

RAPPORT DE STAGE DE FIN D'ÉTUDE D'ÉCOLE D'INGÉNIEUR.E

ECOLE CENTRALE DE MARSEILLE

LABORATOIRE DE NEUROSCIENCES COGNITIVES DE MARSEILLE

Indexer la complexité linguistique en lien avec la dynamique cérébrale à partir de signaux MEG

Auteur

Lucas BECQUET

Tuteur et tutrice

Jean-Luc BLANC

Anne-Sophie DUBARRY



Avril-Septembre 2023

Résumé

Dans ce rapport de stage de fin d'étude, on aborde la complexité des activités cognitives lors de la compréhension de phrases avec une démarche inter-disciplinaire des systèmes complexes en nous référant principalement aux formalismes de la théorie de l'information et des systèmes dynamiques [2]. Dans ce contexte, on explore la compréhension linguistique sur la base de mesures de signaux neurophysiologiques, issus de l'activité cérébrale, collectés à partir d'un magnétoencéphalographe (MEG). Ces mesures permettent de capturer la dynamique cérébrale que l'on cherche à indexer à différentes conditions expérimentales, qui reflètent différents degrés de complexité linguistique.

Sur le plan méthodologique, on a utilisé une approche basée sur la représentation symbolique d'un système dynamique. Ces méthodes permettent de quantifier la complexité intrinsèque de la dynamique cérébrale et la quantité d'information produite lors de la compréhension linguistique.

Remerciements

J'aimerais remercier tout d'abord toute la communauté scientifique à laquelle j'appartiens, qui, depuis tout temps, s'évertue à dégager les concepts du monde puis à les définir, les représenter. Sans elle, mon travail n'existerait pas.

J'aimerais surtout remercier mon tuteur Jean-Luc Blanc et ma tutrice Anne-Sophie pour leur patience. Ils ont réussi à me transmettre leur passion pour les Neurosciences et la recherche, m'ont guidé et accompagné durant tout mon stage et se sont montré.e.s compréhensif et compréhensive, même dans les moments difficiles.

J'aimerais également remercier Loïc Bonnier de l'équipe DI²S²C qui a toujours été là pour m'aider sur les aspects techniques de codage et avec qui l'ambiance était toujours stimulante et chaleureuse. J'aimerais aussi remercier Laurent Pezard qui m'a aidé et inspiré dans mes travaux et avec qui travailler était un plaisir.

Enfin, j'aimerais remercier mon père et ma mère, qui, depuis toujours, sont d'un soutien et d'une volonté de me comprendre sans égal.

Table des matières

Introduction	5
0.1 Contexte	5
0.2 Objectifs du stage	5
1 Notions de neurophysiologie	9
1.1 De l'activité neuronale aux signaux neurophysiologiques	9
1.2 La technique d'enregistrement MEG	10
2 Description du dataset (MOUS)	14
3 Pré-traitement des signaux MEG	17
3.1 Protocole de pré-traitement	18
3.2 Séries temporelles et filtre passe-haut	18
3.3 Spectre et filtre coupe-bande	19
3.4 Segmentation temporelle du signal	20
4 Analyse dans le cadre de théorie de l'information	23
4.1 Représentation symbolique d'un système dynamique	23
4.1.1 Théorie	23
4.1.2 Pratique	25
4.2 Entropie	28
4.3 Processus stochastique et taux d'entropie	30
4.4 Estimateur de Lempel-Ziv	31
5 Développement et code Python	33
5.1 Scikits-symbolic	33
5.2 Structure et classes	33
5.3 Fonctions	35
5.4 Tests sur le scikits et documentation associée	35
6 Résultats	38
6.1 Tests d'indépendance statistique	38
6.2 Interprétation des résultats	38
6.2.1 Target word vs individual word	39
6.2.2 Phrases simples vs listes aléatoires de mots issues de phrases simples	39
6.2.3 Tâche visuelle vs tâche auditive	40

6.2.4 Récapitulatif des résultats des tests statistiques	41
Conclusion	42
Bibliographie	44
Appendices	46

Introduction

0.1 Contexte

J'ai eu la chance d'effectuer mon stage au Laboratoire de Neurosciences-Cognitives de Marseille (LNC). Ce laboratoire constitue un centre de recherche de pointe dans les domaines des Neuroscience et de la Psychologie cognitive avec le Laboratoire de Psychologie Cognitive de Marseille (LPC). Ces deux laboratoires fusionneront pour former le CRPN, à partir de janvier 2024.

J'ai travaillé au sein de l'équipe de Développement Informatique et Infrastructure Système pour les Sciences du Cerveau (DI²S²C). Cette équipe de soutien à la recherche a deux missions principales. Tout d'abord, certaines membres s'occupent de la gestion, l'administration et la maintenance de l'infrastructure informatique du centre de recherche. La deuxième mission consiste en la participation à des projets de recherche en apportant notamment une expertise en informatique scientifique (développement d'applications, conseils méthodologiques, calcul scientifique etc ...). L'équipe DI²S²C est dirigée par Jean-Luc Blanc, ingénieur de recherche CNRS, qui m'a accueilli et guidé avec Anne-Sophie Dubarry, ingénierie de recherche CNRS dans la même équipe, pendant ces 6 mois. La structuration de cette équipe est détaillée par un organigramme en Figure 1. J'ai également eu l'opportunité d'échanger avec François-Xavier Alario, directeur de recherche CNRS au LPC, sur les aspects psyco-linguistiques que comprend mon travail, notamment dans les choix à effectuer pour la segmentation temporelle du signal et l'interprétation des résultats. J'ai aussi pu avoir le plaisir de travailler avec Laurent Pezard, professeur dans l'équipe Dynamique Auditivne Neuronale (DNA) au LNC, et qui m'a beaucoup apporté en termes de compétence de développement informatique et de démarche scientifique.

0.2 Objectifs du stage

Durant ce stage, nous avions pour ambition de faire le lien entre complexité linguistique et la dynamique cérébrale. Que se passe-t-il dans le cerveau, en termes d'activation de population de neurones, lorsqu'une personne comprend, ou non, une phrase qu'elle lit ou qu'elle entend ? La compréhension linguistique, qu'elle soit orale ou écrite, passe par la capacité à combiner des mots d'une phrase pour en saisir le sens ; Qui a fait quoi ? A qui ? Ce n'est pas une simple concaténation. On s'intéresse donc à la dynamique cérébrale, au processus d'opérations cognitives qui permet la compréhension de chaque mot et de leur sémantique ainsi que la syntaxe et le contexte dans lesquels ils sont inscrits pour comprendre le sens de la phrase.

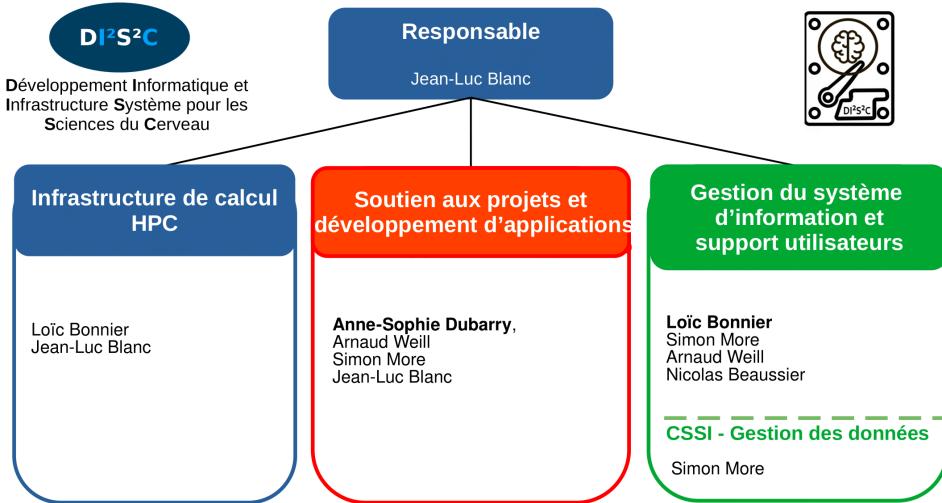


FIGURE 1 – Organigramme de l'équipe DISC

Le point de départ de mon travail repose sur des enregistrements des champs magnétiques émis par le cerveau de par son activité (signal magnétoencéphalographique ou MEG). Ces données constituent une mesure de la dynamique cérébrale que j'ai étudié. L'objectif de mon stage était donc de mettre en place un algorithme permettant d'extraire des représentations symboliques à partir de segments des séries temporelles du signal MEG (Figure 2), découpées en fonction des différents stimuli. Ces stimuli correspondent à différents degrés de complexité linguistique. Par exemple, on peut s'intéresser à comparer l'activité cérébrale lors de la lecture ou de l'écoute de phrases simples par rapport à celle lors de phrases complexes au sens grammatical. On se focalisera sur la dynamique cérébrale liée à la compréhension de chaque mot d'une phrase. Ces représentations symboliques permettent de représenter un système dynamique continu (le cerveau) comme une séquence de symboles correspondants à des états discrets du système étudié. En effet, on se base sur l'espace des phases (ou espace des états) [11] qui constitue l'espace mathématique dans lequel tous les états possibles du système sont représentés ; chaque état possible correspond à un point unique dans l'espace des phases. Cet espace permet de nous affranchir du temps en représentant les capteurs les uns en fonction des autres dans un hyperespace, où chacun d'eux représente un degré de liberté du système dynamique étudié. La représentation symbolique est une méthode qui permet d'associer un symbole à chaque partie de l'espace des phases en regard d'une partition comme illustré en (c) de la Figure ???. On a alors des séquences de symboles que l'on appelle séquences symboliques qui rendent compte de la dynamique cérébrale. En effet, chaque symbole correspond alors à un ou plusieurs états possibles (en fonction de la finesse de la partition) du système dynamique étudié. Cela permet de rendre compte de la complexité intrinsèque de la dynamique cérébrale et de la quantité d'information contenue dans celle-ci grâce à une métrique puissante : l'entropie.

Le but est donc de calculer l'entropie de représentations symboliques associées à des conditions expérimentales et de les comparer entre elles. De cette manière, on lie la complexité linguistique et sa compréhension à la dynamique cérébrale associée. Cette métrique qu'est

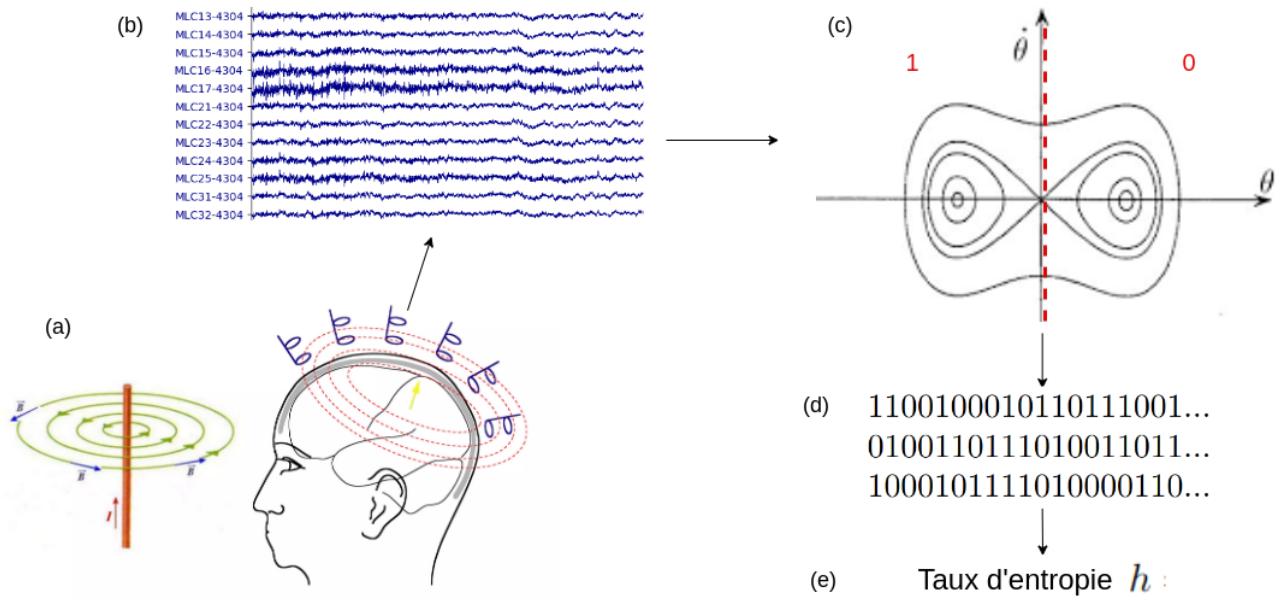


FIGURE 2 – Schéma de l'algorithme mis en place durant le stage. (a) Mesure des champs magnétiques générés de par l'activité cérébrale. (b) Séries temporelles des capteurs. (c) Représentation symbolique de l'espace des phases des capteurs avec à une partition. Ici représenté en seulement 2 dimensions avec une partition binaire, les points des trajectoires à gauche de la ligne en pointillée rouge sont associées au symbole 1 tandis que ceux à droite au symbole 0. (d) Séquences symboliques représentées pour chaque capteur. (e) Taux d'entropie de chaque séquence symbolique.

l'entropie est définie à l'intersection de 3 domaines. Dans le cadre de la théorie de l'information, l'entropie mesure la quantité d'information du signal d'origine qui a été produite par une source. Dans le cadre de la théorie des systèmes dynamiques, cette quantité rend compte de la complexité intrinsèque du système dynamique étudié en lien avec les notions d'ordre et de désordre. Enfin, dans le cadre de l'informatique théorique, l'entropie indique de la complexité du codage de la séquence symbolique permettant de décrire un phénomène observé par rapport à un alphabet universel. L'entropie est donc un indicateur, une métrique, qui nous renseigne sur la dynamique du système complexe que l'on étudie : le cerveau. Dans le cadre de cette étude, l'entropie nous permet donc mesurer l'effet de la complexité linguistique sur l'activité neuronale et de rendre compte de la dynamique cérébrale sous-jacente associée à la compréhension.

La problématique de mon stage est donc la suivante : Peut-on discriminer des conditions expérimentales et rendre compte d'une compréhension linguistique sur la base de la quantification précise de la dynamique cérébrale ?

J'étudierai dans un premier temps l'origine neurophysiologique et les techniques d'enregistrement des signaux sur lesquelles nous travaillons avant d'introduire la base de données MEG que nous avons utilisée pour cette étude. J'expliquerai ensuite le processus de pré-traitement du signal MEG ainsi que la segmentation temporelle de celui-ci en regard des conditions expérimentales. Puis, je présenterai les concepts et les méthodes de la théorie de

l'information et de la théorie des systèmes dynamiques que nous avons utilisé pour notre algorithme. Je me focaliserai principalement sur la représentation symbolique d'un système dynamique et l'entropie d'une séquence symbolique. Enfin, je détaillerai l'implémentation de l'algorithme mis en place au cours du stage et les résultats obtenus à partir des données expérimentales.

Chapitre 1

Notions de neurophysiologie

Nous avons décidé de travailler avec des données de magnétoencéphalographie comme mesure de l'activité cérébrale. Dans cette partie, nous allons alors expliquer comment sont générés ces signaux et comment en obtenir une mesure.

1.1 De l'activité neuronale aux signaux neurophysiologiques

Au sein de l'activité cérébrale, les neurones se transmettent des informations à l'aide de signaux électromagnétiques (au niveau des axones et des dendrites) et chimiques (au niveau des synapses). Les neurones corticaux sont regroupés en un réseau dense et constituent le cortex. Leur activation unitaire (c'est-à-dire le potentiel d'action) consiste en une dépolarisation transitoire de la membrane qui se propage le long des axones. Ces potentiels électriques, transmis le long des axones ou des dendrites, sont appelés courants primaires et correspondent à l'activation proprement dite des neurones [5]. Lorsque ce courant se propage dans le milieu conducteur environnant (liquide céphalo-rachidien, crâne, peau, etc.), il devient un courant de volume. Ces deux types de courant s'additionnent dans le cortex et contribuent aux signaux neurophysiologiques. Lorsque les neurones sont orientés de façon aléatoire, les courants ont tendance à s'annuler. Parce qu'elles sont orientées orthogonalement à la surface de la substance blanche (c'est-à-dire à la surface du cortex), les cellules pyramidales, elles, contribuent principalement aux signaux enregistrés en neurophysiologie. Ces différents cas de figure sont illustrés en Figure 1.1.

Lorsqu'un groupe de cellules neuronales contigüës s'active de manière synchrone, elles donnent naissance à un courant macroscopique. Lorsqu'il est détecté, il est représenté par un dipôle de courant. Comme la surface corticale est pliée, ces dipôles peuvent être orientés radialement ou tangentiellement par rapport à la surface du crâne.

La loi de Biot-Savard indique que pour un courant circulant le long d'un circuit en ligne droite, il existe un champ magnétique orienté de façon circulaire autour de cette ligne [9]. Sa direction dépend de la direction de la charge électrique et est donnée par ce que l'on appelle la "règle de la main droite". Le pouce de la main droite pointe dans la direction du flux de

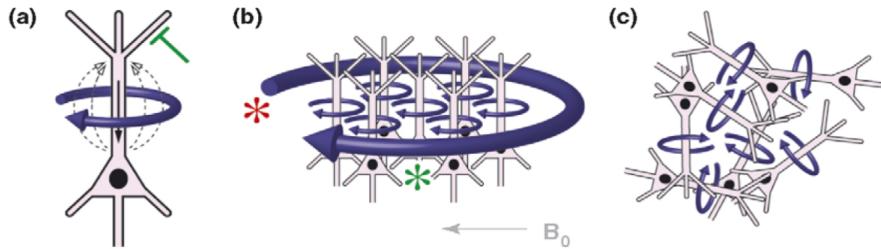


FIGURE 1.1 – Activation neuronale. (a) Neurone seul avec représentation de son potentiel électrique (ligne pointillée) et du champ magnétique induit (flèche bleue). (b) Groupe de neurones orientés dans le même sens, leur activation synchrone crée un champ magnétique résultant. (c) Neurones orientés de manière aléatoire, les champs magnétiques s'annulent. Adapté de [10]

courant (de la polarité négative à la polarité positive), les autres doigts pliés indiquent le sens du champ magnétique.

Cette propriété est importante car elle permet aux physiologistes et aux médecins d'explorer sous deux angles différents (avec des potentiels électriques et des champs magnétiques) la même activité cérébrale en recourant à des techniques différentes.

Cette activité reflète directement le traitement d'un ensemble de neurones spécifiques et peut être enregistrée de manière invasive (SEEG) ou non invasive (EEG, MEG).

1.2 La technique d'enregistrement MEG

L'activité cérébrale peut être mesurée à différentes échelles et à l'aide de différentes grandeurs. Il existe deux grandes familles de mesures de l'activité cérébrale : celles qui capturent les fluctuations hémodynamiques (modulations du flux, du volume sanguin, etc.), comme l'IRMf, et les mesures de l'activation neuronale réelle, comme l'EEG, la MEG et la SEEG (Figure 1.2). Ces dernières sont appelées techniques neurophysiologiques. Elles enregistrent l'activité électromagnétique des neurones, comme décrit dans la section précédente.

L'EEG et la MEG sont les deux principales techniques d'enregistrement non invasives utilisées pour mesurer l'activité neurophysiologique au niveau du cortex entier. Il s'agit de techniques standards largement utilisées pour l'étude des activités physiologiques ou pathologiques dans les réseaux cérébraux à grande échelle, avec une résolution temporelle de l'ordre de la milliseconde [4]. Elles se distinguent les unes des autres par la nature du signal recueilli. Alors que l'EEG mesure l'activité cérébrale en termes de potentiels électriques, la MEG enregistre les champs magnétiques provenant de l'activité cérébrale.

Il va maintenant se focaliser sur la technique d'enregistrement d'intérêt pour ce stage, à savoir, la MEG. Les paragraphes qui suivent présentent brièvement son histoire ainsi que son fonctionnement.

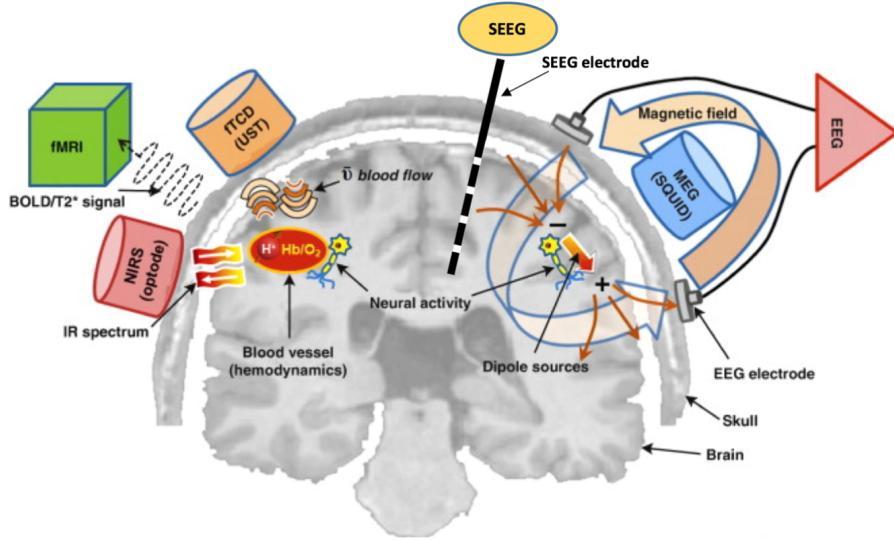


FIGURE 1.2 – Illustration des différentes techniques d’enregistrement du cerveau. À gauche : techniques mesurant l’hémodynamique. À droite : Techniques neurophysiologiques. Adapté de [15]

Les premières mesures MEG ont été réalisées à la fin des années 1960 grâce à la collaboration de deux physiciens nord-américains : David Cohen et James Zimmerman. Initialement utilisés pour mesurer l’activité cardiaque, les premiers systèmes d’enregistrement MEG n’étaient composés que de quelques capteurs ; ce n’est qu’en 1992 que le premier système pour la tête entière a été construit. Aujourd’hui, les systèmes MEG disposent de plusieurs centaines de canaux organisés en forme de casque couvrant l’ensemble de la tête. Les champs magnétiques générés par l’activité cérébrale sont de l’ordre de quelques dizaines de femtoTesla (10^{-15} Telsa jusqu’à des pico Tesla pour certaines activités cérébrales de grande amplitude telles qu’un pic épileptique). Ces champs très faibles peuvent être mesurés à l’aide de détecteurs magnétiques ultrasensibles appelés SQUID (superconducting quantum interference device), basés sur la supraconductivité et la physique quantique [7]. Couplés à des bobines d’induction, ces dispositifs, immergés dans de l’hélium liquide, peuvent capter de très faibles fluctuations du champ magnétique provenant du cerveau comme on peut l’observer schématiquement en Figure 1.4.

Comme ces champs magnétiques sont très faibles par rapport à tout autre champ magnétique (le champ magnétique terrestre qui est de l’ordre de 10^{-5} ou encore celui d’aimants qui peuvent aller jusqu’à quelques Tesla), un système MEG doit être installé dans une pièce à blindage magnétique pour permettre aux appareils de capter les faibles fluctuations. Contrairement à l’EEG, il n’existe pas de position standard pour les capteurs (il n’y a qu’une seule taille de casque). Les positions des capteurs doivent être coregistrées en fonction de l’anatomie du sujet/patient.

J’ai eu la chance de visiter le centre MEG de la Timone à Marseille, on pourra observer en Figure ?? des photos prises des différents composants de leur système MEG.

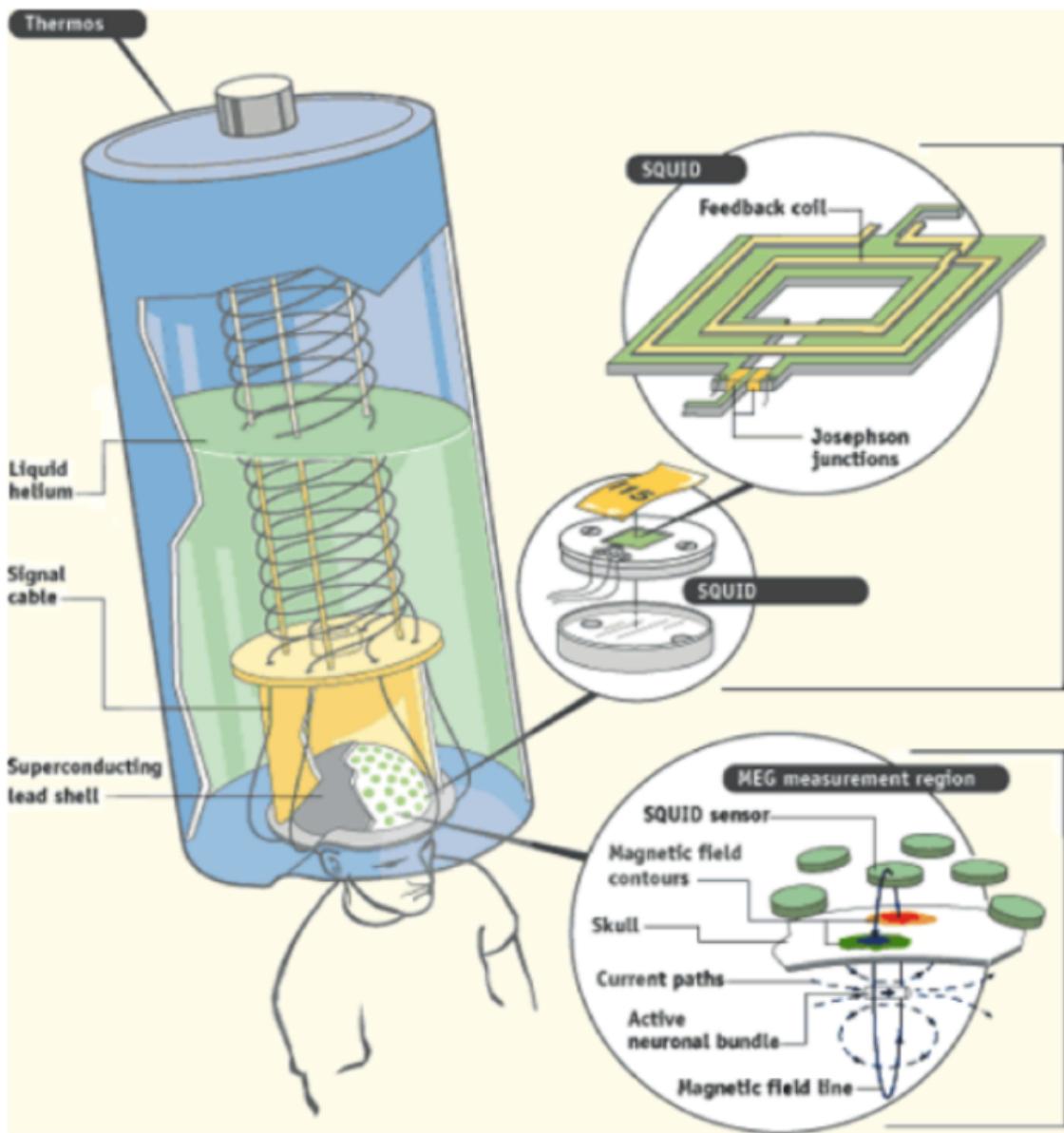


FIGURE 1.3 – Illustration de la MEG. Schéma d'un système MEG complet et de ses composants (issu de [5]).

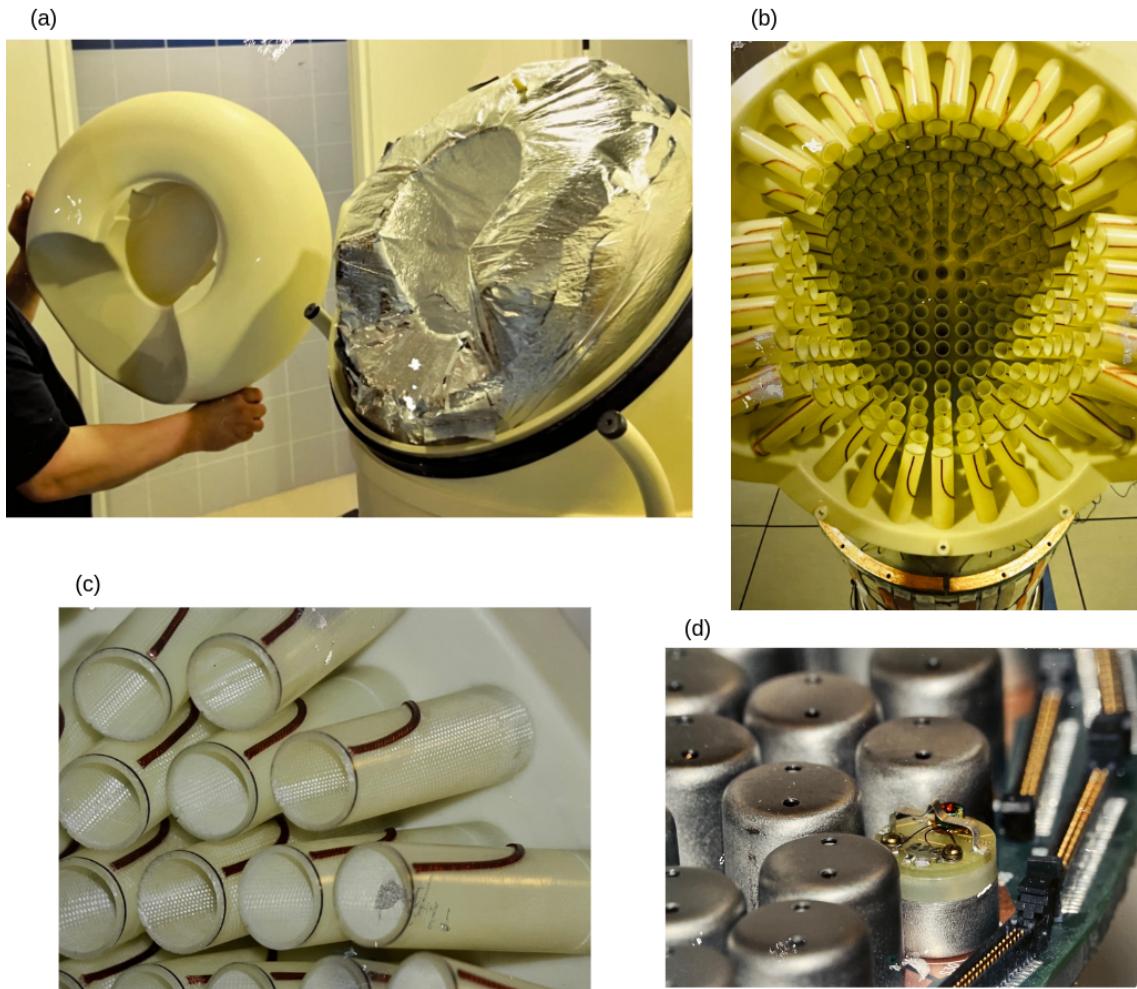


FIGURE 1.4 – Photographies des différents composants du système d’acquisition MEG, prises à différentes profondeurs. (a) Support pour le casque d’acquisition et aperçu de l’isolation du système. (b) Intérieur du dispositif une fois l’isolant retiré où l’on peut apercevoir les SQUID. (c) Gros plan sur les SQUID. (d) Plan correspondant à une prise de vue du dos de la photo précédente, on y voit les bobines.

Chapitre 2

Description du dataset (MOUS)

On va dans cette partie présenter le jeu de données MEG sur lequel j'ai travaillé et expliciter les différents stimuli de compréhension linguistique auxquels les sujets ont été soumis. Cela nous permettra de nous approprier les données et de bien comprendre le protocole expérimental auquel les sujets ont été soumis. On souhaite appliquer notre algorithme à un dataset public d'étude de la compréhension linguistique, le Mother Of Unification Studies (MOUS) [17].

L'étude a été réalisée sur un total 204 locuteurs natifs de Hollande et ayant comme langue maternelle le néerlandais. Il y a 100 hommes et 104 femmes, d'un âge moyen de 22 ans (de 18 à 33 ans). Dans la procédure de consentement éclairé, les sujets ont explicitement consenti à ce que les données anonymes collectées soient utilisées à des fins de recherche. Chaque sujet a effectué soit la tâche visuelle, soit celle auditive. Tous les sujets étaient droitiers, avaient une vision normale ou corrigée par le port de lunettes de vue. Ils n'avaient pas d'antécédents de déficits neurologiques, développementaux ou linguistiques.

Ce dataset est structuré de la façon suivante : Il y a un total de 204 sujets répartis en deux moitiés pour deux tâches de compréhension linguistique différentes, une tâche visuelle et une tâche auditive. Pour chaque sujet, on dispose d'un enregistrement MEG d'une durée d'environ 1 heure.

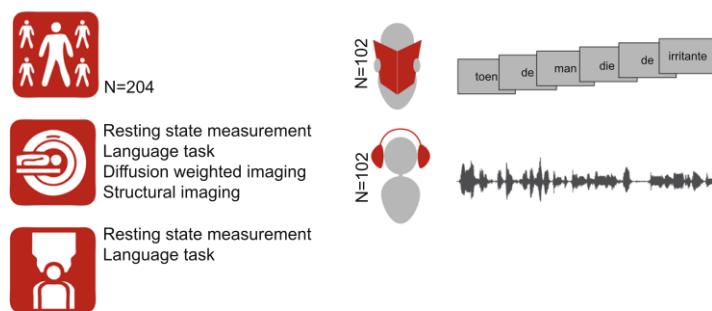


FIGURE 2.1 – Organisation du dataset. Issu de [17]

Les stimuli sont composés de 360 phrases en hollandais et leur analogues mélangés i.e une phrase ayant subi des permutations aléatoires entre les différents mots (donc inintelligibles a

priori).

On a alors

1. 360 phrases, simples ou complexes (grammaticalement parlant)
2. 360 listes aléatoires de mots

On rappelle qu'une phrase simple ne comporte qu'une seule proposition, donc un verbe conjugué. Tandis qu'une phrase complexe comporte plusieurs propositions, donc plusieurs verbes conjugués. Il y a donc naturellement plus de COD, COI et subordonnées au sein d'une phrase complexe.

Les phrases sont composées de 9 à 15 mots. Parmi ces différentes phrases, il y a :

1. 180 phrases simples appelées simple relative clause sentence (RC-)
2. 180 phrases complexes appelées relative clause sentence (RC+)

	Sentence	Word list
Complex (Relative Clause, RC+)	Het aardige vrouwtje gaf Henk die een kleurige papegaai gekocht had een zak pitjes <i>The nice lady gave Henk, who had bought a colorful parrot, a bag seeds.</i>	Zak een kleurige aardige een had die vrouwtje papegaai gaf het gekocht pitjes Henk <i>Bag a colorful nice a had who lady parrot gave the bought seeds Henk</i>
Simple (RC-)	Dit zijn geen regionale problemen zoals die op de Antillen. <i>These are no regional problems such as those on the Antilles.</i>	zoals geen die Antillen problemen regionale zijn dit op <i>such as no those Antilles problems regional are the these on</i>

FIGURE 2.2 – Exemple des différents stimuli : Phrase simple (RC- sentence), phrase complexe (RC+ sentence) et listes aléatoires de mots associées. Issu de [17]

Chaque sujet est soumis à 180 phrases simples ou complexes donc à 180 sentences (RC+ ou RC-) ainsi qu'à 180 liste de mots aléatoires ou Random lists.

Sur chaque sujet, les phrases et les listes de mots aléatoires ont été utilisé le même nombre de fois. Chaque phrase et sa liste de mots aléatoire associée contiennent un nom commun qui se trouve à la même position. Cette position peut varier de la 3ième à la 13ième position et ce mot est annoté comme cible ("target word"). Le mot placé en amont du "target word" ne diffère en taille d'un maximum de deux lettres entre la phrase et la liste de mots aléatoire associée.

Les stimuli sont soumis par blocs de 5 phrases ou 5 listes aléatoires de mots. Au début de chaque bloc, une présentation de 1500 ms indiquait le type de bloc : zinnen (phrases) ou woorden (mots). Dans les phrases, le premier mot commençait par une majuscule et le dernier mot se terminait par un point.

L'intervalle entre les essais a été réparti entre 3200 et 4200 ms. Pendant cette période, un écran vide était présenté, suivi d'une croix de fixation. C'est la phase d'instructions identifiée comme "Fixation picture, pre-trial baseline" et indexée par le nombre 20.

Ce document va nous permettre d'identifier quels stimuli sont envoyés, un à un. Lors de l'analyse, cela permettra de segmenter les séries temporelles en courtes séquences appelées

Trigger value	meaning
1	Onset of individual word (visual task) or first word (auditory task) in a Relative Clause containing sentence (RC+)
2	Onset of 'target' word in a RC+ sentence.
3	Onset of individual word (visual task) or first word (auditory task) in a word list derived from a RC+ sentence.
4	Onset of 'target' word in a word list derived from a RC+ sentence.
5	Onset of individual word (visual task) or first word (auditory task) in a sentence without a relative clause (RC-)
6	Onset of 'target' word in a RC- sentence.
7	Onset of individual word (visual task) or first word (auditory task) in a word list derived from a RC- sentence.
8	Onset of 'target' word in a word list derived from a RC- sentence.
10	Mini block instruction stimulus 'WOORDEN' (words) or 'ZINNEN' (sentences)
11	Response (index), auditory task (in visual task, this event has value 1)
12	Response (middle), auditory task (in visual task, this event has value 2)
13	Experimenter response to continue after break, auditory task (in visual task, this event has value 3)
14	Start of audio file (auditory task only)
15	Offset of word picture (visual task) or audio file (auditory task)
20	Fixation picture, pre-trial baseline.
30	Pause
40	Question

FIGURE 2.3 – Identification des différents événements et stimuli lors de l'enregistrement de la tâche de compréhension. Issu de [17]

"époques" et donc de se focaliser sur les activités cérébrales d'intérêt par rapport à la tâche de compréhension linguistique.

On verra dans la partie suivante le pré-traitement que j'ai effectué sur les données ainsi que la manière dont j'ai segmenté le signal sur la base des différentes conditions expérimentales en se basant sur les événements.

Chapitre 3

Pré-traitement des signaux MEG

Avant de pouvoir appliquer l'algorithme développé durant le stage, il nous faut préparer les données. Il existe différentes librairies et logiciels qui permettent le traitement de données MEG. Il me fallait une librairie développée sous le même langage de programmation que celui que j'allais utiliser pour implémenter mon algorithme, de plus je bénéficiais du travail d'un ancien stagiaire, Hiroyoshi Yamasaki, qui avait réalisé un projet de recherche sur la compréhension linguistique avec MNE-Python. J'ai donc, sous les conseils de mes encadrants, choisi d'utiliser MNE-Python (<https://mne.tools/stable/index.html> [6] qui est une librairie complète reconnue internationalement et développée sous Python. Cette librairie permet de manipuler des données MEG avec des fonctions de pré-traitement et d'analyse déjà implémentées. J'ai travaillé sous Linux et j'ai donc créé un environnement virtuel dans lequel j'ai téléchargé les modules et librairies Python dont j'avais besoin (MNE Python, numpy) afin d'avoir un environnement de codage stable et indépendant des mises à jour système. Cela me permet également de donner précisément l'état de mon environnement avec toutes les versions des librairies et toolbox utilisées pour que mon travail puisse être reproductible, suivant ainsi les bonnes pratiques scientifiques dans ce domaine [16]. Il existe trois grandes marques de systèmes d'acquisition MEG qui ont des caractéristiques différentes. Les enregistrements issus de ces différents systèmes se présentent sous la forme de structure de données de différents formats/types. Dans notre cas, l'enregistrement des données MEG a été réalisé avec une MEG de marque CTF. Ce système d'acquisition est essentiellement caractérisé par ses capteurs dits "gradiomètres axiaux". Un des avantages de MNE-Python est qu'il permet de simplement spécifier le type de système d'acquisition pour importer les données.

Les données MEG sont des données très complexes qui contiennent un certain nombre de meta-données permettant au Data analyst de retrouver le protocole d'enregistrement ainsi que les paramètres d'acquisition utilisés. En effet, on crée avec MNE-Python un objet que l'on appelle raw (pour brut, les données telles qu'elles ont été enregistrées) et qui contient les données MEG relatives à l'enregistrement d'un sujet. On manipule ensuite cet objet, on peut notamment afficher les meta-données relatifs à l'enregistrement (3.1) avec la commande raw.info().

On prendra l'exemple d'un enregistrement de la tâche visuelle pour illustrer les propos.

Measurement date	January 01, 1970 10:46:00 GMT
Experimenter	Unknown
Participant	Anonymized297739_1534851269_0
Digitized points	7 points
Good channels	3 Stimulus, 44 misc, 28 Reference Magnetometers, 273 Magnetometers, 4 EEG
Bad channels	None
EOG channels	Not available
ECG channels	Not available
Sampling frequency	1200.00 Hz
Highpass	0.00 Hz
Lowpass	600.00 Hz
Filenames	sub-V1001_task-visual_meg.meg4 sub-V1001_task-visual_meg.1_meg4 sub-V1001_task-visual_meg.2_meg4 sub-V1001_task-visual_meg.3_meg4
Duration	01:07:08 (HH:MM:SS)

FIGURE 3.1 – Meta-données d'un enregistrement de la tâche de compréhension visuelle. On peut observer les différents canaux, y accéder aussi plus en détails, on voit aussi que la fréquence d'échantillonnage est de 1200Hz

3.1 Protocole de pré-traitement

On présente ici (Figure 3.2) le protocole de pré-traitement des données qui a été mis en oeuvre durant le stage.

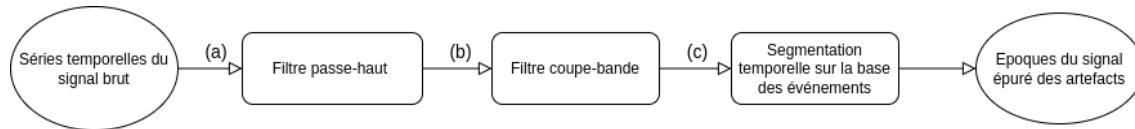


FIGURE 3.2 – Protocole de pré-traitement des données MEG. (a) On applique d'abord un filtre passe-haut. (b) On applique un filtre coupe-bande. (c) On effectue la segmentation temporelle et donc la création d'époques des séries temporelles du signal MEG

J'ai donc effectué les étapes de pré-traitement suivantes :

1. Un filtre passe-haut avec une fréquence de coupure de 1Hz afin de corriger la dérive lente ou "slow-drift" des séries temporelles ;
2. Un filtre coupe-bande autour de 50Hz permettant de corriger le bruit des lignes électriques du signal MEG ;
3. Une segmentation des séries temporelles en courtes époques en se basant sur les différentes conditions expérimentales afin d'appliquer l'algorithme mis en place durant le stage sur ces segments temporels d'intérêt.

3.2 Séries temporelles et filtre passe-haut

Nous visualisons dans un premier temps le signal temporel pour repérer de potentiels artefacts visibles comme les battements du coeur ou les clignements des yeux.

Les artefacts limités à une plage de fréquences étroite peuvent parfois être réparés en filtrant les données. Les dérives lentes et le bruit des lignes électriques sont deux exemples d'artefacts limités à une plage de fréquences. Ce sont donc ces artefacts que nous avons traité

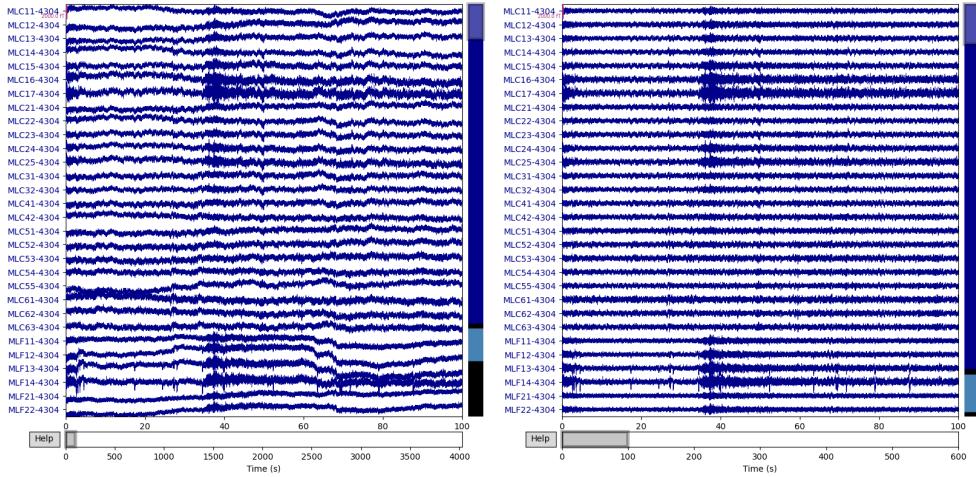


FIGURE 3.3 – A gauche, séries temporelles des différents canaux du signal MEG brut. A droite, séries temporelles après application du filtre passe-haut

afin de modifier le moins possible le signal d'origine et ainsi conserver de manière intacte la dynamique cérébrale.

On observe que les signaux ne sont pas centrés sur l'origine, ils dérivent. C'est là une dérive lente que l'on peut réparer en filtrant notre signal avec un filtre passe-haut avec une fréquence de coupure de 1Hz .

Nous avons appliqué un filtre passe-haut pour corriger la dérive lente des séries temporelles. Par rapport, aux filtres coupe-bande, cela ne pose pas de problèmes car les activités fréquentielles du cerveau sont définies sur une plage de fréquence $\geq 1\text{Hz}$. Après application du filtre passe-haut, le signal est à présent centré autour de 0.

3.3 Spectre et filtre coupe-bande

Il est essentiel de représenter les amplitudes en fonction des fréquences contenues dans le signal, comme en Figure 3.4. Ce type de représentation est appelé spectre ; elle permet de mettre en évidence le contenu fréquentiel du signal. On observe un pic à 50Hz ainsi que ses harmoniques. Cette composante fréquentielle correspond au bruit dû à l'alimentation en courant du système d'acquisition, c'est le bruit des lignes électriques. Cet artefact environnemental se manifeste par des oscillations persistantes centrées autour de la fréquence de la ligne électrique, d'où le pic observé.

J'ai donc utilisé un filtre coupe-bande à 50Hz pour retirer la fondamentale du bruit des lignes électriques. J'ai fait le choix de ne pas appliquer de filtres coupe-bande pour retirer les harmoniques de cet artefact ($100\text{Hz}, 150\text{Hz}, \dots$) car même si la bande passante est très fine, cela risquerait de modifier significativement tout le signal et de perdre des informations quand à l'activité neuronale.

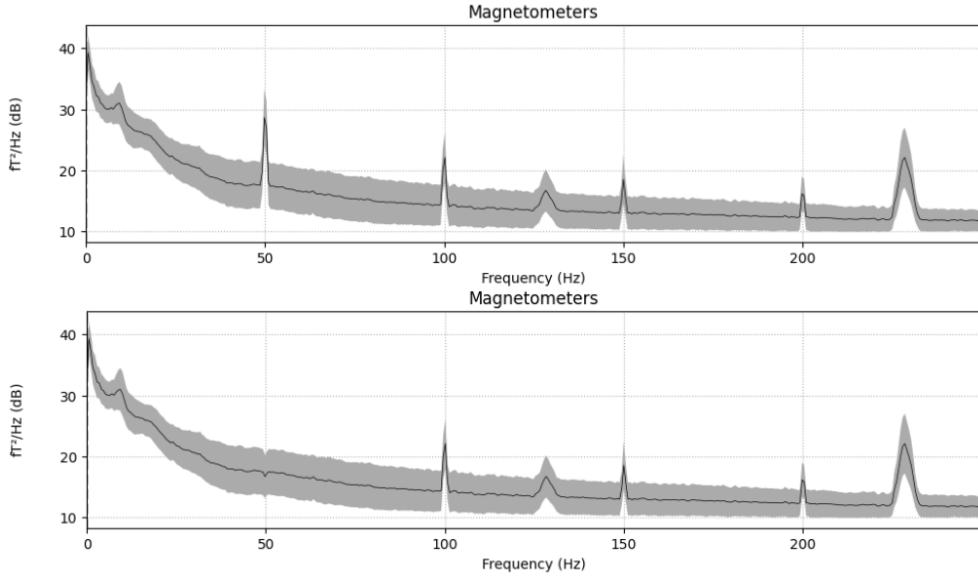


FIGURE 3.4 – Graphe de la densité spectrale relativ à l'enregistrement d'un sujet de la tâche visuelle. En haut, le spectre du signal brut. En bas, le spectre du signal une fois le filtre coupe-bande à 50Hz appliqué

3.4 Segmentation temporelle du signal

Une fois que les artefacts gênants pour l'étude ont été retirés des signaux MEG, il en va de le découper en courtes séquences que l'on appelle époques. En effet, pour réduire le rapport signal sur bruit, il faut appliquer l'algorithme développé au cours du stage sur une répétition de plusieurs segments temporels correspondants à un stimuli du même type (par exemple un mot cible d'une phrase simple), à une même condition expérimentale auquelle le sujet a été soumis.

Les événements, dont on a présenté les différents types dans le chapitre précédent (Figure 2.3), sont une des méta-données de l'enregistrement MEG contenu dans le canal UPPT001 et nous renseigne sur la distribution des stimuli au cours du temps. Ces époques vont donc être sélectionnées et découpées à partir des indexées temporelles des différents événements (events). En effet, lors de l'enregistrement relatif à un sujet, il est renseigné une liste d'indexées temporelles qui donne la trace du protocole expérimental. Cela permet de connaître le début d'une phrase et la caractérisation de celle-ci (simple, complexe, liste aléatoire de mots associée) et plus précisément l'apparition sur l'écran de chaque mot d'une phrase. Les événements sont cruciales pour l'analyse des données et un choix d'époques judicieux est indispensable pour obtenir des résultats significatifs. En effet, une fenêtre de temps trop grande, en prenant par exemple comme époque le signal MEG sur toute une phrase, a comme inconvénient de diluer l'activité cérébrale. Les premiers mots d'une phrase avant que la compréhension ne s'opère vont avoir une activité similaire à ceux d'une liste aléatoire de mots. Il est donc préférable de choisir une fenêtre de temps resserée autour de chaque mot pour définir les époques, en s'intéressant tout particulièrement aux target words.

On extrait les événements relatifs au protocole d'un sujet à partir du canal UPPT001 des données MEG. Cela nous permet de créer un objet event contenant les index temporels des différents événements. On renseigne ensuite un dictionnaire que l'on nomme event-id et dans lequel on fait correspondre les étiquettes (qui sont des entiers) des événements avec ce qu'ils signifient dans le protocole d'enregistrement (Figure 2.3). On peut visualiser en Figure 3.5 le positionnement des événements dans le temps ainsi que le nombre de fois qu'ils apparaissent (sur 600 secondes d'enregistrement seulement, d'où l'absence de certains événements) :

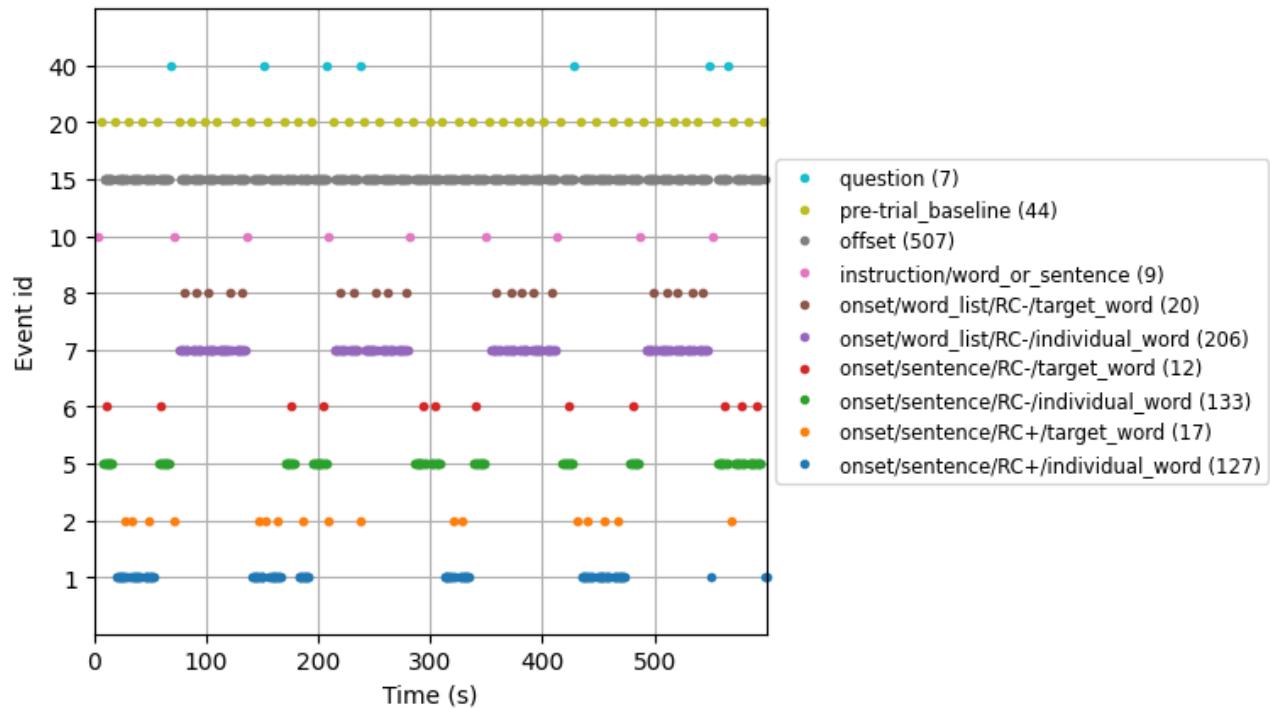


FIGURE 3.5 – Visualisation des différents événements au cours du temps d'un sujet de la tâche visuelle

Par exemple, l'événement 5, "onset/sentence/RC+/individual-word" signifie l'affichage sur l'écran du sujet d'un mot individuel (qui n'est pas un target word) d'une phrase complexe. Sur la base de ces objets, on va créer des époques autour des événements d'intérêts.

Tout d'abord, on détermine l'écart maximum entre deux mots (et donc entre deux événements correspondants) d'une même phrase ou liste aléatoire de mots, cela nous permet de définir la durée que l'on choisira pour créer nos époques. Pour la tâche visuelle, on décide construire les époques en découplant le signal temporel sur chaque canaux 0.2 s avant l'affichage du mot et 1.2 s après. Ce qui nous fait des époques d'une durée de 1.4 s.

On discrimine ensuite les époques en fonction de l'événement d'intérêt. Dans notre cas, on construit des époques centrées autour des target words, d'autres autour des individual words, tout ça au sein des phrases simples et des listes de mots aléatoires issues de phrases simples. L'idée est de pouvoir comparer la dynamique cérébrale lors de la compréhension mot

par mot entre la tâche visuelle et la tâche auditive, les phrases simples et leurs listes de mots aléatoires associées, les target words et les individual words.

Pour chaque époque, on a donc une matrice ayant pour lignes les 301 canaux et pour colonnes les valeurs que prennent les canaux sur le segment temporel que constitue l'époque, i.e., 1200 points.

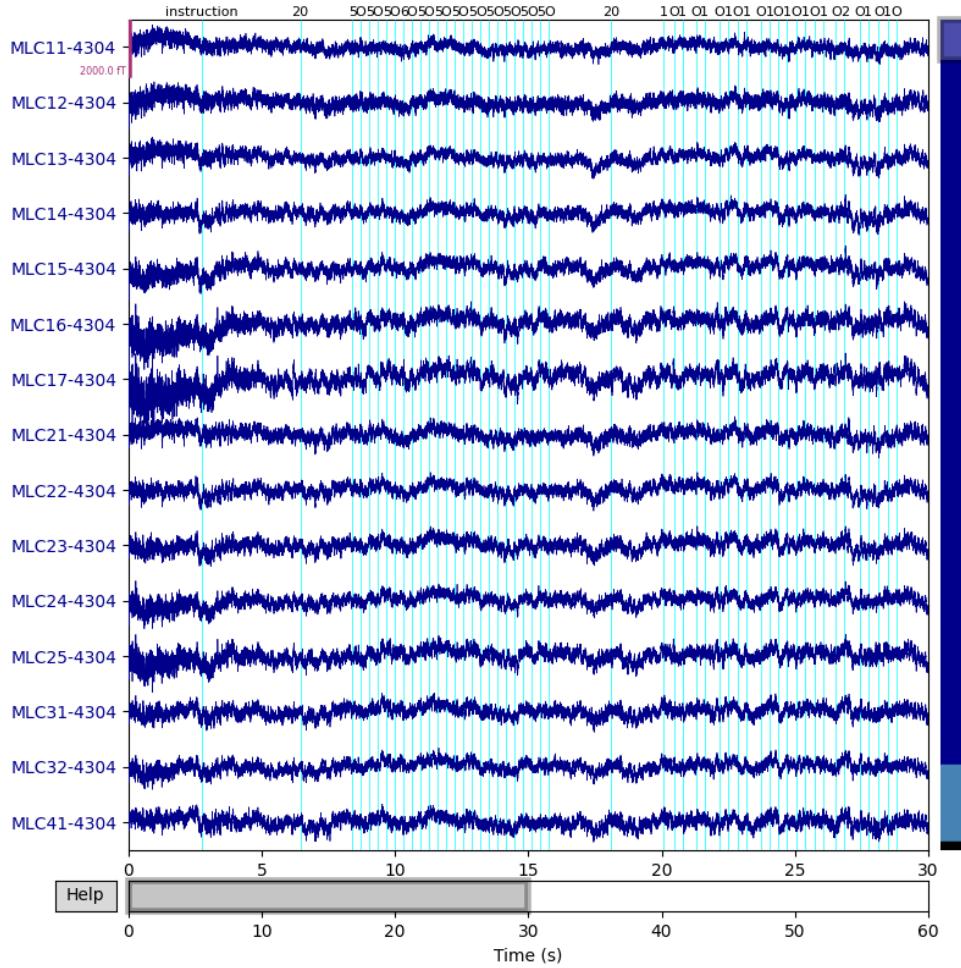


FIGURE 3.6 – Visualisation des événements directement sur les différents canaux du signal temporel

Les données sont prêtes et nous avons créé les époques à partir des séries temporelles sur la base des différentes conditions expérimentales que nous souhaitons comparer. Nous pouvons à présent appliquer la partie centrale de l'algorithme mis en place durant le stage : la représentation symbolique de la dynamique cérébrale de conditions expérimentales et le calcul de l'entropie associée.

Chapitre 4

Analyse dans le cadre de théorie de l'information

Dans cette section, je vais introduire les fondements théoriques de l'approche de la théorie de l'information que j'ai utilisé. Plus particulièrement, la représentation symbolique et les indicateurs : l'entropie et le taux d'entropie d'une séquence symbolique.

4.1 Représentation symbolique d'un système dynamique

Je vais présenter le concept de représentation symbolique d'un système dynamique, i.e., comment extraire des séquences de symboles, qui rendent compte de la complexité du système dynamique d'intérêt, à partir de séries temporelles. Cela dans le but de calculer ensuite l'entropie associée à ce système.

4.1.1 Théorie

La méthodologie sur laquelle repose la démarche scientifique que nous avons choisi d'appliquer pendant mon stage est centrée autour du concept de représentation symbolique. En effet, on souhaite calculer l'entropie relative à différentes conditions expérimentales, i.e., un target word, un mot individuel, issu d'une phrase simple ou bien d'une phrase complexe, pour pouvoir ensuite comparer les entre elles. N'ayant pas accès de manière déterministe à la dynamique cérébrale mais ayant comme mesures de celle-ci les séries temporelles du signal MEG, il n'est pas possible de calculer directement l'entropie. En se basant sur la théorie des systèmes dynamiques [8], on va donc passer par un système dynamique discret isomorphe au système dynamique d'étude, c'est ce que l'on appelle la représentation symbolique.

On définit dans un premier temps l'espace des phases (ou espace des états) [11] qui constitue l'espace mathématique dans lequel tous les états possibles d'un système dynamique sont représentés ; chaque état possible correspond à un point unique dans l'espace des phases comme illustré en Figure 4.1. Cet espace permet de nous affranchir du temps en représentant les capteurs les uns en fonction des autres dans un hyperespace, où chacun d'eux représente un degré de liberté du système dynamique étudié, en l'occurrence ici, la dynamique cérébrale.

La représentation symbolique peut être vue comme le fait de transformer les trajectoires dans l'espace des phases d'un système dynamique (séries temporelles) en une séquence d'états

discrets représentés par un ensemble fini de symboles qui constitue les étiquettes des éléments de la partition. L'ensemble des symboles d'une séquence symbolique est appelé alphabet et correspond à la partition utilisée. Celui-ci est lié à la séquence symbolique et nous donne la trace de la représentation symbolique en question, i.e., la partition qui a été utilisée pour créer des représentations symboliques à partir des séries temporelles du système dynamique étudié.

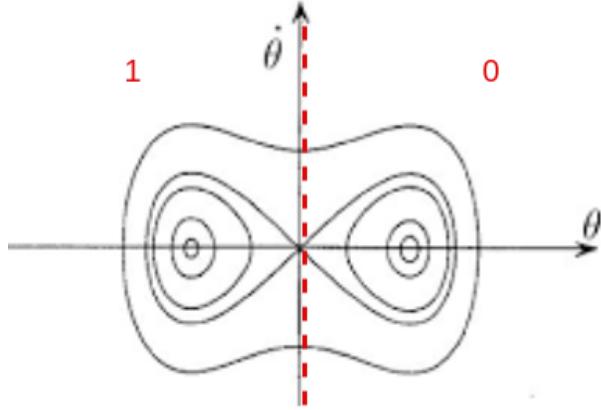


FIGURE 4.1 – Illustration de la représentation symbolique dans l'espace des phases. Ici représenté en seulement 2 dimensions avec une partition binaire, les points des trajectoires à gauche de la ligne en pointillé rouge sont associés au symbole 1 tandis que ceux à droite sont associés au symbole 0.

La dynamique symbolique revient donc à échanger des séries temporelles à valeurs continues d'un système dynamique contre des séquences symboliques avec un alphabet fini d'un système dynamique discret, qui sont les représentations symboliques. Les symboles de ces représentations sont les états discrets de la dynamique symbolique. Une séquence symbolique peut donc être vue comme un processus stochastique où la collection de variables aléatoires prend comme valeurs des symboles codés à partir d'un alphabet. En effet, Une partition divise l'espace des phases en un nombre fini de régions. Chaque région se voit attribuer un symbole. Toute orbite de longueur infinie peut alors être convertie en une séquence symbolique en enregistrant la région de l'espace des phases que la trajectoire visite à chaque instant. Une représentation symbolique est donc une séquence de symboles qui rendent compte des états discrets d'un système dynamique au cours du temps par rapport à une partition que l'on souhaite être génératrice dans l'espace des phases (espace des états).

L'entropie d'une séquence symbolique est une mesure qui nous donne énormément de renseignement. Dans le cadre de la théorie de l'information, comme on vient de le voir, l'entropie de Shannon nous renseigne sur la production d'information d'une source et l'imprévisibilité de celle-ci. Dans le cadre des systèmes dynamiques, l'entropie de Kolmogorov-Sinai $h_\mu(f) = \text{sup} h_\mu(f, \alpha)$ [8] représente la quantité d'informations nécessaire pour reconstruire une trajectoire dans l'espace des phases à partir d'une condition initiale. Cela nous donne en fait la complexité de la dynamique intrinsèque du système observé et dans quelle mesure celui-ci est chaotique, i.e., désordonné.

Le lecteur ou la lectrice trouvera les détails et les définitions mathématiques des concepts

que l'on vient d'expliciter en annexe.

4.1.2 Pratique

La première manière de représenter symboliquement la trajectoire associée à notre système dynamique est d'extraire une séquence symbolique pour chaque série temporelle des canaux. Son principe est assez simple et on traite les canaux un par un. En effet, pour chaque canal, on mesure l'amplitude crête à crête du signal pour savoir dans quel intervalle le signal prend ses valeurs. On détermine alors l'histogramme des valeurs du canal en question que l'on va diviser par équpartition. En effet, à partir du nombre de symboles que l'on décide de fixer, i.e. la taille de l'alphabet, on découpe uniformément l'intervalle des valeurs en autant d'intervalles que l'on souhaite de symboles. On effectue ensuite l'attribution des symboles, i.e., les entiers qui sont étiquettes de chaque intervalles, aux points de la série temporel grâce à la fonction `digitize` du module `numpy`.

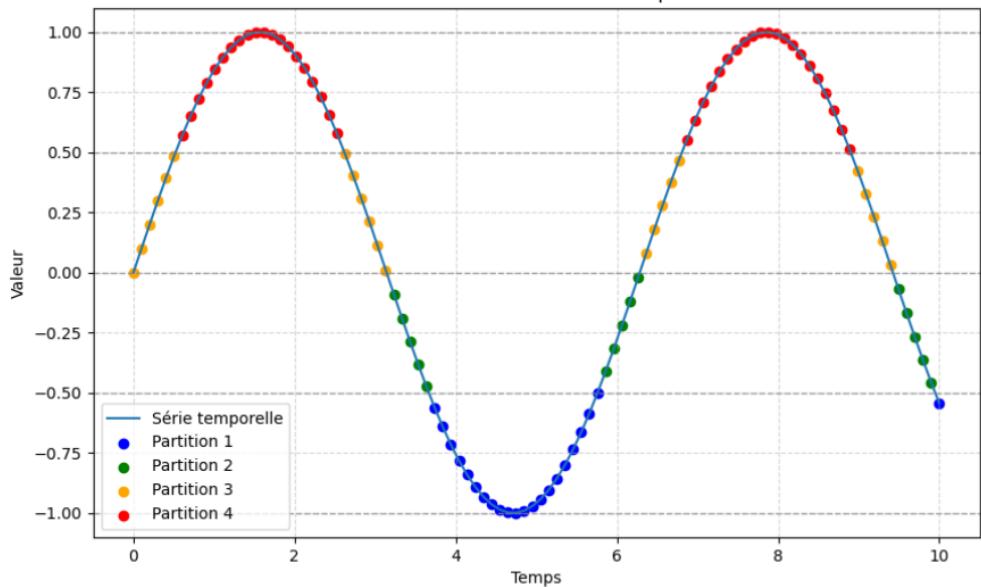


FIGURE 4.2 – Illustration d'une équpartition par histogramme des valeurs d'une série temporelle. Exemple d'une partition à 4 symboles d'une sinusoïde. On peut identifier les différents intervalles qui correspondent chacuns à un symbole de la partition

En y associant un alphabet par défaut de la même taille que le nombre de symboles, on a alors une représentation symbolique de chaque canal et il suffit alors de créer les objets séquences symboliques correspondants grâce à la classe `sequence` du module `scikits-symbolic` que l'on présentera dans le chapitre suivant.

La deuxième façon d'effectuer une représentation symbolique de notre système dynamique que nous avons utilisé est une représentation symbolique dans l'espace des phases des capteurs. Pour se faire, on effectue une décomposition en valeurs singulières (SVD) [1] de la matrice de nos données (capteurs \times temps), plus précisement, comme la convention est

d'avoir une matrice des données (temps \times valeurs), on fait une décomposition en valeurs singulières de la transposée de notre matrice de données.

Soit M la transposée de la matrice des données $m \times n$, la décomposition en valeurs singulières est définie par

$$M = USV^T \quad (4.1)$$

1. $U \in \mathcal{R}^{m \times n}$ contient un ensemble de vecteurs orthonormés de \mathcal{R}^m , dits "de sortie"
2. S est une matrice diagonale de $\mathcal{R}^{n \times n}$ qui contient les valeurs propres de $M^T M$
3. $V^T \in \mathcal{R}^{n \times n}$ contient un ensemble de vecteurs orthonormés de \mathcal{R}^n , dits "d'entrée". Ce sont les vecteurs propres associés aux valeurs propres de $M^T M$ contenues dans S

La SVD est une méthode qui permet notamment de faire une réduction de dimension de nos données tout en conservant un maximum d'informations sur la variance de celles-ci.

On projette ensuite nos données dans un sous-espace formé par les vecteurs propres ayant les valeurs propres les plus grandes. Le choix du nombre de valeurs propres que nous gardons est fait en fonction du critère de sélection de Schwarz [18] qui revient à regarder le décrochage au niveau de la pente lorsque l'on représente les valeurs propres par ordre décroissant comme on peut l'observer en Figure 4.3.

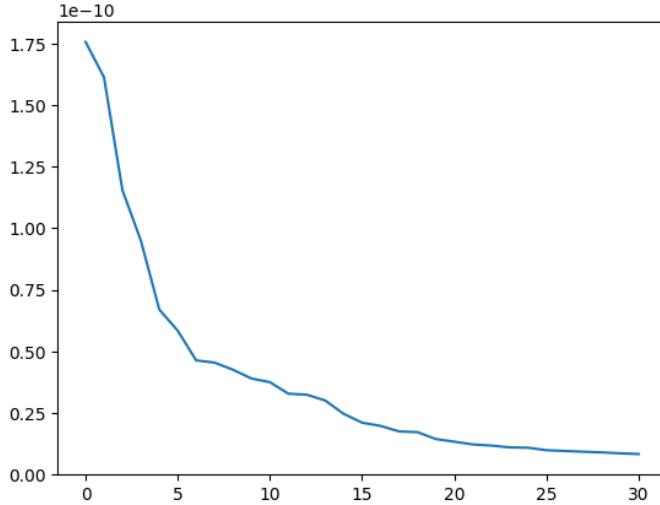


FIGURE 4.3 – Graphe des 30 premières valeurs propres issues de la SVD de la matrice des données pour un sujet de la tâche visuelle

Dans notre cas, cela revient à prendre les 10 premières valeurs propres. Les vecteurs propres que nous gardons correspondent aux directions selon lesquelles on retrouve le plus d'informations quand à la variabilité des valeurs des champs magnétiques au cours du temps mesurés par les capteurs. Ce sont les vecteurs propres associés aux valeurs propres qui permettent d'expliquer le plus, quantitativement parlant, la variance de nos données. Dans cet espace de moindre dimension, chaque dimension, i.e., chaque valeur singulière, correspond à des combinaisons linéaires des différents capteurs.

On ne garde donc qu'une partie d des vecteurs propres contenus dans V de sorte à avoir une matrice $U' \in R^{n \times d}$. De la même manière, on crée $S' \in R^{d \times n}$ en tronquant S et $V' \in R^{n \times d}$. On obtient alors notre matrice des données réduite par $M' = U'S'V'^T$.

On représente ces nouveaux canaux les uns en fonction des autres, pour former un espace des phases des valeurs singulières des capteurs. C'est dans ce nouvel hyperespace que l'on va déterminer l'histogramme des valeurs selon toutes les dimensions afin de déterminer la partition associée et découper cet hyperespace en autant d'intervalles que l'on souhaite avoir de symboles pour nos représentations symboliques. J'ai illustré cet méthode avec les données d'un sujet de la tâche visuelle en Figure 4.4.

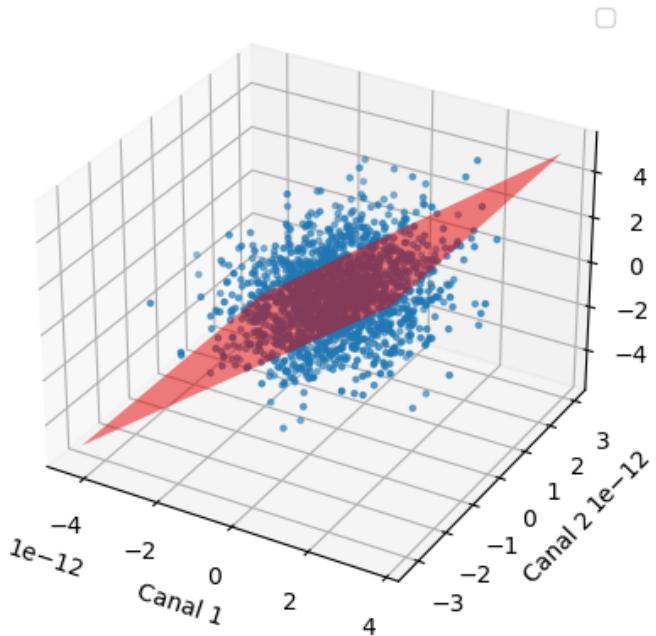


FIGURE 4.4 – Espace des phases de 3 valeurs singulières avec une partition binaire

On récupère alors une partition selon chaque axe qui nous permet de créer nos séquences symboliques.

Dans le cas où l'on fait une représentation symbolique dans l'espace des phases des données projetées sur une partie des vecteurs propres, i.e., l'espace des phases des valeurs singulières, deux choix s'offre à nous. On peut directement quantifier la dynamique de chaque séquence symbolique ou bien de créer une nouvelle séquence symbolique globale à partir des séquences symboliques obtenues. On obtient alors une séquence symbolique qui représente la dynamique globale du cerveau plutôt que par valeurs singulières, quand bien même quantifier la dynamique de chaque séquence permet d'obtenir un vecteur de taux d'entropie qui quantifie la dynamique cérébrale globale. La création d'une unique séquence symbolique résultante a du sens avec la méthode de représentation symbolique utilisée car les séquences symboliques sont représentées à partir des valeurs singulières des données, i.e., des combinaisons linéaires des capteurs. Pour se faire, on utilise la méthode recode de la classe séquence que l'on détaillera

dans le chapitre suivant. Cette fonction permet de créer une séquence symbolique avec un nouvel alphabet à partir de plusieurs séquences symboliques et le fait que leur alphabet soit tous différents.

Que ce soit avec la première méthode ou la seconde, on peut ensuite quantifier la dynamique de chaque séquence symbolique et ainsi comparer la dynamique associée à différentes conditions expérimentales. C'est ce que l'on va approfondir par la suite grâce à la métrique qu'est l'entropie.

4.2 Entropie

Les éléments de la théorie de l'information présentés ci-dessous sont des résultats issus de [20]. On présentera donc seulement les définitions utiles pour notre étude. Le concept d'entropie a été définie pour la première fois par Shannon en 1948 [19]. Cette quantité est une mesure de l'incertitude d'une variable aléatoire. Soit X une variable aléatoire discrète ayant \mathcal{X} comme alphabet (ensemble des valeurs ou symboles que peut prendre la variable aléatoire X) et $p(x) = \Pr\{X = x\}, x \in \mathcal{X}$ comme fonction de masse de probabilité. L'entropie $H(X)$ d'une variable aléatoire discrète X est définie par

$$H(X) = - \sum_{x \in \mathcal{X}} p(x) \log p(x) \quad (4.2)$$

Lorsqu'on s'intéresse à une seule variable aléatoire X , on peut aussi noter l'entropie $H(p)$ où p désigne la fonction de masse de probabilité de la variable X . Le \log est en base 2, l'unité de l'entropie est donc le bit. Il est d'usage également utiliser le logarithme népérien en base e pour d'autres domaines que la compression et la transmission de données. En fonction de la base du logarithme choisie, si par exemple celle-ci est b , on notera l'entropie associée $H_b(X)$.

Voici quelques propriétés importantes de l'entropie :

1. $H(X) \geq 0$
2. $H(p)$ est une fonction concave de p .
3. $H_b(X) = (\log_b a) H_a(X)$

Illustrons la notion d'entropie et ses propriétés de base avec un exemple simple, celui d'un tirage aléatoire binaire [2]. Comme on peut l'observer sur la Figure 4.5, on a $H(p) = 0$ lorsque $p = 0$ ou $p = 1$ et $H(p) = 1$ lorsque $p = 1/2$. Cela revient à dire que l'entropie est nulle lorsque la variable n'est plus aléatoire (i.e. non incertaine) et que l'entropie atteint sa valeur maximale lorsque l'incertitude est maximale.

En considérant une paire de variables aléatoires discrètes (X, Y) de distribution joint $p(x, y)$ comme une seule variable aléatoire vectorielle, on obtient aisément la définition de l'entropie jointe $H(X, Y)$ par

$$H(X, Y) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log p(x, y) \quad (4.3)$$

On peut également définir l'entropie conditionnelle d'une variable aléatoire sachant une autre variable aléatoire comme la moyenne par rapport à la variable aléatoire de condition-

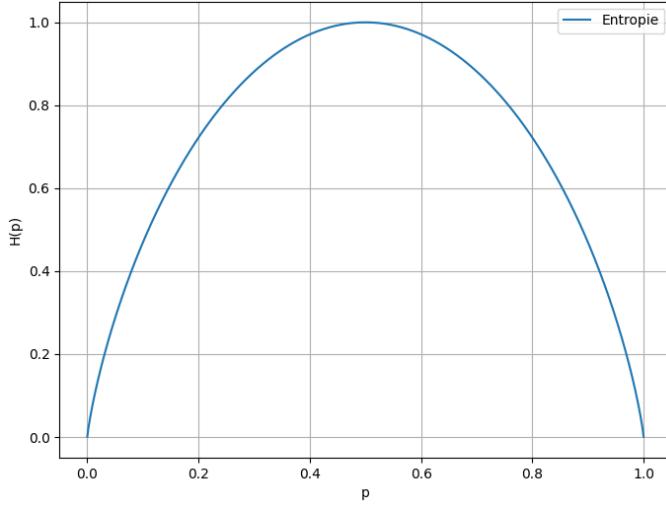


FIGURE 4.5 – Entropie $H(p)$ en fonction de la probabilité p d'un événement binaire

nement des entropies des distributions conditionnelles. Si $(X, Y) \sim p(x, y)$, alors l'entropie conditionnelle $H(Y|X)$ est donnée par

$$H(Y|X) = \sum_{x \in \mathcal{X}} p(x) H(Y|X = x) = - \sum_{x \in \mathcal{X}} p(x) \sum_{y \in \mathcal{Y}} p(y|x) \log p(y|x) \quad (4.4)$$

Ainsi, on a

$$H(Y|X) = - \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log p(y|x) \quad (4.5)$$

L'entropie jointe et l'entropie conditionnelle d'une paire de variables aléatoires sont reliées par :

$$H(X, Y) = H(X) + H(Y|X) \quad (4.6)$$

Ce qui nous permet d'introduire la règle de la chaîne pour l'entropie. Afin de pouvoir définir le taux d'entropie d'un processus stochastique dans la section suivante, on généralise la règle de la chaîne pour une collection de n variables aléatoires. En effet, l'entropie d'une collection de variables aléatoires est défini par la somme des entropies conditionnelles.

Soit X_1, X_2, \dots, X_n une collection de variables aléatoires distribuées selon $p(x_1, x_2, \dots, x_n)$.

On a alors

$$H(X_1, X_2, \dots, X_n) = \sum_{i=1}^n H(X_i | X_{i-1}, \dots, X_1) \quad (4.7)$$

Nous allons à présent détailler notre démarche quand au calcul du taux d'entropie d'une séquence symbolique.

4.3 Processus stochastique et taux d'entropie

Les séquences symboliques extraites à partir des segments temporels des données MEG peuvent donc être considérées comme des processus stochastiques.

D'après [20], un processus stochastique $\mathbf{Z} = \{Z_i\}$ est une séquence indexée de variables aléatoires. En général, il peut y avoir une dépendance arbitraire entre les variables aléatoires. Le processus est caractérisé par les fonctions de masse de probabilité conjointes

$$Pr\{(Z_1, Z_2, \dots, Z_n) = (z_1, z_2, \dots, z_n)\} = p(z_1, z_2, \dots, z_n) \quad (4.8)$$

, $(z_1, z_2, \dots, z_n) \in \mathcal{Z}^n$ pour $n = 1, 2, \dots$

Une caractéristique importante d'un processus stochastique est la stationnarité. Un processus stochastique est dit *stationnaire* si la distribution conjointe de tout sous-ensemble de la séquence de variables aléatoires est invariante par rapport aux changements de l'indice de temps ; c'est-à-dire,

$$Pr\{Z_1 = z_1, Z_2 = z_2, \dots, Z_n = z_n\} = Pr\{Z_{1+l} = z_1, Z_{2+l} = z_2, \dots, Z_{n+l} = z_n\} \quad (4.9)$$

Dans notre cas, on fera l'assumption que les séquences symboliques extraites des données MEG sont des processus stochastiques stationnaires. En effet, la représentation symbolique du signal en entier ne peut pas être considéré comme un processus stochastique mais il est stationnaire par morceaux, i.e. localement stationnaire. En choisissant des segments temporels courts (1.4 sec) centrés sur des événements clairement identifiés (comme par exemple l'apparition sur l'écran d'un target word lors de la tache visuelle), on peut considérer les représentations symboliques extraites de ces segments comme stationnaires.

On a vu dans la partie précédente, que l'on pouvait calculer l'entropie d'une collection de variables aléatoires $H(Z_1, Z_2, \dots, Z_n)$ (5.7). On s'intéresse à la limite lorsque n tend vers l'infini de cette métrique que l'on appelle taux d'entropie. Le taux d'entropie permet de mesurer des structures de récurrence et donc la production d'information de la source, c'est l'accélération de l'entropie. Le taux d'entropie d'un processus stochastique $\mathbf{Z} = \{Z_i\}$ est alors défini d'après [20] par

$$H(\mathbf{Z}) = \lim_{n \rightarrow \infty} \frac{1}{n} H(Z_1, Z_2, \dots, Z_n) \quad (4.10)$$

Quand la limite existe. Ce taux d'entropie correspond à la limite des entropies par symboles des variables aléatoires.

On peut aussi définir une quantité liée pour le taux d'entropie [20] d'après (4.7) telle que

$$H'(\mathbf{Z}) = \lim_{n \rightarrow \infty} H(Z_n | Z_{n-1}, Z_{n-2}, \dots, Z_1) \quad (4.11)$$

Lorsque la limite existe. Cette deuxième notion du taux d'entropie correspond à la limite de l'entropie conditionnelle de la dernière variable aléatoire du processus stochastique sachant les états des variables aléatoires précédentes. Pour un processus stochastique stationnaire, les limites convergent et on a égalité entre les deux taux.

$$H(\mathbf{Z}) = H'(\mathbf{Z}) = h \quad (4.12)$$

Ce résultat est asymptotique alors que nous disposons d'un jeu de données fini et donc de séquences symboliques finies. Il en va de déterminer le taux d'entropie en utilisant un estimateur, c'est ce que nous allons présenter maintenant.

4.4 Estimateur de Lempel-Ziv

L'ultime partie de notre algorithme est le calcul du taux d'entropie des séquences symboliques issues des séries temporelles des données MEG. Le taux d'entropie étant asymptotique et les séquences que l'on génère étant finies, on va donc utiliser un estimateur de cette quantité. Nous avons décidé d'utiliser l'estimateur de Lempel-Ziv [14]. En effet, les séquences symboliques extraites à partir des époques des séries temporelles MEG contiennent 1200 symboles et la convergence de la complexité de Lempel-Ziv vers le réel taux d'entropie a été prouvée [14] pour de courtes séquences, i.e., des séquences de moins de 1000 symboles. De plus, c'est un estimateur sans paramètres libres qui constitue une véritable mesure du taux d'entropie. Enfin, l'estimateur de Lempel-Ziv avait déjà été implémenté par Jean-Luc Blanc et Laurent Pezard dans le module scikits-symbolic. C'est pour ces raisons que nous avons choisi d'utiliser cet estimateur du taux d'entropie. Nous allons à présent expliquer son fonctionnement.

Considérons une source stationnaire qui émet à chaque pas de temps un symbole d'un alphabet de taille finie k . Son entropie de bloc d'ordre n est définie comme l'entropie de Shannon de la distribution de probabilité $p_n(w)$ des mots de longueur n (i.e. des mots de n symboles),

$$H_n = - \sum_w p_n(w) \ln p_n(w) \quad (4.13)$$

1. w mot de longueur n
2. $p_n(w)$ distribution de probabilité

Cette somme est calculée sur tous les mots de longueur n possibles w , et dépend donc de la dynamique de la source sur les n intervalles de temps. En conséquence, $h_n = H_{n+1} - H_n$ converge vers une limite h lorsque la longueur des mots tend vers ∞ , ce qui correspond au taux d'entropie de la source.

$$h = \lim_{n \rightarrow \infty} \frac{H_n}{n} = \lim_{n \rightarrow \infty} H_{n+1} - H_n \quad (4.14)$$

Ces deux définitions du taux d'entropie correspondent dans l'ordre aux définitions établies en (4.10) et en (4.11) d'après [20]. Les équations (4.12) et (4.14) sont donc équivalentes.

En pratique, le taux d'entropie h doit le plus souvent être estimé à partir d'une seule séquence observée $[s] = (s_i)_{1 \geq i \geq N}$ de longueur N . Nous désignerons dorénavant \hat{X} comme l'estimateur d'une quantité X , sans mentionner explicitement qu'il dépend de la séquence $[s]$.

et de sa longueur N . Regarder la limite du taux d'entropie lorsque n tend vers l'infini revient à observer la limite de cette même quantité lorsque N tend vers l'infini.

Le point de vue adopté pour calculer la complexité de Lempel-Ziv est a priori très différent de celui associé au taux d'entropie de Shannon h . En effet, la définition du taux d'entropie de Shannon h implique une caractéristique globale de la dynamique, à savoir sa mesure invariante. Elle peut être calculée à partir de la connaissance d'une seule trajectoire tant que la mesure est ergodique et permet de reconstruire la distribution de probabilité de la source à partir de l'observation d'une seule séquence typique. Mais elle n'est pas, en soi, significative en tant que caractéristique d'une séquence unique. En revanche, la complexité de Lempel-Ziv fournit une mesure de la compressibilité de la séquence symbolique unique considérée, en d'autres termes, le contenu de l'information par symbole. Dans l'hypothèse où la source est stationnaire et ergodique, les théorèmes de Lempel-Ziv [13] garantissent que la complexité de Lempel-Ziv coïncide avec h jusqu'à un facteur $\ln k$ impliquant le nombre k de symboles de l'alphabet (la partition utilisée) comme défini en (4.17). Cette hypothèse implique en effet que presque toutes les séquences symboliques ont les mêmes caractéristiques de compressibilité ; le calcul peut donc être effectué de manière équivalente avec n'importe quelle séquence typique et son résultat coïncide avec la moyenne.

Selon le schéma de Lempel-Ziv, la séquence de longueur N est découpée en \mathcal{N}_w mots. Deux manières de découper la séquence et donc de construire le dictionnaire de mots ont été proposées. On utilisera la première manière de créer des mots à partir des symboles de la séquence, publié en 1976 [12], qui considère comme mot le plus court ensemble de symboles qui n'a pas encore été rencontré lors du balayement de la séquence. Par exemple :

$$1 \cdot 0 \cdot 01 \cdot 11 \cdot 100 \cdot 101 \cdot 00 \cdot 010 \cdot 11 \dots \quad (4.15)$$

Une fois le dictionnaire de mots construit à partir de la séquence, on calcule

$$\hat{L} = \frac{\mathcal{N}_w[1 + \log_k \mathcal{N}_w]}{N} \quad (4.16)$$

où

$$\lim_{n \rightarrow \infty} \hat{L} = \frac{h}{\ln k} \quad (4.17)$$

1. k taille de l'alphabet
2. \mathcal{N}_w nombre de mots du dictionnaire
3. N taille de la séquence symbolique

C'est donc de cette manière que nous avons calculé la complexité de Lempel-Ziv \hat{L} , qui nous donne une très bonne estimation du taux d'entropie d'une séquence symbolique. L'algorithme de Lempel-Ziv que nous avons utilisé fait partie du module scikits-symbolic sur lequel j'ai apporté une contribution et que je vais présenter dans le prochain chapitre.

Chapitre 5

Développement et code Python

5.1 Scikits-symbolic

Dans cette section, nous allons décrire l'implémentation des approches de la théorie de l'information pour pouvoir appliquer notre algorithme sur les données MEG. Pour se faire, j'ai travaillé sur un module initialement développée par Jean-Luc Blanc et Laurent Pezard il y a quelques années. Nous avons travaillé de manière collaborative en utilisant le logiciel GitLab. Le Git du module scikits-symbolic est disponible à cette adresse <http://git.lnc.dcs.univ-amu.fr/lpezard/scikits-symbolic/>.

Le module scikits-symbolic est un ensemble de fonctions, de classes et de méthodes de la théorie de l'information permettant de manipuler les séquences symboliques. En effet, j'ai utilisé le module scikits-symbolic pour représenter symboliquement la dynamique cérébrale en déterminant des séquences symboliques à partir des canaux du signal MEG et d'en calculer leur taux d'entropie. C'est donc dans cette perspective que j'ai alimenté le code déjà existant.

J'ai donc travaillé avec Laurent Pezard sur le scikits-symbolic afin d'obtenir une version stable en y ajoutant les tests nécessaires pour pouvoir le publier en tant que Scikit. Ce travail m'a permis de manipuler des métriques issus de la théorie de l'information en lien avec mon sujet telles que les séquences symboliques, les fonctions de partition et de représentation symbolique dans l'espace des phases. J'ai donc pu affiner ma compréhension des outils que j'ai par la suite utilisé lors de mon stage.

Toutefois, l'objectif de mon travail sur ce module était essentiellement d'implémenter une batterie de tests sur les scripts pour vérifier et prouver leur bon fonctionnement. Nous détaillerons plus tard cet aspect dans la partie dédiée.

5.2 Structure et classes

L'arborescence principale du scikits-symbolic est présentée en Figure 5.1.

On affiche ici seulement les principaux fichiers .py sans expliciter le contenu des autres répertoires. Toutefois, il est important de noter que le module est également composée de

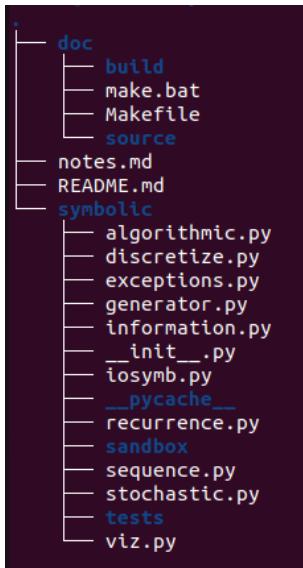


FIGURE 5.1 – Arborescence (de profondeur 2) du module scikits-symbolic

fichiers .rst dans les répertoires souce et build permettant la création de la documentation via Sphinx, que l'on détaillera plus tard, ainsi que d'autres fichiers sous-jacents permettant le fonctionnement du module scikits-symbolic. On va ici présenter seulement les différents fichiers sur lesquels j'ai apporté une contribution.

Le fichier central de scikits-symbolic est sequence.py où sont définis les différentes classes ainsi que leurs méthodes associées. Il y a 3 classes différentes, la classe State permet de définir un état. On peut créer un état, celui-ci représente et constitue le lien entre un entier et un symbole (qui peut-être une lettre d'un alphabet).

La seconde classe est l'Alphabet. L'alphabet est une liste d'état, il peut être créé à partir d'une liste de States définis préalablement ou bien en donnant simplement le nombre de symboles que l'on souhaite dans notre alphabet. Cet objet constitue une des deux bases indispensables à la définition d'une représentation symbolique. En effet, l'alphabet permet de conserver la représentation de la dynamique symbolique comme un vecteur de nombres entiers (facilement manipulable pour des calculs de la théorie de l'information) tout en ayant le lien visible avec la signification de chaque entiers de par leur association avec les symboles de l'alphabet.

La dernière classe qui est au centre de nos intérêts et constitue la pièce centrale du scikit est donc l'objet Sequence qui permet de définir informatiquement une séquence symbolique. On voit donc que ces 3 classes sont imbriquées les unes dans les autres. La définition d'un objet séquence symbolique nécessite un vecteur qui est la séquence d'entier ainsi qu'un alphabet. Et la création d'un alphabet nécessite la définition des différents States qui le constituent. De cette manière, la classe sequence permet d'avoir un objet informatique contenant toutes les propriétés et informations nécessaires pour correspondre au concept de séquence symbolique et donc à sa définition mathématique. Les méthodes implémentées pour la classe sequence permettent donc en autres d'obtenir la taille d'une séquence symbolique ainsi que la taille de

son alphabet, de calculer son entropie de Shannon, etc. Il est également possible de créer une séquence symbolique à partir de deux séquences symboliques à condition qu'elles soient de même taille gracie à la fonction recode. Cela est possible pour deux séquences symboliques ayant des alphabets différents et même de taille différente. Cette fonction crée donc un nouvel alphabet qui prend en compte les symboles contenu dans les alphabets d'origine. Comme expliqué dans la partie 4.2.2, c'est une fonction que l'on a utilisé pour obtenir une séquence symbolique globale à partir de séquences symboliques représentées dans l'espace des phases de séries temporelles singulières des capteurs (projetés après une SVD). Cette séquence symbolique permet alors de rendre compte de la dynamique cérébrale globale.

5.3 Fonctions

Le reste du scikit est un ensemble de fonctions et de méthodes permettant de manipuler et de faire des opérations et des calculs au sens de la théorie de l'information sur des séquences symboliques que nous présentons ici :

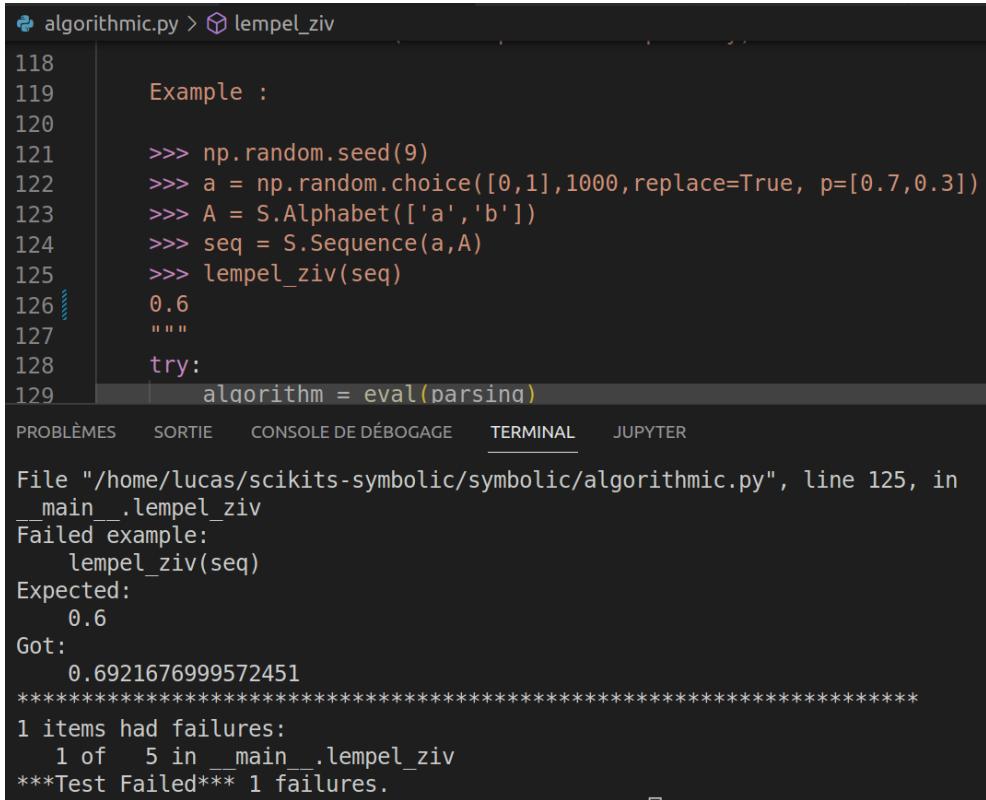
1. algorithmic : contient les différentes versions de l'estimateur du taux d'entropie de Lempel-Ziv ;
2. stochastic : permet le calcul des matrices de transition, matrices conditionnelles et d'influence de deux séquences symboliques qui représentent des processus stochastiques ;
3. information : permet le calcul de de différentes métriques issues de la théorie de l'information. Parmis celles-ci, on pourra citer l'information mutuelle de deux séquences symboliques ou plus, l'entropie de Shannon, etc ;
4. discretize : contient les fonctions permettant l'extraction de séquences symboliques à partir de séries temporelles

Les fonctions du fichier discretize correspondent aux implémentations des méthodes de représentation symbolique explicitées dans la partie 4.2.

5.4 Tests sur le scikits et documentation associée

La convergence et les fondements théoriques des différents codes du module scikits-symbolic sur lesquels j'ai effectué les tests ont déjà été prouvés. Mon travail était donc de vérifier la correction des différents algorithmes. J'ai implémenté les tests directement sur le code, grâce au module Doctest qui permet d'écrire un test directement dans la définition de la fonction ou de la classe. On écrit les tests entre guillemets, comme si l'on écrivait dans un document texte. En effet, le module Doctest permet d'écrire des prompts de commande que l'ordinateur va exécuter lorsque qu'il va run le code. Comme on peut le voir en Figure 5.2, on écrit des commandes comme si les écrivait directement dans un shell Python avec "»>" en amont pour indiquer une requête. Les lignes sans "»>" correspondent à ce qu'est censé renvoyé l'ordinateur lors de la lecture des commandes, ici 0.6. On peut donc lui faire exécuter une suite de commandes qui utilisent les fonctions et les objets du module scikits-symbolic sur

un exemple dont on connaît déjà le résultat. De cette manière, on vérifie que l'implémentation des scripts qui composent le module a été correctement effectuée et que l'algorithme produit les résultats attendus. C'est la vérification informatique de la correction de l'algorithme. Si la sortie est égale avec ce que l'ordinateur a calculé en suivant notre suite de commande, alors le test est concluant et aucun messages d'erreur ne s'affiche. Par contre, si le résultat obtenu est différent de ce qui est censé être trouvé, un message d'erreur s'affiche et cela permet de mettre en exergue un mauvais fonctionnement du code qui ne fait pas ce que l'on voudrait qu'il fasse (Figure 5.2).



```

algorithmic.py > lempel_ziv
118
119     Example :
120
121     >>> np.random.seed(9)
122     >>> a = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
123     >>> A = S.Alphabet(['a','b'])
124     >>> seq = S.Sequence(a,A)
125     >>> lempel_ziv(seq)
126     0.6
127     """
128
129     try:
130         algorithm = eval(parsing)
PROBLÈMES SORTIE CONSOLE DE DÉBOGAGE TERMINAL JUPYTER
File "/home/lucas/scikits-symbolic/symbolic/algorithmic.py", line 125, in
__main__.lempel_ziv
Failed example:
    lempel_ziv(seq)
Expected:
    0.6
Got:
    0.6921676999572451
*****
1 items had failures:
  1 of  5 in __main__.lempel_ziv
***Test Failed*** 1 failures.

```

FIGURE 5.2 – Exemple d'un test réalisé avec Doctest échoué. Expected représente ce que l'on a indiqué comme étant la sortie de notre ligne de commande "lempel-ziv(seq)", i.e., ce qu'est censé obtenir l'ordinateur lorsqu'il run le script en question. J'ai volontairement mis un résultat erroné pour pouvoir afficher le message d'erreur d'un test echoué. Got correspond au résultat effectivement obtenu par l'ordinateur lors de la lecture du script.

Cette manière de tester en écrivant directement dans la définition d'une classe ou d'une fonction allait de paire avec la création d'une documentation pour le scikits-symbolic. En effet, le générateur de documentation Python Sphinx permet de générer facilement une documentation en prenant les informations contenues dans la définition d'une classe ou d'une fonction. Il suffit alors de décrire ce que fait notre code, de renseigner les différents paramètres avec une certaine typologie, d'écrire les test Doctest, tout cela au même endroit dans le code. Quelques manipulations des outils apportés par Sphinx nous permettent ensuite de générer un document pdf ou bien une page html en décidant de la structure. La documentation est

consultable en deuxième partie de l'Annexe.

Chapitre 6

Résultats

6.1 Tests d'indépendance statistique

Je présente ici les résultats obtenus en utilisant la méthode de représentation symbolique par canaux, pour deux sujets, un de la tâche visuelle et un de la tâche auditive, sélectionnés de manière aléatoire. Pour chaque ensemble d'époques relatives à une condition expérimentale, on extrait des représentations symboliques dont on estime le taux d'entropie grâce à l'estimateur de Lempel-Ziv. On obtient alors une matrice contenant le taux d'entropie les séquences symboliques extraites à partir de segments temporels d'intérêt des canaux du signal MEG (ce sont les lignes de la matrice) et ce pour chaque époque (colonnes). Pour chaque canal, on calcule la médiane sur les époques que l'on garde. On a alors un vecteur contenant les taux d'entropie relatifs à chaque canal.

On décide d'effectuer des tests-t statistiques [3] entre les vecteurs d'entropie associés à des conditions expérimentales différentes. Par exemple, on va comparer la mesure obtenue sur les target words au sein des phrases avec celle obtenue sur les target words au sein des listes aléatoires de mots. Le test-t nous donne un résultat important, il nous informe si la différence observée entre les deux mesures est statistiquement significative.

Etant donné que nos mesures proviennent du même sujet, même s'il a été soumis à des stimuli différents, on peut donc considérer nos vecteurs d'entropie comme appariés. En effet, le test-t pour échantillons appariés peut être utilisé pour des sujets qui ont été exposés à deux conditions expérimentales. Les quantités que l'on compare sont des taux d'entropie qui constituent bien une mesure au sens mathématique du terme et l'on a utilisé la même métrique pour chaque condition expérimentale.

6.2 Interprétation des résultats

On s'intéresse à la valeur p ou "p-value" comme résultat du test-t. Celle-ci nous donne la probabilité de l'hypothèse nulle, i.e., que les deux populations sont statistiquement identiques. Lorsque l'on obtient une valeur p inférieure à 5%, $p \geq 0.05$, on peut considérer que la probabilité de l'hypothèse nulle correspond au hasard et donc que les deux échantillons sont statistiquement différents.

On présente ici les résultats que nous avons obtenus en comparant les échantillons de taux d'entropie relatifs à différentes conditions expérimentales.

6.2.1 Target word vs individual word

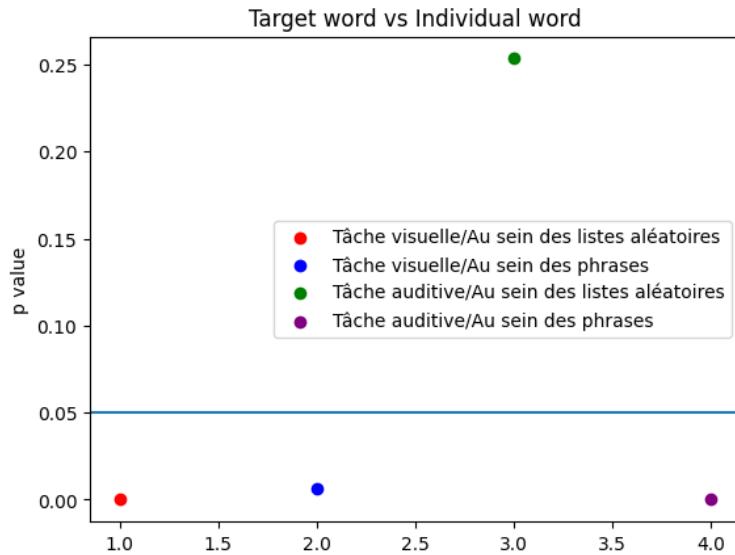


FIGURE 6.1 – Tests t appariés entre les vecteurs d'entropie relatifs aux target words par rapport aux individual words

On observe que pour la tâche visuelle, les dynamiques cérébrales associées aux target words et aux individual words sont statistiquement différentes, que ce soit au sein des phrases ou des listes aléatoires de mots. Pour la tâche auditive, les dynamiques cérébrales associées aux targets words et aux individual words sont statistiquement différentes au sein des phrases tandis qu'elles sont statistiquement différentes au sein des listes aléatoires de mots. On peut donc confirmer que pour la tâche auditive, la dynamique cérébrale permet effectivement de distinguer des degrés de complexité linguistique en lien avec des conditions expérimentales différentes.

6.2.2 Phrases simples vs listes aléatoires de mots issues de phrases simples

On retrouve ici ce que l'on a pu observer sur le graphique de résultats précédent. En effet, les target words semblent être les piliers de compréhension des phrases, ce qui permet d'indexer la complexité linguistique avec la dynamique cérébrale. En effet, les mesures de la dynamique cérébrale sont statistiquement différentes entre les phrases simples et les listes aléatoires de mots lorsque l'on s'intéresse aux targets words. Ce n'est pas le cas lorsque l'on se base sur les individual words, où les dynamiques cérébrales associées sont statistiquement semblables.



FIGURE 6.2 – Tests t apparaçus entre les vecteurs d'entropie relatifs aux phrases par rapport aux listes aléatoires de mots

6.2.3 Tâche visuelle vs tâche auditive

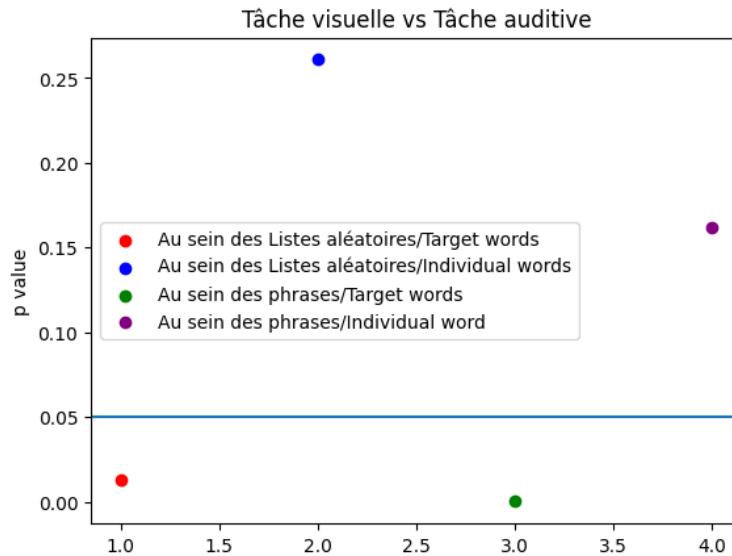


FIGURE 6.3 – Tests t apparaçus entre les vecteurs d'entropie relatifs à la tâche visuelle par rapport par rapport à la tâche auditive

On observe que les dynamiques cérébrales associées à la tâche visuelle et auditive sont statistiquement différentes lorsque l'on se base sur les target words, que ce soit au sein des phrases ou bien des listes aléatoires de mots. Encore une fois, la compréhension linguistique auditive et visuelle ne s'opérant pas de la même manière, on peut les discriminer grâce à la dynamique cérébrale en se basant sur les target words, que ce soit au sein des phrases

ou des listes aléatoires de mots. Comme on pouvoit s'y attendre, cette conclusion n'est pas possible lorsque l'on s'intéresse aux individual word. En effet, les dynamiques associées sont statistiquement semblables dans la tâche auditive comme dans la tâche visuelle.

6.2.4 Récapitulatif des résultats des tests statistiques

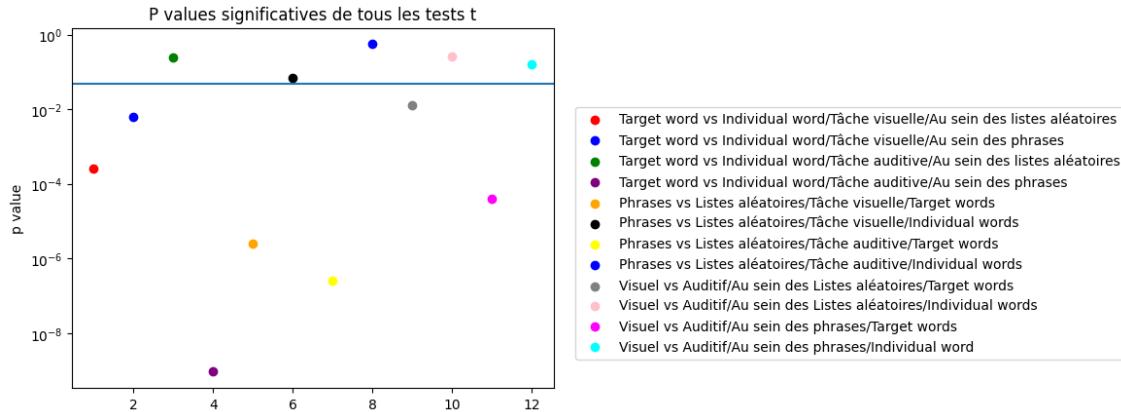


FIGURE 6.4 – Valeurs p significatives des tests t. L'axe des ordonnées est en échelle logarithmique afin de pouvoir distinguer les différentes valeurs

On peut conclure qu'avec la démarche scientifique que l'on a appliquée et la segmentation temporelle effectuée, le taux d'entropie est majoritairement un bon indicateur pour indexer le complexité linguistique. Toutefois, la dynamique cérébrale ne permet pas de discriminer des conditions expérimentales à chaque fois.

Conclusion

A travers le travail que j'ai mené durant mon stage, j'ai pu appliquer une démarche méthodologique basée sur la théorie de l'information et la théorie des systèmes dynamiques. Grâce à la représentation symbolique et sur la base de signaux MEG qui constituent une mesure de l'activité cérébrale, j'ai réussi mettre en exergue le lien entre dynamique cérébrale et complexité linguistique en utilisant comme métrique l'entropie. En effet, dans le contexte où on l'a utilisé, la représentation symbolique permet la majeure partie du temps de caractériser un système dynamique continu (le cerveau) comme une séquence de symboles correspondants à des états discrets. Cela permet de rendre compte de la complexité intrinsèque de la dynamique cérébrale et de la quantité d'information contenue dans celle-ci grâce à l'entropie. Le taux d'entropie comme résultat final de l'algorithme mis en place durant le stage semble être un bon indicateur de la complexité linguistique et a permis de discriminer des conditions expérimentales à partir des mesures de l'activité cérébrale.

Comme perspectives futures, il serait intéressant d'appliquer une segmentation temporelle différente et s'intéresser à la variation du taux d'entropie au cours d'une phrase. Aussi, il faudrait appliquer l'algorithme développé au cours du stage sur l'ensemble des 204 sujets de la base de données MOUS. En effet, de par la variabilité interindividuelle, il serait intéressant de mener l'étude sur une plus grande quantité de sujets différents. Cela permettrait de réduire encore plus le rapport signal sur bruit et obtenir des résultats plus robustes. On notera aussi qu'il existe des méthodes de représentation symbolique plus complexes qui permettent de rendre compte plus en détails de la dynamique cérébrale. En effet, la décomposition atomique [21] est une méthode de représentation symbolique qui consiste à extraire des atomes spatio-temporels du signal. Cette manière plus fine de représenter symboliquement un système dynamique pourrait être une des pistes de développement futur du projet.

Bibliographie

- [1] Christofer M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.
- [2] Jean-Luc Blanc. *Transmission de l'information et complexité des activités de populations neuronales*. PhD thesis, Laboratoire de Neurosciences Intégrative et adaptative, 2012.
- [3] Paul Landais Bruno Riou. Principes des tests d'hypothèse en statistique : alpha, beta et p. *Annales Françaises d'Anesthésie et de Réanimation*, 17, 1998.
- [4] F.H. Lopes da Silva, A. van Rotterdam, P. Barts, E. van Heusden, and W. Burr. Models of neuronal populations : The basic mechanisms of rhythmicity. *Progress in Brain Research*, 2005.
- [5] Anne-Sophie Dubarry. *Linking neurophysiological data to cognitive functions : methodological developments and applications*. PhD thesis, Laboratoire de Psychologie Cognitive, UMR7290, 2016.
- [6] Alexandre Gramfort, Martin Luessi, Eric Larson, Denis A. Engemann, Daniel Strohmeier, Christian Brodbeck, Roman Goj, Mainak Jas, Teon Brooks, Lauri Parkonen, and Matti S. Hämäläinen. MEG and EEG data analysis with MNE-Python. *Frontiers in Neuroscience*, 7(267) :1–13, 2013.
- [7] Peter C. Hansen, Morten L. Kringelbach, and Riitta Salmelin. *MEG : An introduction to methods*. Oxford, 2010.
- [8] Yoshito Hirata and José María Amigó. A review of symbolic dynamics and symbolic reconstruction of dynamical systems. *Chaos*, 2023.
- [9] Matti Hämäläinen, Riitta Hari, Risto J. Ilmoniemi, Jukka Knuutila, and Olli V. Lounasmaa. Magnetoencephalography—theory, instrumentation, and applications to noninvasive studies of the working human brain. *Reviews of Modern Physics*, 1993.
- [10] Alan Jasanoff. Bloodless fmri. *Trends in Neurosciences*, 2007.
- [11] Andreï KOLMOGOROV. *Théorie générale des systèmes dynamiques de la mécanique classique*, volume 1. Séminaire Janet. Mécanique analytique et mécanique céleste, 1958.
- [12] Abraham Lempel and Jacob Ziv. Transactions on information theory. *IEEE*, 22, 1976.
- [13] Abraham Lempel and Jacob Ziv. Transactions on information theory. *IEEE*, 24, 1978.
- [14] Annick Lesne, Jean-Luc Blanc, and Laurent Pezard. Entropy estimation of very short symbolic sequences. *Physical Review, E* 79, 046208 :1–5, 2009.
- [15] Byoung-Kyong Min, Matthew J. Marzelli, and Seung-Schik Yoo. Neuroimaging-based approaches in the brain-computer interface. *Trends Biotechnol*, 2010.

- [16] Guiomar Niso, Laurens R. Krol, Etienne Combrisson, A. Sophie Dubarry, Madison A. Elliott, Clément François, Yseult Héjja-Brichard, Sophie K. Herbst, Karim Jerbi, Vanja Kovic, Katia Lehongre, Steven J.Luck, Manuel Mercier, John C.Mosher, Yuri G.Pavlov, Aina Puce, ANtonio Schettino, Daniele Schön, Walter Sinnott-Armstrong, Bertille Somon, and Maximilien Chaumon. Good scientific practice in eeg and meg research : Progress and perspectives. *NeuroImage*, 2022.
- [17] Jan-Mathijs Schoffelen, Robert Oostenveld, Nietzsche H.L.Lam, Julia Uddén, Annika Hultén, and Peter Hagoort. A 204-subject multimodal neuroimaging dataset to study language processing. *Scientific data*, 2019.
- [18] Gideon E. Schwarz. *L'estimation de la dimension d'un modèle*, volume 6. Annals of Statistics, 1978.
- [19] Claude E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27, 1948.
- [20] Joy A. Thomas and Thomas M. Cover. *Elements of Information theory*. Wiley Inter-science, 1991.
- [21] Thomas Dupré La Tour, Thomas Moreau, Mainak Jas, and Alexandre Gramfort. Multivariate convolutional sparse coding for electromagnetic brain signals. *arXiv*, 2018.

Annexes

Séquence symbolique

Cette partie a été adaptée de [8].

Partitions

Soit $(\Omega, \mathcal{B}, \mu)$ un espace mesuré, i.e Ω est un ensemble non-vide appelé l'espace des états, \mathcal{B} est une tribu de sous-ensembles de Ω , et μ est une mesure positive sur l'espace mesurable (Ω, \mathcal{B}) . Une partition de Ω est une famille disjointe d'éléments de \mathcal{B} , non-vides et dont l'union est Ω . On suppose que $\mu(\Omega) \geq \infty$, on peut alors dire que $(\Omega, \mathcal{B}, \mu)$ est un espace de probabilité i.e. $\mu(\Omega) = 1$. On considérera par la suite essentiellement des partitions finies, i.e. une partition avec un nombre fini d'éléments qui ont une mesure positive. La Figure 5 illustre un exemple d'une partition finie.

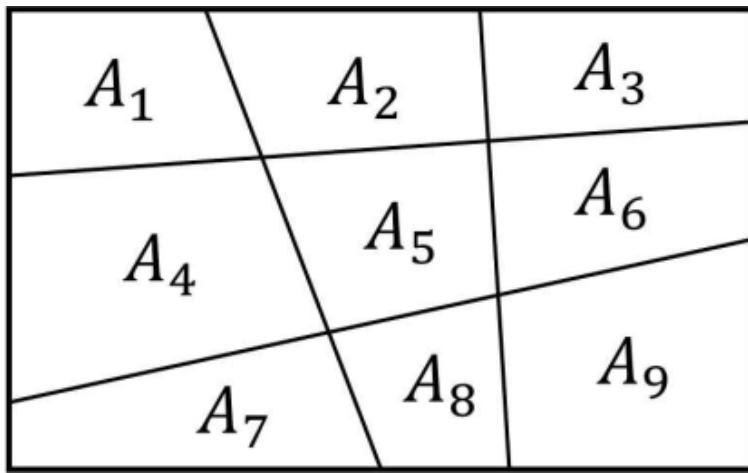


FIGURE 5 – Un domaine rectangulaire du plan est divisé en 9 sous-ensembles disjoints A_1, A_2, \dots, A_9 (extrait de [8])

Les partitions seront représentées par des lettres grecques minuscules. On introduit la partition de points distincts qui nous servira dans la partie sur les partitions génératrices :

$$\epsilon = \{\{x\} : x \in \Omega\} \quad (1)$$

Soit $\alpha = \{A_1, \dots, A_{|\alpha|}\} = \{A_a : 1 \geq a \geq |\alpha|\}$ et $\beta = \{B_1, \dots, B_{|\beta|}\} = \{B_a : 1 \geq a \geq |\beta|\}$ des partitions de Ω , où $|\alpha|$ désigne le cardinal de l'ensemble α .

On définit la partition $\alpha \vee \beta = \{A_a \cap B_b : 1 \geq a \geq |\alpha|, 1 \geq b \geq |\beta|\}$ comme le raffinement de α et β . La raffinement de plus de deux partitions est défini de manière récursive.

L'entropie de α est définie par

$$H_\mu(\alpha) = - \sum_{a=1}^{|\alpha|} \mu(A_a) \log_\mu(A_a) \quad (2)$$

où le logarithme peut être en base 2 ou e.

Si $X : \Omega \rightarrow \{1, \dots, |\alpha|\}$ est une variable aléatoire définie par $X(\omega) = a$ si et seulement si $\omega \in A_a$, alors $H_\mu(\alpha)$ coïncide avec l'entropie de Shannon de X ,

$$H(X) = - \sum_{a=1}^{|\alpha|} Pr\{X = a\} \log Pr\{X = a\} \quad (3)$$

puisque la probabilité que $X = a$ est $Pr\{X = a\} = \mu\{X^{-1}(a)\} = \mu(A_a)$.

Comme vu précédemment, $H(X)$ mesure l'incertitude des valeurs prises par la variable aléatoire X .

Système dynamique et entropie

Une fois que les espaces de probabilité et les partitions ont été introduits, l'ingrédient restant est le concept de dynamique ou de passage du temps. Considérons un espace probabilisé $(\Omega, \mathcal{B}, \mu)$ comme défini précédemment ainsi qu'une application mesurable $f : \Omega \rightarrow \Omega$, i.e., $f^{-1}(B) \in \mathcal{B}$ pour tout $B \in \mathcal{B}$. Le lien entre l'espace de probabilité et l'application mesurable réside dans le fait que f préserve la mesure, i.e. $\mu(f^{-1}(B)) = \mu(B)$ pour tout $B \in \mathcal{B}$. Alternativement, on dit que f est μ -invariant ou bien que la mesure μ est f -invariante. En particulier, si on prend une partition $\alpha = \{A_a : 1 \geq a \geq |\alpha|\}$ de Ω , alors $f^{-1}(\alpha) = \{f^{-1}(A_a) : 1 \geq a \geq |\alpha|\}$ est aussi une partition de Ω . Si, de plus, f est une application préservant la mesure, un à un, et que f^{-1} préserve également la mesure, alors on dit que f est une application inversible préservant la mesure.

En s'appuyant sur les définitions précédentes, on appelle $(\Omega, \mathcal{B}, \mu, f)$ un système dynamique (préservant la mesure). Les dynamiques de ce système sont générées par les itérations $f^{n+1}(x) = f(f^n(x))$ au pas de temps discrets $n \in \mathbb{T}$. La séquence $(f^n(x))_{n \in \mathbb{T}}$ est appelée orbite ou trajectoire avant (resp. complète) de x si $\mathbb{T} = \mathbb{N}_0$ (resp. $\mathbb{T} = \mathbb{Z}$) ; x est parfois appelé la condition initiale.

A ce stade et pour la suite, il nous faut introduire une définition. Il est dit que $(\Omega_1, \mathcal{B}_1, \mu_1, f_1)$ et $(\Omega_2, \mathcal{B}_2, \mu_2, f_2)$ sont isomorphes s'il existe $M_i \in \mathcal{B}_i$ ($i = 1, 2$) avec $\mu(M_i) = 1$ et $f(M_i) \subset M_i$, et qu'il existe une application inversible préservant la mesure $\Phi : M_1 \rightarrow M_2$ avec $(\Phi \circ f_1)(x) = (f_2 \circ \Phi)(x)$, $\forall x \in M - 1$. L'application Φ est appelée un isomorphisme.

Si l'on considère le système dynamique à mesure preservée $(\Omega, \mathcal{B}, \mu, f)$ et une partition $\alpha = \{A_a : 1 \geq a \geq |\alpha|\}$ de $(\Omega, \mathcal{B}, \mu)$, alors

$$h_\mu = \lim_{n \rightarrow \infty} \frac{1}{n} H_\mu \left(\bigvee_{i=0}^{n-1} f^{-i} \alpha \right) \quad (4)$$

Est appelée l'entropie de f par rapport à la partition α . Avec l'équation (5.7), on obtient

$$H_\mu\left(\bigvee_{i=0}^{n-1} f^{-i}\alpha\right) = \sum_{a_0, a_1, \dots, a_{n-1}} \mu(A_{a_0} \cap f^{-1}A_{a_1} \cap \dots \cap f^{-n+1}A_{a_{n-1}}) \log \mu(A_{a_0} \cap f^{-1}A_{a_1} \cap \dots \cap f^{-n+1}A_{a_{n-1}}) \quad (5)$$

On peut à présent introduire l'entropie de Kolmogorov-Sinai de l'application f avec le même système dynamique à mesure preservée que précédemment $(\Omega, \mathcal{B}, \mu, f)$. Celle-ci est définie de la manière suivant

$$h_\mu(f) = \sup h_\mu(f, \alpha) \quad (6)$$

où le supremum est pris sur toutes les partitions finies de $(\Omega, \mathcal{B}, \mu)$. On verra par la suite que le supremum est atteint pour des partitions génératrices dans l'espace des phases (états).

Dynamique symbolique par rapport à une partition

On a vu que tout processus aléatoire stationnaire à temps discret sur un espace de probabilité correspond à un système dynamique préservant la mesure.

En effet, soit $Z = \{Z_n\}_{n \in \mathbb{T}}$ un processus aléatoire (i.e. stochastique) sur un espace de probabilité $(\Omega, \mathcal{B}, \mu)$ avec comme états $\{1, \dots, k\}$ et une distribution jointe de probabilité $p(z_{n_1}, \dots, z_{n_r})$ où $r \geq 1$ et $z_{n_1}, \dots, z_{n_r} \in \{1, \dots, k\}$. On note $\{1, \dots, k\}^{\mathbb{T}}$ l'ensemble de toutes les séquences ayant comme symboles $\{1, \dots, k\}$. On définit l'application de codage mesurable $\Phi : \Omega \longrightarrow \{1, \dots, k\}^{\mathbb{T}}$ tel que $\Phi(\omega) = (Z_n(\omega))_{n \in \mathbb{T}}$ et \mathcal{C} la tribu produit générée par les ensembles cylindriques, i.e. les ensembles des séquences à valeurs dans $\{1, \dots, k\}^{\mathbb{T}}$ avec un nombre fini de composantes fixées. En outre, on dote l'espace mesurable $(\{1, \dots, k\}^{\mathbb{T}}, \mathcal{C})$ d'une mesure transportée $m = \mu \circ \Phi^{-1}$, i.e.,

$$m(C) = \mu(\Phi^{-1}C), C \in \mathcal{C} \quad (7)$$

de sorte à ce que Φ préserve la mesure.

L'opérateur de décalage $\sigma : (z_n)_{n \in \mathbb{T}} \longrightarrow (z_{n+1})_{n \in \mathbb{T}}$ sur l'espace de probabilité $(\{1, \dots, k\}^{\mathbb{T}}, \mathcal{C}, m)$ est alors m -invariant de par la stationnarité de Z . On a alors $(\{1, \dots, k\}^{\mathbb{T}}, \mathcal{C}, m, \sigma)$ qui constitue un système dynamique préservant la mesure que l'on appelle système dynamique discret ou support de Z , $Supp(Z)$. Les états du système dynamique $Supp(Z)$ sont les trajectoires ou réalisations $(z_n)_{n \in \mathbb{T}}$ du processus aléatoire Z . Si $n = 0$ correspond au temps actuel, les valeurs de Z sont mis à jour grâce à l'application de décalage σ .

Comme vu dans les parties précédentes, le taux d'entropie de Shannon du processus stochastique stationnaire $Z = \{Z_n\}_{n \in \mathbb{T}}$, avec comme états $(\{1, \dots, k\})$, est défini par

$$h(Z) = - \lim_{n \rightarrow \infty} \sum_{z_1, \dots, z_n} Pr\{Z_1 = z_1, \dots, Z_n = z_n\} \log Pr\{Z_1 = z_1, \dots, Z_n = z_n\} \quad (8)$$

Cette limite converge car le processus stochastique Z est stationnaire.

Nous venons de voir que tout processus aléatoire stationnaire peut être associé à un système dynamique appelé son support. Cette association est rendue possible par l'application

de codage Φ . Il se trouve que ce qui nous intéresse dans notre étude est le corrolaire : tout système dynamique peut être associé à un processus stochastique stationnaire avec un nombre d'états fini appelé dynamique symbolique. Cette seconde association est rendue possible par les partitions, c'est ce que l'on va expliciter par la suite.

On reprend le système dynamique à mesure préservée $(\Omega, \mathcal{B}, \mu, f)$ et une partition $\alpha = \{A_a : 1 \leq a \leq |\alpha|\}$ de $(\Omega, \mathcal{B}, \mu)$ introduit.e.s. On définit, pour tout $n \in \mathbb{T}$, l'application mesurable $Z_n^\alpha : \Omega \longrightarrow \{1, \dots, |\alpha|\}$ telle que

$$Z_n^\alpha(x) = a_n \iff f^n(x) \in A_{a_n} \quad (9)$$

Alors $\mathbf{Z}^\alpha = \{Z_n^\alpha\}_{n \in \mathbb{T}}$ est un processus stochastique stationnaire. En effet, la distribution de probabilité jointe de \mathbf{Z}^α nous est donnée par le biais de la mesure de probabilité μ

$$\Pr\{Z_0^\alpha = a_0, \dots, Z_n^\alpha = a_n\} = \mu\{x \in \Omega : x \in A_{a_0}, f(x) \in A_{a_1}, \dots, f^n(x) \in A_{a_n}\} = \mu\{A_{a_0} \cap f^{-1}A_{a_1} \cap \dots \cap f^{-n+1}A_{a_n}\} \quad (10)$$

La stationnarité de \mathbf{Z} est triviale de par le fait que μ soit f -invariant.

La séquence $(a_n)_{n \in \mathbb{T}}$ constitue la trajectoire symbolique de x par rapport à la partition α , $a_n \in \{1, \dots, |\alpha|\}$ étant le n ième symbole de la trajectoire.

$\mathbf{Z}^\alpha = \{Z_n^\alpha\}_{n \in \mathbb{T}}$ défini dans l'équation (3.19) est la dynamique symbolique de f par rapport à la partition α . On en déduit de l'équation (3.20) que

$$h(\mathbf{Z}^\alpha) = h_\mu(f, \alpha) \quad (11)$$

C'est l'entropie de Kolmogorov-Sinai de l'application f préservant la mesure μ par rapport à la partition α qui est égale à l'entropie de sa dynamique symbolique par rapport à la même partition. Cela prouve que $h_\mu(f, \alpha)$ existe toujours car \mathbf{Z}^α est un processus stochastique stationnaire.

Partitions génératrices

Le point clef de la représentation symbolique est l'existence de partitions α de $(\Omega, \mathcal{B}, \mu)$ telles que la correspondance entre les trajectoires $(f^n(x))_{n \in \mathbb{T}}$ d'une application qui préserve la mesure $f : \Omega \longrightarrow \Omega$ et leurs représentations symboliques $\Phi^\alpha = (a_n)_{n \in \mathbb{T}}$ soit exacte.

Une partition α de $(\Omega, \mathcal{B}, \mu)$ est appelée partition génératrice unilatérale pour une application préservant la mesure $f : \Omega \longrightarrow \Omega$ si $\bigvee_{i=0}^{\infty} f^{-i}\alpha = \epsilon$. En outre, si de plus f est inversible et $\bigvee_{i \in \mathbb{Z}} f^{-i}\alpha = \epsilon$ alors α est une partition génératrice bilatérale pour f .

Si α est une partition génératrice pour f , alors $\text{Supp}(\mathbf{Z}^\alpha)$ est isomorphe à $(\Omega, \mathcal{B}, \mu, f)$. Ce théorème implique la correspondance un à un des trajectoires des séries temporelles avec leurs représentations symboliques grâce aux partitions génératrices.

On considère toujours le même système dynamique à mesure préservée $(\Omega, \mathcal{B}, \mu, f)$. Le théorème de Kolmogorov-Sinai nous dit que si α est une partition génératrice pour f , alors on a

$$h_\mu(f) = h_\mu(f, \alpha) \quad (12)$$

Enfin, les partitions génératrices nous permettent de faire le lien entre le taux d'entropie de Shannon (5.19) et l'entropie de Kolmogorov-Sinai via les dynamiques symboliques. En effet, si α est une partition génératrice pour f , alors on obtient à partir de l'équation (5.22) et en vertue du théorème (5.23)

$$h(\mathbf{X}^\alpha) = h_\mu(f) \quad (13)$$

De plus, de par le fait que $(\Omega, \mathcal{B}, \mu, f)$ et $Supp(\mathbf{Z}^\alpha) = (\{1, \dots, |\alpha|\}^{\mathbb{T}}, \mathcal{C}, m, \sigma)$ soient isomorphes, l'équation (5.24) devient

$$h(\mathbf{X}^\alpha) = h_\mu(f) = h_m(\sigma) \quad (14)$$

scikits-symbolic

Release 0.0

Laurent Pezard, Jean-Luc Blanc and others

Oct 09, 2023

CHAPTER
TWO

SYMBOLIC SEQUENCES

A symbolic sequence is a list of symbols taken from a finite alphabet of length k

Internally they are encoded according to integers from 0 to $k - 1$

Alphabet should be bidirectional dictionary...

This is the doc!

class `sequence.Alphabet(nsymb)`

The set of states or symbols that can be visited for a sequence or Markov process realization.

Tests on State and Alphabet

```
>>> state1 = State('One')
>>> state1
State(- | One)
```

An integer representation of a state is only attributed once the state is inserted in an alphabet.

```
>>> state2 = State('Two')
>>> state3 = State('Three')
```

Alphabets can be created with a list of states:

```
>>> alpha = Alphabet([state1, state2, state3])
>>> alpha
Alphabet[State(0 | One), State(1 | Two), State(2 | Three)]
>>> print(alpha)
Alphabet[State(0 | One), State(1 | Two), State(2 | Three)]
>>> len(alpha)
3
```

Alphabets can also be created using only the length as argument.

```
>>> beta = Alphabet(3)
>>> beta
Alphabet[State(0 | 0), State(1 | 1), State(2 | 2)]
>>> alpha[0]
State(0 | One)
```

States can be changed but the integer value is kept:

```
>>> alpha[1] = State('Deux')
>>> alpha
Alphabet[State(0 | One), State(1 | Deux), State(2 | Three)]
```

Alphabet's states can be changed using a dictionary representation.

```
>>> alpha.rename({0 : 'Uno', 2 : 'Tre'})
>>> alpha
Alphabet[State(0 | Uno), State(1 | Deux), State(2 | Tre)]
>>> beta.rename({0 : 'Uno', 1 : 'Deux', 2 : 'Tre'})
>>> alpha == beta
True
```

sequence.DEFAULT_DTYPE

alias of numpy.uint16

class sequence.Sequence(*symbols*, *alphabet*, *dtype*=<class 'numpy.uint16'>, *check*=True)

Defines a symbolic sequence coded using integers in $\{0, k - 1\}$ and their methods.

Test for the Sequence class and its methods

```
>>> a = [1,0,0,0,1,0,1,0,1,1,0,1]
>>> b = [0,1,0,0,1,1,1,0,0,1,0]
>>> A = Alphabet(['a', 'b'])
>>> s1 = Sequence(a, A)
>>> s2 = Sequence(b, A)
>>> s1
Sequence: [1 0 0 0 1 0 1 0 1 1 0 1]
Alphabet[State(0 | a), State(1 | b)]
N = 12 ; k = 2
>>> print(s1)
[1 0 0 0 1 0 1 0 1 1 0 1]
```

The length of the alphabet is a property named *k* of the sequence.

```
>>> s1.k
2
>>> s1.alphabet
Alphabet[State(0 | a), State(1 | b)]
```

Slices return Sequence object

```
>>> s1[0]
Sequence: [1]
Alphabet[State(0 | a), State(1 | b)]
N = 1 ; k = 2
>>> s1[4:8]
Sequence: [1 0 1 0]
Alphabet[State(0 | a), State(1 | b)]
N = 4 ; k = 2
>>> s1[s1.ivals < 1]
Sequence: [0 0 0 0 0 0]
Alphabet[State(0 | a), State(1 | b)]
N = 6 ; k = 2
```

We can access to the lenght of a sequence

```
>>> len(s1)
12
```

Concatenation of two sequences return Sequence object if the alphabet of the two sequences are the same

```
>>> s1 + s2
Sequence: [1 0 0 0 1 0 1 0 1 1 0 1 0 1 0 0 1 1 1 0 0 1 0]
Alphabet[State(0 | a), State(1 | b)]
N = 24 ; k = 2
```

Comparison between two sequences with the same length returns a Sequence object with the results of the comparison

```
>>> s1 == s2
Sequence: [0 0 1 1 1 0 1 0 0 0 0]
Alphabet[State(0 | False), State(1 | True)]
N = 12 ; k = 2
>>> s1 > s2
Sequence: [1 0 0 0 0 0 0 1 1 0 1]
Alphabet[State(0 | False), State(1 | True)]
N = 12 ; k = 2
>>> s1 >= s2
Sequence: [1 0 1 1 1 0 1 0 1 1 0 1]
Alphabet[State(0 | False), State(1 | True)]
N = 12 ; k = 2
>>> s1 < s2
Sequence: [0 1 0 0 0 1 0 1 0 0 1 0]
Alphabet[State(0 | False), State(1 | True)]
N = 12 ; k = 2
>>> s1 <= s2
Sequence: [0 1 1 1 1 1 1 0 0 1 0]
Alphabet[State(0 | False), State(1 | True)]
N = 12 ; k = 2
>>> s1 != s2
Sequence: [1 1 0 0 0 1 0 1 1 1 1 1]
Alphabet[State(0 | False), State(1 | True)]
N = 12 ; k = 2
```

With binary sequences, logical operators return e Sequence object with the result

```
>>> s1 & s2
Sequence: [0 0 0 0 1 0 1 0 0 0 0]
Alphabet[State(0 | False), State(1 | True)]
N = 12 ; k = 2
>>> s1 ^ s2
Sequence: [1 1 0 0 0 1 0 1 1 1 1 1]
Alphabet[State(0 | False), State(1 | True)]
N = 12 ; k = 2
>>> s1 | s2
Sequence: [1 1 0 0 1 1 1 1 1 1 1 1]
Alphabet[State(0 | False), State(1 | True)]
N = 12 ; k = 2
```

It is possible to transform a sequence in place

```
>>> s1.roll(2)
>>> s1
Sequence: [0 1 1 0 0 0 1 0 1 0 1 1]
Alphabet[State(0 | a), State(1 | b)]
N = 12 ; k = 2
>>> s1.reverse()
>>> s1
Sequence: [1 1 0 1 0 1 0 0 0 1 1 0]
Alphabet[State(0 | a), State(1 | b)]
N = 12 ; k = 2
>>> s1.reduce()
>>> s1
Sequence: [1 0 1 0 1 0 1 0]
Alphabet[State(0 | a), State(1 | b)]
N = 8 ; k = 2
```

And we can do the same creating a new sequence modified from a base sequence

```
>>> s3 = roll(s2,12)
>>> s3
Sequence: [0 1 0 0 1 1 1 1 0 0 1 0]
Alphabet[State(0 | a), State(1 | b)]
N = 12 ; k = 2
>>> s3 = reduce(s2)
>>> s3
Sequence: [0 1 0 1 0 1 0]
Alphabet[State(0 | a), State(1 | b)]
N = 7 ; k = 2
>>> s3 = reverse(s2)
>>> s3
Sequence: [0 1 0 0 1 1 1 1 0 0 1 0]
Alphabet[State(0 | a), State(1 | b)]
N = 12 ; k = 2
```

This functions provide us information about a sequence

```
>>> s1.count()
array([4, 4])
>>> s1.frequency()
array([0.5, 0.5])
>>> issequence(s2)
True
```

From a list of sequences of the same length but that can have different alphabets, we can recode them creating a new sequence with new symbols and a new alphabet

```
>>> B = Alphabet(['aa','bb','cc'])
>>> s4 = Sequence([2,2,1,0,2,0,0,1,2,1,0,0],B)
>>> s4
Sequence: [2 2 1 0 2 0 0 1 2 1 0 0]
Alphabet[State(0 | aa), State(1 | bb), State(2 | cc)]
N = 12 ; k = 3
```

(continues on next page)

(continued from previous page)

```
>>> s2
Sequence: [0 1 0 0 1 1 1 1 0 0 1 0]
Alphabet[State(0 | a), State(1 | b)]
N = 12 ; k = 2
>>> s3 = recode([s2,s4], new_alphabet=True, names=['seq2','seq4'])
>>> s3
Sequence: [2 5 1 0 5 3 3 4 2 1 3 0]
Alphabet[State(0 | seq2_a+seq4_aa), State(1 | seq2_a+seq4_bb), State(2 | seq2_
˓→a+seq4_cc), State(3 | seq2_b+seq4_aa), State(4 | seq2_b+seq4_bb), State(5 | seq2_
˓→b+seq4_cc)]
N = 12 ; k = 6
```

count(ival=None)Counts the number of each symbol in $\{0, k - 1\}$ if code is None or the number of the code symbol**Returns** a numpy.ndarray of integers**frequency()**Returns the probability of each symbol in $\{0, k - 1\}$ **Returns** a numpy.ndarray of floats**reduce()**Delete the repetitions of symbols in a sequence *in place***reverse()**Reverse the sequence *in place***See also:**

numpy.flipud function

roll(step)Roll the sequence *in place***See also:**

numpy.roll function

shuffle()Shuffle the order of the sequence *in place*.**See also:**

numpy.random.shuffle function

class sequence.State(strval)

A state (or symbol) is used to define the state of the system at time \$t\$.

It has two properties:

- *strval*: its name which can be accessed, changed but not deleted
- *ival*: its associated integer value which can be accessed but neither changed nor deleted

setter and deleter raise exception for explicit behavior.

sequence.issequence(obj)

Returns True if x is a symbolic sequence

sequence.recode(lseq, new_alphabet=False, sep='+', names=None)

Recodes a list of sequences with (possibly) different alphabets but with the same length (This is an error to pass Sequences with different length.) A new dictionary is built for the new sequence.

Parameters `lseq` – a list of Sequences

Raises `LengthError`: when the length of the Sequences are different.

Returns a Sequence

`sequence.reduce(seq)`

Returns a reduced sequence (ie only keep the transitions)

`sequence.reverse(seq)`

Reverse the sequence

`sequence.roll(seq, step)`

Roll the sequence

`sequence.shuffle(seq)`

Shuffle the sequence

`sequence.transform(seq, correspondance, new_alphabet=None)`

Transforms the initial sequence according to the correspondence iterable

`sequence.words(seq, wlen, new_alphabet=False)`

Returns a sequence encoded according to the m-words in seq

Todo: Write the doc of “words”

DISCRETIZERS

discretize.partition(*arr, method='histogram', nbin=10, d=None*)

Discretize a continuous series according to method.

Methods are described in Hlavackova-Schindler et al. Physics Reports 441 (2007) 1–46 pages 14–19

method = ‘histogram’ simple histogram method with equidistant binning

method = ‘marginal_equipartition’ marginal equipartition ie does its best to let equal number of observation in each bin.

Parameters

- **x** – a continuous series
- **method** – a string in [“histogram”, “marginal_equipartition”]
- **nbin** – the number of bins ie the length of the alphabet
- **d** – a dictionary

Raises `NotImplementedError` if method is not in the list above.

Returns A symbolic Sequence

Todo: To be completed with the other methods described in Hlavackova-Schindler (2007)

Hint: look at R implementation of histogram function.

Tests and examples of the fonctionnement of the module

```
>>> x = np.linspace(0,10,11)
>>> x
array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.])
>>> seq = partition(x, method='histogram', nbin=6)
>>> seq
Sequence: [0 0 1 1 2 3 3 4 4 5 5]
Alphabet[State(0 | 0), State(1 | 1), State(2 | 2), State(3 | 3), State(4 | 4),
          ↵State(5 | 5)]
N = 11 ; k = 6
>>> seq = partition(x, method='marginal_equipartition',nbin=6)
>>> seq
Sequence: [0 0 1 1 2 2 3 3 4 4 5]
```

(continues on next page)

(continued from previous page)

```
Alphabet[State(0 | 0), State(1 | 1), State(2 | 2), State(3 | 3), State(4 | 4),  
         ↳ State(5 | 5)]  
N = 11 ; k = 6
```

discretize.phase_cluster(*data, nb_symb, target_dim=10*)

This function provides the symbolic dynamic of a multivariate data It is based on the clusterisation of the “phase space” of the channels of MEG temporal signal

Parameters

- **data** – The input matrix, the lines are the channels and the columns are the time, must be an array
- **nb_symb** – The number of bins used for the clusterisation i.e. the number of symbols of the symbolic sequences that will be created

:param nb_vp : The number of eigen vectors that we want to conserve to project our data on it

discretize.subdivision(*data, iter_max*)

Ulam method Adaptive subdivision technique based on:

Set oriented numerical methods for dynamical systems Dellnitz M. and Junge O. Handbook of dynamical systems vol. 2 p. 221-264 Elsevier 2002.

and

Numerical approximation of random attractors Keller H. and Ochs G. in “Stochastic dynamics” Crauel H. and Gundlach M. Eds Springer 1999. p. 93-115

Input

x: numpy array kmax: integer, maximum number of boxes

discretize.symbolize(*arr, bins, d=None*)

Todo: is this funtion symbolize useful? (duplicate numpy.digitize?) Qu'est-ce que nbin ? Peut-être d ?

```
>>>
```

ALGORITHMIC COMPLEXITY PROCEDURES

Algorithms and procedures related to algorithmic approach of complexity

algorithmic.contains_sublist(lst, sublst)

Check whether a sublist appears in a list

found at: <http://stackoverflow.com/questions/3313590/... ... check-for-presence-of-a-sublist-in-python>

algorithmic.lempel_ziv(seq, parsing='lz76', norm=False, nbsur=None)

Returns the Lempel-Ziv normalized complexity using either lz76 or lz77 parsing.

Parameters

- **seq** – a Sequence object
- **parsing** – a string in [“lz76”, “lz77”]
- **norm** – a boolean (should the complexity be normalized?)
- **ns** – the number of surrogate data used in the normalization.

Raise `NotImplementedError` if *parsing* is not in the list above.

Returns a float (the Lempel-Ziv complexity)

Example :

```
>>> np.random.seed(9)
>>> a = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> A = S.Alphabet(['a','b'])
>>> seq = S.Sequence(a,A)
>>> lempel_ziv(seq)
0.6
```

algorithmic.lz76(arr, summary=False)

Returns Lempel-Ziv complexity according to LZ76 parsing.

Parameters

- **arr** – an array of integers
- **summary** – A boolean (should the dictionary be returned)

Returns either an integer (*summary=False*) or a tuple (*summary=True*) with an integer and a list of strings.

Example :

```
>>> np.random.seed(9)
>>> a = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> lz76(a)
92
```

algorithmic.lz77(*arr*, *summary=False*)

Returns Lempel-Ziv complexity according to LZ77 parsing.

Parameters

- **arr** – an array of integers
- **summary** – A boolean (should the dictionary be returned)

Returns either an integer (*summary=False*) or a tuple (*summary=True*) with an integer and a list of strings.

Example :

```
>>> np.random.seed(9)
>>> a = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> lz77(a)
158
```

STOCHASTIC

Defines stochastic matrices

stochastic.conditional_matrix(seq1, seq2)

Returns the conditional matrix ie $P(s_1=j | x_2=i)$.

This is estimated using the maximum likelihood estimator.

Parameters

- **seq1** – a symbolic Sequence object
- **seq2** – a symbolic Sequence object

Returns A numpy.matrix of floats

..todo:

check the doc and implementation of conditional_matrix

NB: lines should sum to one (one should go somewhere) see markov_sequence in generate.py ie `np.sum(matrix, axis=1) == [[1]...[1]]`

Example :

```
>>> np.random.seed(9)
>>> a = np.random.choice([0,1], 1000, replace=True, p=[0.7,0.3])
>>> np.random.seed(6)
>>> b = np.random.choice([0,1], 1000, replace=True, p=[0.7,0.3])
>>> A = S.Alphabet(['a', 'b'])
>>> seq1 = S.Sequence(a,A)
>>> seq2 = S.Sequence(b,A)
>>> conditional_matrix(seq1, seq2)
[[0.71947674 0.28052326]
 [0.69551282 0.30448718]] [688 312]
(array([], dtype=int64),)
matrix([[0.71947674, 0.28052326],
 [0.69551282, 0.30448718]])
```

stochastic.influence_matrix(seq1, seq2, time=1)

Returns the influence matrix ie $P(x_1(T+t)=j | x_2(T)=i)$.

This is estimated using the maximum likelihood estimator.

Parameters

- **seq1** – a symbolic Sequence object

- **seq2** – a symbolic Sequence object

Returns A numpy.matrix of floats

Example :

```
>>> np.random.seed(9)
>>> a = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> np.random.seed(6)
>>> b = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> A = S.Alphabet(['a','b'])
>>> seq1 = S.Sequence(a,A)
>>> seq2 = S.Sequence(b,A)
>>> influence_matrix(seq1, seq2)
[[0.70887918 0.29112082]
 [0.71794872 0.28205128]] [687 312]
(array([], dtype=int64),)
matrix([[0.70887918, 0.29112082],
       [0.71794872, 0.28205128]])
```

stochastic.transition_matrix(*seq*, *time*=1)

Returns the transition matrix.

This is estimated using the maximum likelihood estimator.

Parameters **seq** – a symbolic Sequence object

Returns A numpy.matrix of floats

Example :

```
>>> np.random.seed(9)
>>> a = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> A = S.Alphabet(['a','b'])
>>> seq = S.Sequence(a,A)
>>> transition_matrix(seq)
[[0.70224719 0.29775281]
 [0.73519164 0.26480836]] [712 287]
(array([], dtype=int64),)
matrix([[0.70224719, 0.29775281],
       [0.73519164, 0.26480836]])
```

CHAPTER
SIX

INFORMATION

information.H(*seq*)

Returns Shannon's (metric) entropy of sequence

Parameters **seq** – a symbolic Sequence object

Returns a float

Example :

On fixe la seed pour pouvoir contrôler la génération de vecteur “aléatoires”

```
>>> np.random.seed(9)
>>> a = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> A = S.Alphabet(['a','b'])
>>> seq = S.Sequence(a,A)
>>> metric_entropy(seq)
0.6003511877776578
```

information.R(*seq, coef*)

Returns the Rényi entropy

..todo:

```
Make the doc of renyi_entropy!
```

Example :

```
>>> np.random.seed(9)
>>> a = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> A = S.Alphabet(['a','b'])
>>> seq = S.Sequence(a,A)
>>> renyi_entropy(seq, 0.9)
0.6088567303148161
```

information.T(*seq*)

Returns the topological entropy

Parameters **seq** – a symbolic Sequence object

Returns a float

Example :

```
>>> np.random.seed(9)
>>> a = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
```

(continues on next page)

(continued from previous page)

```
>>> A = S.Alphabet(['a', 'b'])
>>> seq = S.Sequence(a,A)
>>> topological_entropy(seq)
0.6931471805599453
```

information.block_entropy(*seq, wlen*)

Returns the block entropy

Parameters

- **seq** – a symbolic Sequence object
- **n** – the block length
- **method** – a string in [“metric”, “shannon”, “topological”, “renyi”, “all”]

Raises ValueError if $n < 0$

NotImplementedError if the method is not in the list above.

Returns either the value of the demanded entropy or all their value in a tuple Tn, Hn, hav

Example :

```
>>> np.random.seed(9)
>>> a = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> A = S.Alphabet(['a', 'b'])
>>> seq = S.Sequence(a,A)
>>> block_entropy(seq, 6)
3.577559335188841
```

information.effective_complexity(*seq, n_max*)

Computes the effective complexity defined by Grassberger

..todo:

Make the doc of effective complexity

information.entropy_rate(*seq, wlen, method='average'*)

Returns the entropy rate

Parameters

- **seq** – a symbolic Sequence object
- **method** – a string in [‘lempel_ziv’, ‘average’]
- **kwargs** – parameter to pass to the method

Returns the entropy rate computed using the method

Example :

```
>>> np.random.seed(9)
>>> a = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> A = S.Alphabet(['a', 'b'])
>>> seq = S.Sequence(a,A)
>>> entropy_rate(seq, 6)
0.5962598891981402
```

information.metric_entropy(*seq*)

Returns Shannon's (metric) entropy of sequence

Parameters **seq** – a symbolic Sequence object

Returns a float

Example :

On fixe la seed pour pouvoir contrôler la génération de vecteur “aléatoires”

```
>>> np.random.seed(9)
>>> a = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> A = S.Alphabet(['a','b'])
>>> seq = S.Sequence(a,A)
>>> metric_entropy(seq)
0.6003511877776578
```

information.multi_information(*seq1*, *seq2*, *seq3*)

Computes the multi information for 3 symbolic sequences,

A kind of 3 variables mutual information (See Blanc J.L. & Coq J.O., J.Physiol. 2007)

Parameters **z** (*x*, *y*,) – three symbolic Sequences

Returns the three variable mutual information

Example :

```
>>> np.random.seed(9)
>>> a = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> np.random.seed(6)
>>> b = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> np.random.seed(3)
>>> c = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> A = S.Alphabet(['a','b'])
>>> seq1 = S.Sequence(a,A)
>>> seq2 = S.Sequence(b,A)
>>> seq3 = S.Sequence(c,A)
>>> multi_information(seq1, seq2, seq3)
-4.8757282800737656e-05
```

information.mutual_information(*seq1*, *seq2*)

Computes the mutual information for symbolic sequences

Parameters **y** (*x*,) – two symbolic Sequences

Returns the mutual information (float)

Example :

```
>>> np.random.seed(9)
>>> a = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> np.random.seed(6)
>>> b = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> A = S.Alphabet(['a','b'])
>>> seq1 = S.Sequence(a,A)
>>> seq2 = S.Sequence(b,A)
>>> mutual_information(seq1, seq2)
0.0002988020334349084
```

information.renyi_entropy(*seq, coef*)

Returns the Rényi entropy

..todo:

Make the doc of renyi_entropy!

Example :

```
>>> np.random.seed(9)
>>> a = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> A = S.Alphabet(['a','b'])
>>> seq = S.Sequence(a,A)
>>> renyi_entropy(seq, 0.9)
0.6088567303148161
```

information.shannon_entropy(*seq*)

Returns Shannon's (metric) entropy of sequence

Parameters **seq** – a symbolic Sequence object

Returns a float

Example :

On fixe la seed pour pouvoir contrôler la génération de vecteur "aléatoires"

```
>>> np.random.seed(9)
>>> a = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> A = S.Alphabet(['a','b'])
>>> seq = S.Sequence(a,A)
>>> metric_entropy(seq)
0.6003511877776578
```

information.topological_entropy(*seq*)

Returns the topological entropy

Parameters **seq** – a symbolic Sequence object

Returns a float

Example :

```
>>> np.random.seed(9)
>>> a = np.random.choice([0,1],1000,replace=True, p=[0.7,0.3])
>>> A = S.Alphabet(['a','b'])
>>> seq = S.Sequence(a,A)
>>> topological_entropy(seq)
0.6931471805599453
```

information.transfer_entropy(*seq1, seq1p, seq2*)

Computes the symbolic transfer entropy $T_{y \rightarrow x}$

we can use: $P(x|y) = P(x,y) / P(y)$ in the formula: $P(x+, x, y) \log (P(x+|x,y) / P(x+|x))$

but (see Kugiumtzis, 2011)

$-H(x+, x, y) + H(x, y) + H(x+, x) - H(x)$

gives a better implementation