutive elements of an array.
gradient(f, *varargs[, axis, edge_order])	Return the gradient of an N-dimensional array.
cross(a, b[, axisa, axisb, axisc, axis])	Return the cross product of two (arrays of) vectors.
trapz(y[, x, dx, axis])	Integrate along the given axis using the composite trapezoidal rule.

Exponents and logarithms

exp(x, /[, out, where, casting, order, ...])	Calculate the exponential of all elements in the input array.
expm 1(x, /[, out, where, casting, order, ..])	Calculate exp (x) − 1 for all elements in the array.
exp 2(x, /[, out, where, casting, order, ...])	Calculate $2 * *p$ for all p in the input array.
log(x, /[, out, where, casting, order, ...])	Natural logarithm, element-wise.
log 10(x, /[, out, where, casting, order, ...])	Return the base 10 logarithm of the input array, elementwise.
log 2(x, /[, out, where, casting, order, ...])	Base-2 logarithm of x.
log 1 p(x, /[, out, where, casting, order,..])	Return the natural logarithm of one plus the input array, element-wise.
logaddexp(x1, x2, /[, out, where, casting,,...])	Logarithm of the sum of exponentiations of the inputs.
logaddexp2(x1, x2, /[, out, where, casting,,..])	Logarithm of the sum of exponentiations of the inputs in base- 2.

Polynomials

numpy.roots()

return the roots of a polynomial with coefficients given in p. The values in the rank-1 array p are coefficients of a polynomial. If the length of p is n+1 then the polynomial is described by : $p[0] * x * *n + p[1] * x * *(n-1) + \ldots + p[n-1] * x + p[n]$

Syntax : numpy.roots(p)

Parameters : p : [array_like] Rank-1 array of polynomial coefficients.

Return : [ndarray] An array containing the roots of the polynomial.

Document technique DT3 : Extrait de la documentation *numpy.bincount*

numpy.bincount (x, /, weights=None, minlength=0)

Count number of occurrences of each in array of non-negative ints.

The number of bins (of size 1) is one larger than the largest value in x. If *minlength* is specified, there will be at least this number of bins in the output array (though it will be longer if necessary, depending on the contents of x). Each bin gives the number of occurrences of its index value in x. If *weights* is specified the input array is weighted by it, i.e. if a value n is found at position i, out[n] += weight[i] instead of out[n] += 1.

Parameters :	x : array_like, 1 dimension, nonnegative ints Input array.
	weights : array_like, optional Weights, array of the same shape as x.
	minlength : int, optional A minimum number of bins for the output array.
Returns :	out : ndarray of ints The result of binning the input array. The length of out is equal to np.amax(x) + 1.
Raises :	ValueError If the input is not 1-dimensional, or contains elements with negative values, or if *minlength* is negative.
	TypeError If the type of the input is float or complex.

Examples

```
>>> np.bincount(np.arange(5))
array([1, 1, 1, 1, 1])
>>> np.bincount(np.array([0, 1, 1, 3, 2, 1, 7]))
array([1, 3, 1, 1, 0, 0, 0, 1])

>>> x = np.array([0, 1, 1, 3, 2, 1, 7, 23])
>>> np.bincount(x).size == np.amax(x)+1
True
```

Document technique DT4 : Extrait de la documentation *numpy.convolve*

numpy.convolve(a, v, mode='full')

Returns the discrete, linear convolution of two one-dimensional sequences.

The convolution operator is often seen in signal processing, where it models the effect of a linear timeinvariant system on a signal [1]. In probability theory, the sum of two independent random variables is distributed according to the convolution of their individual distributions.

If v is longer than a, the arrays are swapped before computation.

Tournez la page S.V.P.

Parameters :	a : (N,) array_like First one-dimensional input array.
	v : (M,)array_like Second one-dimensional input array.
	mode : {'full', 'valid', 'same'}, optional 'full' : By default, mode is 'full'. This returns the convolution at each point of overlap, with an output shape of (N + M − 1,). At the end-points of the convolution, the signals do not overlap completely, and boundary effects may be seen. 'same' : Mode 'same' returns output of length max(M, N). Boundary effects are still visible. 'valid' : Mode 'valid' returns output of length max(M, N) − min(M, N) + 1. The convolution product is only given for points where the signals overlap completely. Values outside the signal boundary have no effect.
Returns :	out : ndarray Discrete, linear convolution of a and v.

Notes

The discrete convolution operation is defined as

$$(a * v)_n = \sum_{m=-\infty}^{\infty} a_m v_{n-m}$$

Note how the convolution operator flips the second array before "sliding" the two across one another :

```
>>>np.convolve([1, 2, 3], [0, 1, 0.5])
array([0., 1., 2.5, 4., 1.5])
```

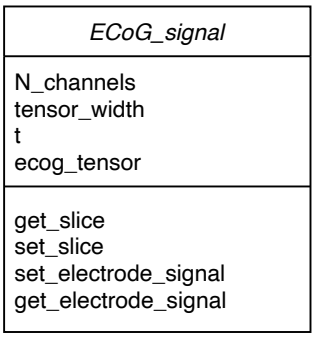
Only return the middle values of the convolution. Contains boundary effects, where zeros are taken into account :

```
np.convolve([1, 2, 3], [0, 1, 0.5], 'same')
array([1., 2.5, 4.])
```

The two arrays are of the same length, so there is only one position where they completely overlap :

```
np.convolve([1, 2, 3], [0, 1, 0.5], 'valid')
array([2.5])
```

Document technique DT5 : Classe ECoG de stockage d'un signal sous forme de tenseur




```python
import numpy as np
import scipy.io
from scipy import signal
import copy

################################
## miscalleneous functions ##
################################
def open_ECoG_recording(foldername, N_channels=64, T_sample=1e-3, unit=1e-6):
    ''' opens files from Shimoda 2012 ECoG Dataset '''
    ECoG_channels = []
    for k in range(1, N_channels+1):
        current_ECoG_sig = scipy.io.loadmat(foldername+'/ECoG_ch'+str(k)+'.mat')
        ECoG_channels.append(current_ECoG_sig['ECoGData_ch'+str(k)][0])#*unit)
    N_samples = len(ECoG_channels[0])
    t = np.linspace(0, (N_samples-1)*T_sample, num=N_samples)
    return t, ECoG_channels

#############################################
## class for ECoG recording as a tensor ##
#############################################
class ECoG_signal:
    '''
    Basic ECoG class containing tensor definition and operations on tensors
    '''
    def __init__(self, N_channels=64):
        '''
        Init the ECoG signal

        Parameters:
        ----------
        N_channels : int
            number of electrode on the recording matrix,
            should be a power of 2 (square matrix)
            by default set to 64
        '''
        if (N_channels & (N_channels-1) == 0) and N_channels != 0:
            self.N_channels = int(N_channels)
            self.tensor_width = int(np.sqrt(N_channels))
            self.T_sample = None
        else:
```


Tournez la page S.V.P.

```
42          raise ValueError('Work with a power of 2 number of channels only')
43
44      def create_tensor(self, length_t_vector, T_sample=1e-3):
45          '''
46          create the 3D tensor (N_channels*Nchannels*length_t_vector)
47
48          Parameters:
49          ----------
50          length_t_vector: int
51              length of the time vector
52          T_sample: float
53              blablabla
54          '''
55          self.ecog_tensor = np.zeros((self.tensor_width, self.tensor_width,
            length_t_vector))
56          self.t = np.linspace(0,(length_t_vector-1)*T_sample, num= length_t_vector
            )
57          self.T_sample = T_sample
58
59
60      def set_slice(self, m, t):
61          '''
62          Set a slice of ECoG (all channels value) as a matrix for a given time
            index
63
64          Parameters:
65          ----------
66          m: array or np.array
67              ECoG value at a given time index
68          t: int
69              time index
70          '''
71          if m.shape[0] == m.shape[1] and m.shape[0] == self.tensor_width:
72              self.ecog_tensor[:,:,t] = m
73          else:
74              raise IndexError('specified slice cannot fit the tensor')
75
76      def get_slice(self, t):
77          '''
78          Get a slice of ECoG (all channels value) as a matrix for a given time
            index
79
80          Parameters:
81          ----------
82          t: int
83              time index
84
85          Returns:
86          -------
87          np.array: all ECoG channels value at the given time index
88          '''
89          return self.ecog_tensor[:,:,t]
90
91      def set_electrode_signal(self, N_elec, signal):
92          '''
93          Set the signal of one electrode along the time dimension
```

```
94
95      Parameters:
96      ----------
97      N_elec: int
98        number of the channel to set
99      signal: array or np.array
100       signal of the electrode
101     '''
102     if N_elec<self.N_channels: # test if the electrode number is in the
        electrde matrix
103       # get electrode position in the tensor
104       row = N_elec // self.tensor_width
105       column = N_elec % self.tensor_width
106       if  len(signal) == self.ecog_tensor.shape[2]:
107         self.ecog_tensor[row, column, :] = signal
108       else:
109         raise IndexError('Given signal is not of the time length of the
        tensor')
110     else:
111       raise ValueError('Specified electrode is out of electrode set')
112
113   def get_electrode_signal(self, N_elec):
114     '''
115     Get the signal of one electrode along the time dimension
116
117     Parameters:
118     ----------
119     N_elec: int
120       number of the chanel to get
121
122     Returns:
123     -------
124     np.array: signal of the specified channel number
125     '''
126     if N_elec<self.N_channels: # test if the electrode number is in the
        electrde matrix
127       # get electrode position in the tensor
128       row = N_elec // self.tensor_width
129       column = N_elec % self.tensor_width
130       return self.ecog_tensor[row, column, :]
131     else:
132       raise ValueError('Specified electrode is out of electrode set')
```


Tournez la page S.V.P.

Document technique DT6 : classe PLS de description des *Partial Least Squares* utilisant l'algorithm NIPALS

Le code suivant comporte des trous faisant référence au Document Réponse DR5 :

```
class PLS(object):
    """A class for PLS calculated by the NIPALS algorithm.
    Initialize with a Pandas DataFrame or an object that can be turned into a
    DataFrame
    (e.g. an array or a dict of lists)"""

    def __init__(self, x_df, y_df):
        super(PLS, self).__init__()
        if type(x_df) != pd.core.frame.DataFrame:
            x_df = pd.DataFrame(x_df)
        if type(y_df) != pd.core.frame.DataFrame:
            y_df = pd.DataFrame(y_df)
        # Make sure data is numeric
        self.x_df = x_df.astype("float")
        self.y_df = y_df.astype("float")
        # Check for and remove infs
        if np.isinf(self.x_df).any().any():
            logging.warning(
                "X data contained infinite values, converting to missing
    values"
            )
            self.x_df.replace([np.inf, -np.inf], np.nan, inplace=True)
        if np.isinf(self.y_df).any().any():
            logging.warning(
                "Y data contained infinite values, converting to missing
    values"
            )
            self.y_df.replace([np.inf, -np.inf], np.nan, inplace=True)

    def fit(
        self,
        ncomp,
        startcol
        center=True,
        scale=True,
        tol=0.000001,
        maxiter=500,
        dropzerovar=False,
    ):
        """The Fit method, will fit a PLS to the X and Y data"""

        # Convert to np array
        self.x_mat = self.x_df.values
        self.y_mat = self.y_df.values
        self.center = center
        self.scale = scale
        self.x_mean = np.nanmean(self.x_mat, axis=0)
        self.y_mean = np.nanmean(self.y_mat, axis=0)
        self.x_std = np.nanstd(self.x_mat, axis=0, ddof=1)
        self.y_std = np.nanstd(self.y_mat, axis=0, ddof=1)

        if center:
```

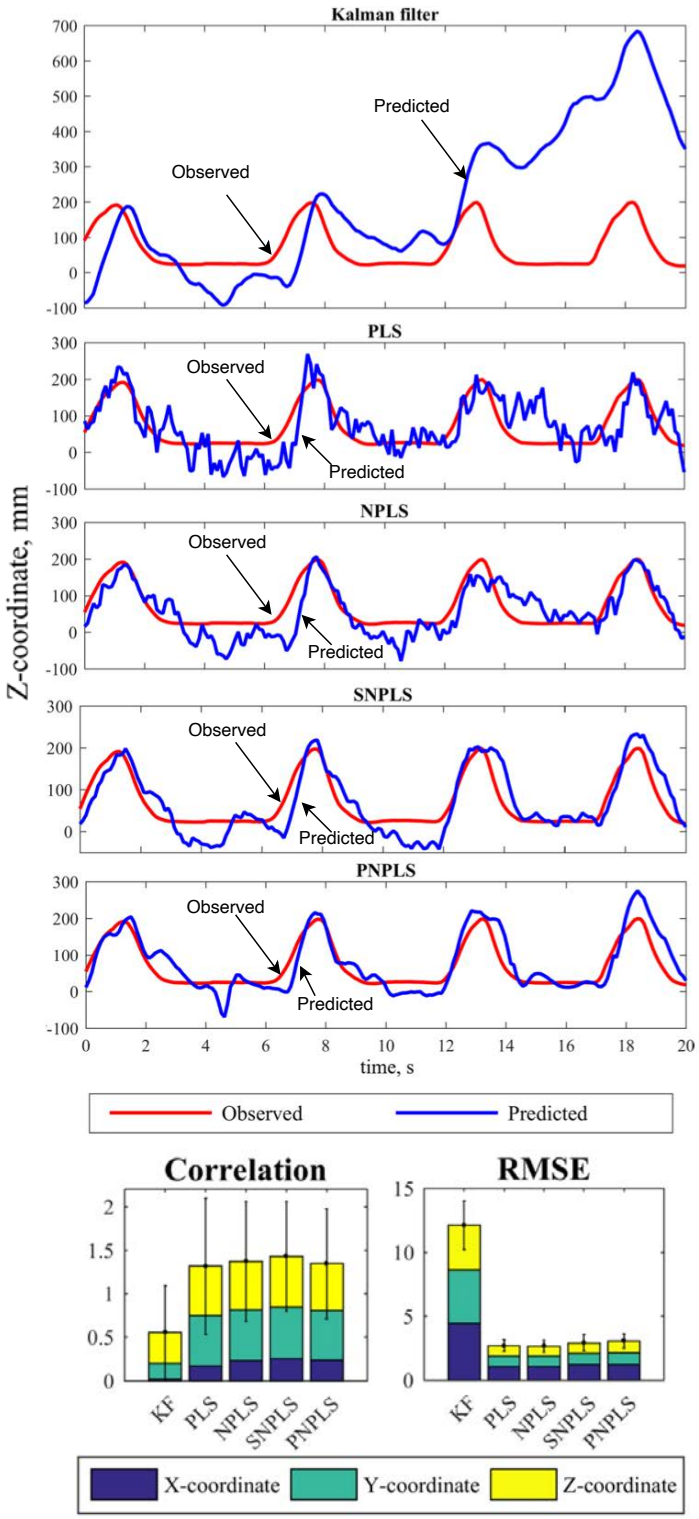
```
            self.x_mat = self.x_mat - self.x_mean
            self.y_mat = self.y_mat - self.y_mean
        if scale:
            self.x_mat = self.x_mat / self.x_std
            self.y_mat = self.y_mat / self.y_std

        # initialize outputs
        loadings = np.empty((x_nc, ncomp))
        scores = np.empty((nr, ncomp))
        u = np.empty((nr, ncomp))
        weights = np.empty((x_nc, ncomp))
        q = np.empty((y_nc, ncomp))
        b = np.empty((ncomp,))

        for comp in range(ncomp):
            train_x_mat = self.x_mat
            train_y_mat = self.y_mat
            # Set u to some column of Y
            uh = train_y_mat[:, startcol]
            th = uh

            it = 0
            while True:
                ## Voir Document Reponse DR5 ##

                # Check convergence
                if np.nansum((th - th_old) ** 2) < tol:
                    break
                it += 1
                if it >= maxiter:
                    raise RuntimeError(
                        "Convergence was not reached in {} iterations for component {}".format(
                            maxiter, comp
                        )
                    )

            # Calculate X loadings and rescale the scores and weights
            ph = train_x_mat.T.dot(th) / sum(th * th)

            loadings[:, comp] = ph
            scores[:, comp] = th
            u[:, comp] = uh
            q[:, comp] = qh
            weights[:, comp] = wh
            bh = ## voir Document Reponse DR5 ##
            b[comp] = bh

            self.x_mat = self.x_mat - np.outer(th, ph)
            self.y_mat = self.y_mat - bh * np.outer(th, qh)

        # Convert results to DataFrames
        self.scores = pd.DataFrame(
            scores, index=self.x_df.index, columns=["PC{}".format(i + 1) for i in range(ncomp)],
        )
```


Tournez la page S.V.P.

```
104         self.loadings = pd.DataFrame(
105             loadings, index=self.x_df.columns, columns=["PC{}".format(i + 1)
    for i in range(ncomp)],
106         )
107         self.u = pd.DataFrame(
108             u, index=self.x_df.index, columns=["PC{}".format(i + 1) for i in
    range(ncomp)],
109         )
110         self.q = pd.DataFrame(
111             q, index=self.y_df.columns, columns=["PC{}".format(i + 1) for i
    in range(ncomp)],
112         )
113         self.weights = pd.DataFrame(
114             weights, index=self.x_df.columns, columns=["PC{}".format(i + 1)
    for i in range(ncomp)],
115         )
116         self.b = pd.Series(b, index=["PC{}".format(i + 1) for i in range(
    ncomp)])
117         return True
```

Document technique DT7 : Résultats de test de différentes méthodes et modèles de décodage des intentions de mouvement

Les figures suivantes sont extraites de [3].



La racine de l'erreur quadratique moyenne est définie par $RMSE = ||\mathbf{y} - \hat{\mathbf{y}}||_2 / ||\mathbf{y} - \bar{\mathbf{y}}||_2$.

Tournez la page S.V.P.

Document technique DT8 : Code des classes concernant les quaternions et rotations

```
import math
import numpy as np

#### OBJECT
class Quaternion:
  '''Quaternion
  object to represent and perform operations on quanternions

  '''
  def __init__(self, a, b, c, d):
    self.a = float(a)
    self.b = float(b)
    self.c = float(c)
    self.d = float(d)

  ## Operator overloading
  def __str__(self):
    sign_i = '+' if self.b >=0 else '-'
    sign_j = '+' if self.c >=0 else '-'
    sign_k = '+' if self.d >=0 else '-'
    return str(self.a)+sign_i+str(abs(self.b))+".i"+sign_j+str(abs(self.c))+".j"+sign_k+str(abs(self.d))+".k"
  def __eq__(self, other):
    if isinstance(other, Quaternion):
      return self.a==other.a and self.b==other.b and self.c==other.c and self.d==other.d
    else:
      return False

  def __ne__(self, other):
    return not self==other

  def __add__(self, other):
    if isinstance(other, Quaternion):
      return Quaternion(self.a+other.a, self.b+other.b, self.c+other.c, self.d+other.d)
    elif isinstance(other, int) or isinstance(other, float):
      return Quaternion(self.a+other, self.b, self.c, self.d)
    elif isinstance(other, complex):
      return Quaternion(self.a+other.real, self.b+other.imag, self.c, self.d)
    else:
      raise TypeError("Operation not allowed, addition can be with Quaternion, int, float or complex, not"+str(type(other)))

  __radd__ = __add__

  def __sub__(self, other):
    if isinstance(other, Quaternion):
      return Quaternion(self.a-other.a, self.b-other.b, self.c-other.c, self.d-other.d)
    elif isinstance(other, int) or isinstance(other, float):
      return Quaternion(self.a-other, self.b, self.c, self.d)
    elif isinstance(other, complex):
```

```
49        return Quaternion(self.a-other.real, self.b-other.imag, self.c, self.d)
50      else:
51        raise TypeError("Operation not allowed, substraction can be with
      Quaternion, int, float or complex, not"+str(type(other)))
52
53    def __truediv__(self, other):
54      if isinstance(other, Quaternion):
55        # 1/other computation
56        inv_other = (1/(other.a**2 + other.b**2 + other.c**2 + other.d**2))*
      Quaternion(other.a, -other.b, -other.c, -other.d)
57        return self.__mul__(inv_other)
58      elif isinstance(other, int) or isinstance(other, float):
59        return Quaternion(self.a/other, self.b/other, self.c/other, self.d/
      other)
60      elif isinstance(other, complex):
61        inv_other = to_Quaternion(1/other)
62        return self.__mul__(inv_other)
63      else:
64        raise TypeError("Operation not allowed, division can be with Quaternion
      , int, float or complex, not"+str(type(other)))
65
66    def __rtruediv__(self, other):
67      if isinstance(other, int) or isinstance(other, float):
68        inv = (other/(self.a**2 + self.b**2 + self.c**2 + self.d**2))*
      Quaternion(self.a, -self.b, -self.c, -self.d)
69        return inv
70      elif isinstance(other, complex):
71        q = to_Quaternion(other)
72        return q.__truediv__(self)
73      else:
74        raise TypeError("Operation not allowed, division can be with Quaternion
      , int, float or complex, not"+str(type(other)))
75
76    def __pow__(self, power, modulo=None):
77      if isinstance(power, int):
78        powered_quat = Quaternion(self.a, self.b, self.c, self.d)
79        for k in range(1, abs(power)):
80          powered_quat = powered_quat*self
81        if power < 0:
82          powered_quat = 1/powered_quat
83        return powered_quat
84      else:
85        raise TypeError("Operation not allowed, power of Quaternion only
      computed for int, not"+str(type(other)))
86
87    ## usefull methods
88    def conjugate(self):
89      return Quaternion(self.a, -self.b, -self.c, -self.d)
90
91    def scalar(self, type=None):
92      if type is 'float':
93        return self.a
94      else:
95        return Quaternion(self.a, 0, 0, 0)
96
97    def is_scalar(self):
```


Tournez la page S.V.P.

```
    if self.b==0 and self.c==0 and self.d==0:
      return True
    else:
      return False

  def vector(self):
    return Quaternion(0, self.b, self.c, self.d)

  def is_vector(self):
    if self.a == 0:
      return True
    else:
      return False

  def norm(self):
    return math.sqrt(self.a**2 + self.b**2 + self.c**2 + self.d**2)

  def is_unit(self, tol=1e-7):
    return np.abs(self.norm() - 1.) < tol

class UnitQuaternion(Quaternion):
  def __init__(self, a, b, c, d):
    assert is_unit(Quaternion(a,b,c,d))
    super().__init__(a,b,c,d)

  def invert(self):
    return self.conjugate()

class Rotation3D:
  def __init__(self, theta, X, Y, Z, deg=True):
    pass
    ## Question 36

  def __call__(self, a, b, c):
    pass
    ## Question 36

### FUNCTIONS
def is_unit(q):
  if not isinstance(q,Quaternion):
    q = to_Quaternion(q)
  return q.is_unit()
```

Document technique DT9 : Code du tracé de la réponse fréquentielle de la FTBO

```
import numpy as np
import matplotlib.pyplot as plt

# definition de la FTBO
w = np.logspace(0, 4, 1200)
p = 1j*w
FTBO = (5/(1+p))*(2/(1+0.1*p))**2*(4/(1+0.01*p))**-2*(3/(1+0.001*p))**2

```

```
# calcul du gain 'G' et de la phase 'Phi'
G = 20 * np.log10(abs(FTBO))
Phi = np.angle(FTBO) * 180 / np.pi

# correction de la phase pour enlever les discontinuites
Phi2 = np.copy(Phi) # Phi2 contiendra la phase corrigee
period = 360
dangle = np.diff(Phi)
dangle_desired = (dangle + period/2.) % period - period/2.
correction = np.cumsum(dangle_desired - dangle)
Phi2[1:] _____________________________

plt.figure()
plt.subplot(311)
plt.semilogx (w, G, color="blue", linewidth="1")
plt.xlabel ("Pulsation")
plt.ylabel ("Gain")
plt.minorticks_on()
plt.grid(which='both')

plt.subplot(312)
plt.semilogx (w, Phi, color="red", linewidth="1")
plt.xlabel("Pulsation")
plt.ylabel("Phase brute")
plt.minorticks_on()
plt.grid(which='both')

plt.subplot(313)
plt.semilogx (w, Phi2, color="red", linewidth="1")
plt.xlabel("Pulsation")
plt.ylabel("Phase corrigee")
plt.minorticks_on()
plt.grid(which='both')
```

Document technique DT10 : Code mesurant les performances et réglant le correcteur de l'asservissement

```
import numpy as np
import matplotlib.pyplot as plt
import control as ct
from scipy.signal import tf2ss
from random import random, randint
from pyswarm import pso

Ki, Kc, Kr, L, R, f = 0.6, 0.1, 1/20, 4E-3, 1, 4E-3, 1.5E-3
numG = [Ki*Kc*Kr]
denG = [L*J, (R*J+L*f), Ki**2+R*f, 0]
# FTBO retour unitaire non corrigee
G = ct.tf(numG, denG)

# Calcul de la FTBF corrigee
def correcteur(Kp, Ti, Td):
    """definition classique du PID, renvoie les polynomes"""
    numC = [Kp*Td*Ti, Kp*Ti, Kp]
```


Tournez la page S.V.P.

```
18      denC = [Ti, 0]
19      return numC, denC
20
21  def multi_FT(num1, den1, num2, den2):
22      """multiplication de fonctions de transferts, polynomes avec puiss
        decroissantes"""
23      num = multi_poly(num1, num2)
24      den = multi_poly(den1, den2)
25      return num, den
26
27  def FTBF(num, den):
28      """calcul de la ftbf a retour unitaire pour une ftbo donnee avec 2 polys
        decroissants"""
29      num_bf = num
30      den_bf = somme_poly(num, den)
31      return num_bf, den_bf
32
33  T = np.linspace(0, 4, 2000)
34
35  A = multi_FT([1], [1], numG, denG)
36  B = FTBF(*A)
37  ftbfCorCalc = ct.tf(*B)
38  t, sCorCalc = ct.step_response(ftbfCorCalc, T)
39  plt.figure()
40  plt.grid()
41  plt.plot(t, sCorCalc, label='Reponse avec FTBF manuelle (correcteur unitaire)
        ')
42  plt.legend()
43  plt.show()
44
45  # PERFORMANCES
46
47  def depassement(s, t):
48      s_fin = s[-1]
49      return((max(s)-s_fin)/s_fin)
50
51  def precision(s, t):
52      return abs(s[-1] - 1)
53
54  def ponderation_cout(T5, D1, P1, A1):
55      duree = T[-1]-T[0]
56      #print(duree)
57      a, b, c, d  = duree, duree, 20, 1   # **2 pour b
58      cout = a*T5 + b*D1*100 + c*P1 + d*A1
59      return cout
60
61  def rep_indicielle(Kp, Ti, Td):
62      numC, denC = correcteur(Kp, Ti, Td)
63      num_BO, den_BO = multi_FT(numG, denG, numC, denC)
64      num_BF, den_BF = FTBF(num_BO, den_BO)
65      BF = ct.tf(num_BF, den_BF)
66      temps, sortie = ct.step_response(BF, T)
67      return(temps, sortie)
68
69  #    Mise en place des fonctions pour optimisation
70  def calcul_coef_correcteur(Np, Ni, Nd):
```

```
71      Kpmin, Kpmax = 0.01, 200
72      Timin, Timax = .01, 100
73      Tdmin, Tdmax = 0, 50
74      Kp = (Kpmax-Kpmin)*Np/(2**10-1)+Kpmin
75      Ti = (Timax-Timin)*Ni/(2**10-1)+Timin
76      Td = (Tdmax-Tdmin)*Nd/(2**10-1)+Tdmin
77      return(Kp, Ti, Td)
78
79  #   Optimisation par algorithme genetique
80  def perf(Li):
81      Ip, Ii, Id = decodage(Li[0]), decodage(Li[1]), decodage(Li[2])
82      return calcul_cout(Ip, Ii, Id)
83
84  cycles = 50
85  population = generer_population_initiale(100, 10)
86  for i in range(cycles):
87      population_triee = tri(population)
88      population = nouv_generation(population_triee)
89      population = doublons(population)
90  solution = population[0]
91  KpGene, TiGene, TdGene = calcul_coef_correcteur(
92      decodage(solution[0]), decodage(solution[1]), decodage(solution[2]))
93
94  print("reglage optimal algo gene :", KpGene, TiGene, TdGene)
95  print("perf algo gene :", perf(solution))
```

Document technique DT11 : Exemple d'utilisation de *pyswarm*

```
1  from pyswarm import pso
2  def cout(x):
3      x1 = x[0]
4      x2 = x[1]
5      return x1**4 - 2*x2*x1**2 + x2**2 + x1**2 - 2*x1 + 5
6  lb = [-3, -1]
7  ub = [2, 6]
8  xopt, fopt = pso(cout, lb, ub)
```

le vecteur *xopt* est la solution optimale de la fonction *cout*. *lb* et *ub* représentent la borne inférieure et la borne supérieure que peuvent prendre les valeurs de *xopt*.

Modèle CMEN-DOC v2 ©NEOPTEC

Nom de famille :
(Suivi, s'il y a lieu, du nom d'usage)

Prénom(s) :

Numéro Inscription :

(Le numéro est celui qui figure sur la convocation ou la feuille d'émargement)

Né(e) le : ☐☐ / ☐☐ / ☐☐☐☐

(Remplir cette partie à l'aide de la notice)

Concours / Examen : …….. **Section/Spécialité/Série :** ……

Epreuve : …….. **Matière :** …….. **Session :** ……..

CONSIGNES

- *Remplir soigneusement, sur CHAQUE feuille officielle, la zone d'identification en MAJUSCULES.*
- *Ne pas signer la composition et ne pas y apporter de signe distinctif pouvant indiquer sa provenance.*
- *Numéroter chaque PAGE (cadre en bas à droite de la page) et placer les feuilles dans le bon sens et dans l'ordre.*
- *Rédiger avec un stylo à encre foncée (bleue ou noire) et ne pas utiliser de stylo plume à encre claire.*
- *N'effectuer aucun collage ou découpage de sujets ou de feuille officielle. Ne joindre aucun brouillon.*

DR1 - DR2 - DR3

Tous les documents réponses sont à rendre, même non complétés.

NE RIEN ECRIRE DANS CE CADRE

Document réponse DR1 : Question 6 (a)

Valeurs de coefficients de la matrice C pour une profondeur de 8 :

0.49	0.416	0.278	0.098				
0.462	0.191	-0.191	-0.462				
0.416	-0.098	-0.49	-0.278	0.278	0.49	0.098	-0.416
0.278	-0.49	0.098	0.416	-0.416	-0.098	0.49	-0.278
0.098	-0.278	0.416	-0.49	0.49	-0.416	0.278	-0.098

Document réponse DR2 : Question 13

```python
import numpy as np

def wavelet2(M, s, w=5):
    x = np.arange(0, M) - (M - 1.0) / 2
    x = x / s
    wave_dat = morlet(x, w=w)
    return (1/np.sqrt(s))*wave_dat

def cwt_convolve(sig, freqs, Ts):
    '''
    Calcul de la CWT en utilisant la fonction numpy.convolve

    Arguments
    ---------
    sig : array_like
        signal a tranformer par ondelette
    freqs : array_like
        frequences centrales des ondelettes filles
    Ts : float
        periode d echantillonage du signal 'sig'

    Returns
    -------
    output : array_like
        transformee en ondelette de sig, pour un vecteur tau de
        meme longueur que sig, aux frequences donnees par freqs
    '''
    w0 = 5
    tlen = len(sig)
    t = np.linspace(0, (tlen-1)*Ts, num=tlen)
    output = np.zeros((_____), dtype=np.complex128)
    for i, freq in enumerate(freqs):
        width = _____
        tau = np.arange(0, tlen) - (tlen - 1.0) / (2*width)
        wavelet_data = _____
        # convolve
        output[i, :] = np.convolve(sig, wavelet_data2, mode=_____)
    return output
```

Document réponse DR3 : Question 21

```
import numpy as np

def wavelet2(M, s, w=5):
    x = np.arange(0, M) - (M - 1.0) / 2
    x = x / s
    wave_dat = morlet(x, w=w)
    return (1/np.sqrt(s))*wave_dat

def cwt_convolve(sig, freqs, Ts):
    '''
    Calcul de la CWT en utilisant la fonction numpy.convolve

    Arguments
    ---------
    sig : array_like
        signal a tranformer par ondelette
    freqs : array_like
        frequences centrales des ondelettes filles
    Ts : float
        periode d echantillonage du signal 'sig'

    Returns
    -------
    output : array_like
        transformee en ondelette de sig, pour un vecteur tau de
        meme longueur que sig, aux frequences donnees par freqs
    '''
    w0 = 5
    tlen = len(sig)
    t = np.linspace(0, (tlen-1)*Ts, num=tlen)
    output = np.zeros((_____________________), dtype=np.complex128)
    for i, freq in enumerate(freqs):
        width = _____________________________________
        tau = ______________________________________
        wavelet_data = _____________________________
        # convolve
        output[i, :] = _____________________________
    return output
```

Modèle CMEN-DOC v2 ©NEOPTEC

Nom de famille :
(Suivi, s'il y a lieu, du nom d'usage)

Prénom(s) :

Numéro Inscription :

(Le numéro est celui qui figure sur la convocation ou la feuille d'émargement)

Né(e) le : ☐☐/☐☐/☐☐☐☐

(Remplir cette partie à l'aide de la notice)

Concours / Examen : …….. **Section/Spécialité/Série :** ……

Epreuve : …….. **Matière :** …….. **Session :** ……..

CONSIGNES

- *Remplir soigneusement, sur CHAQUE feuille officielle, la zone d'identification en MAJUSCULES.*
- *Ne pas signer la composition et ne pas y apporter de signe distinctif pouvant indiquer sa provenance.*
- *Numéroter chaque PAGE (cadre en bas à droite de la page) et placer les feuilles dans le bon sens et dans l'ordre.*
- *Rédiger avec un stylo à encre foncée (bleue ou noire) et ne pas utiliser de stylo plume à encre claire.*
- *N'effectuer aucun collage ou découpage de sujets ou de feuille officielle. Ne joindre aucun brouillon.*

DR4 - DR5

Tous les documents réponses sont à rendre, même non complétés.

NE RIEN ECRIRE DANS CE CADRE

Document réponse DR4 : Question 27

Diagramme schématique des matrices mises en jeu dans l'algorithme NIPALS :

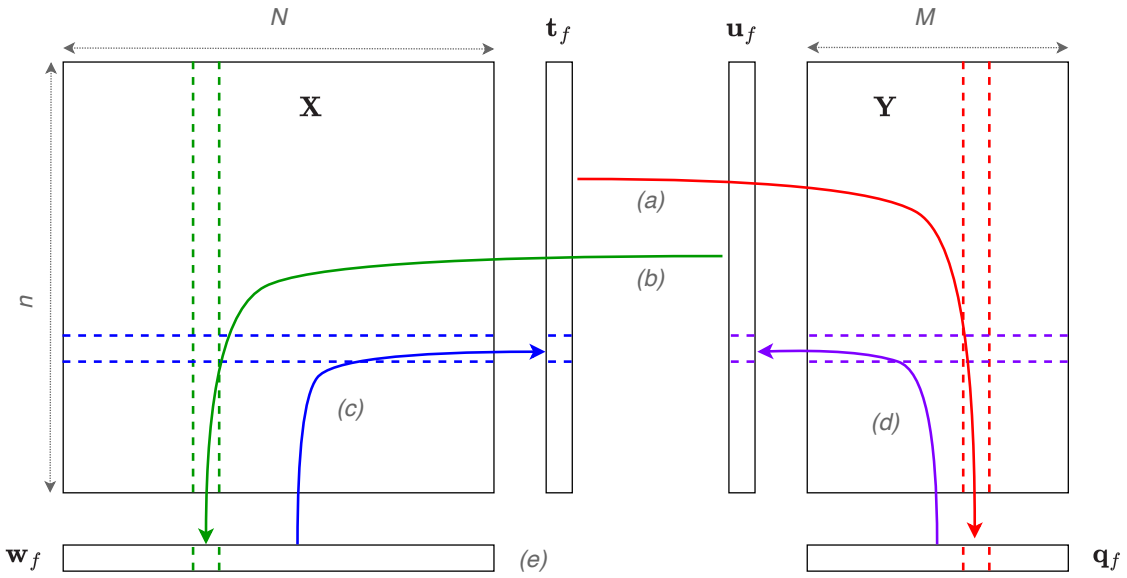


Tableau de correspondance endroit du diagramme/étape de l'algorithme :

endroit	(a)	(b)	(c)	(d)	(e)
étape (cf figure 11)					

Document réponse DR5 : Question 28

```
for comp in range(ncomp):
    train_x_mat = self.x_mat
    train_y_mat = self.y_mat
    nrt, x_nct = train_x_mat.shape
    y_nct = train_y_mat.shape[1]
    train_x_miss = np.isnan(train_x_mat)
    train_y_miss = np.isnan(train_y_mat)
    # Set u to some column of Y
    uh = train_y_mat[:, startcol]
    th = uh

    it = 0
    while True:
        # X-block weights
        wh = _____________________________
        # Normalize
        wh = _____________________________

        # X-block Scores
        th_old = th
        th = _____________________________

        # Y-block weights
        qh = _____________________________

        # Y-block Scores
        uh = _____________________________

        # Check convergence
        if np.nansum((th - th_old) ** 2) < tol:
            break
        it += 1
        if it >= maxiter:
            raise RuntimeError(
                "Convergence was not reached in {} iterations for
   component {}".format(
                    maxiter, comp
                )
            )

```

```
40      # Calculate X loadings and rescale the scores and weights
41      ph = train_x_mat.T.dot(th) / sum(th * th)
42
43      loadings[:, comp] = ph
44      scores[:, comp] = th
45      u[:, comp] = uh
46      q[:, comp] = qh
47      weights[:, comp] = wh
48      bh = _____________________________
49      b[comp] = bh
50
51      self.x_mat = self.x_mat - np.outer(th, ph)
52      self.y_mat = self.y_mat - bh * np.outer(th, qh)
```

Modèle CMEN-DOC v2 ©NEOPTEC

Nom de famille :
(Suivi, s'il y a lieu, du nom d'usage)

Prénom(s) :

Numéro Inscription :

(Le numéro est celui qui figure sur la convocation ou la feuille d'émargement)

Né(e) le : / /

(Remplir cette partie à l'aide de la notice)

Concours / Examen : …….. **Section/Spécialité/Série :** ……

Epreuve : …….. **Matière :** …….. **Session :** ……..

CONSIGNES

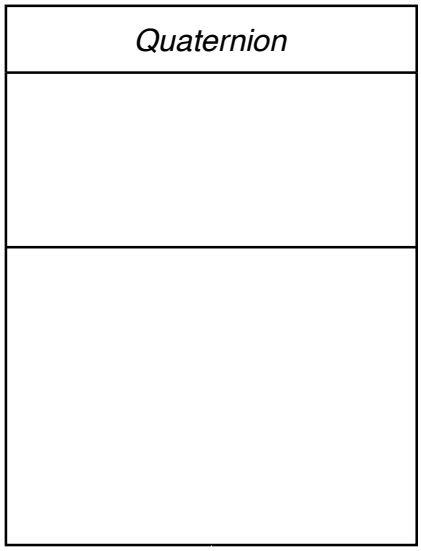
- *Remplir soigneusement, sur CHAQUE feuille officielle, la zone d'identification en MAJUSCULES.*
- *Ne pas signer la composition et ne pas y apporter de signe distinctif pouvant indiquer sa provenance.*
- *Numéroter chaque PAGE (cadre en bas à droite de la page) et placer les feuilles dans le bon sens et dans l'ordre.*
- *Rédiger avec un stylo à encre foncée (bleue ou noire) et ne pas utiliser de stylo plume à encre claire.*
- *N'effectuer aucun collage ou découpage de sujets ou de feuille officielle. Ne joindre aucun brouillon.*

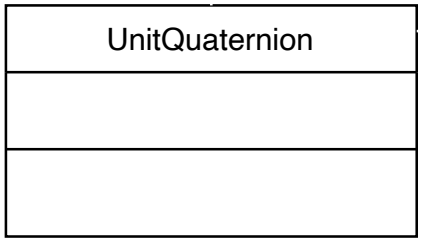
DR6 - DR7 - DR8

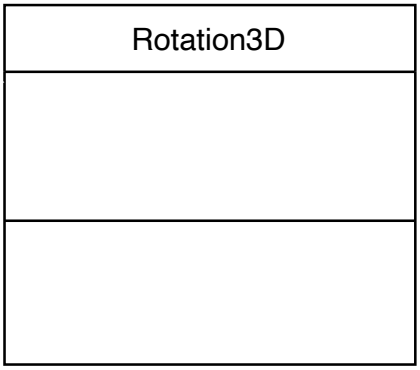
Tous les documents réponses sont à rendre, même non complétés.

NE RIEN ECRIRE DANS CE CADRE

Document réponse DR6 : Question 34







Document réponse DR7 : Question 35

```
class Rotation3D:
  def __init__(self, theta, X, Y, Z, deg=True):
    if deg:
      self.theta_deg = theta
      self.theta = np.radians(theta)
    else:
      self.theta = theta
      self.theta_deg = np.degrees(theta)
    norm = np.sqrt(X**2+Y**2+Z**2)
    self.X = X/norm
    self.Y = Y/norm
    self.Z = Z/norm
    Rq = ____________________________________
    Iq = ____________________________________
    self.q_theta = self.q_theta = UnitQuaternion(Rq, self.X*Iq, self.Y*
   Iq, self.Z*Iq)
    self.q_theta_inv = ____________________________________
```

Document réponse DR8 : Question 37

angle	-55	-104	-164	158	-179	-156	170	122	100
dangle	-49	-60	322	-338	23	326	-48	-21	
dangle_desired									
correction									