

---

## Contents

紹介編	8
本書の構成	8
はじめに	8
本書の目的・限界・対象者・使い方	8
センサーベース/ソースベース解析とはなんぞや	9
スラングや用語の説明と unix 系のお約束	10
それぞれの研究に必要な環境・特色のまとめ	11
脳波センサーレベル研究	11
MEG センサーレベル研究	11
MRI 研究	11
MEG+MRI ソースレベル研究	12
EEG/MRI 同時計測研究	12
MNE-python とは	12
freesurfer とは	13
それぞれのソフトの関係性～それ office に例えるとどうなの？ ～	15
脳の研究	16
$\gamma$ オシレーション	16
何故皮質は $\gamma$ なん？	16
どんな事をする？	17
じゃあ統合失調症にどんな異常があるって？	18
診断ツール	18
つまり？	19
実験編	20
実験をしよう！	20
被験者の非協力の苦しみ	20
外来のノイズの苦しみ	20
体の動きによる苦しみ	21
眼球による苦しみ	21
視覚実験の苦しみ (視線)	22
視覚実験の苦しみ (メガネ)	23
視覚実験の苦しみ (画面との距離)	23
MRI 同時計測の苦しみ	23
聴覚実験の苦しみ	24

---

Resting 実験の苦しみ . . . . .	24
汗と凍えの苦しみ . . . . .	25
震えの問題の苦しみ . . . . .	25
ハゲの苦しみ . . . . .	25
デブと心電図の苦しみ . . . . .	26
青筋の苦しみ . . . . .	26
青二才と老人の苦しみ . . . . .	26
苦しすぎて死にたくなった? . . . . .	26
実験のディレイの測定 . . . . .	27
必要物品 . . . . .	27
MRI と脳磁計の準備 . . . . .	30
脳波計の準備 . . . . .	30
コンピュータの準備 . . . . .	30
<b>導入編</b>	<b>31</b>
OS の準備 . . . . .	31
Windows . . . . .	31
MacOS . . . . .	31
Linux(オススメ) . . . . .	31
開発環境の準備 (脳波、脳磁図の場合。出来るだけリッチに。) . . . . .	33
それぞれの使用感 . . . . .	33
僕の今のおすすめは? . . . . .	34
逆にお勧めしないもの . . . . .	35
構築 . . . . .	35
Jupyter の起動 . . . . .	38
jupyter での plot . . . . .	38
anaconda 仮想環境 . . . . .	39
R を jupyter で動かすために . . . . .	40
CUDA . . . . .	41
バージョン管理 git . . . . .	43
git サーバー . . . . .	43
jupyter で作ったスクリプトのバージョン管理 (小技) . . . . .	43
maxfilter(elekta のやつ) . . . . .	45
freesurfer のインストール . . . . .	46
MNE/python のインストール (脳波、脳磁図をする場合) . . . . .	48
公式のインストール方法 . . . . .	48

---

---

MAYAVI がインストール出来ない	50
HF5 をインストールしたい	51
jupyter kernel	51
CUDA	51
MNE/C のインストール	52
コラム 1-SNS の活用	52
freesurfer を使う (MRI)	53
recon-all 同時掛け (freesurfer)	54
freesurfer の解析結果の表示	55
解析結果のまとめ	55
画像解析の修正	55
mricon/crogl (MRI を使う場合)	59
MRI のファイルの変換	59
<b>解析編</b>	<b>63</b>
肩慣らし	63
MNEpython を使う前に学んでおくべきパッケージ	63
python を綺麗に書くために	63
numpy で遊ぼう	64
解析を始める前の warning!	65
jupyter 用おまじないセット	67
データの読み込みとフィルタリング・リサンプル (公式サイト版)	68
データの読み込みと filter, resample (僕の解説)	69
脳波読み込みの問題	70
日本光電	71
EDF で event 情報が読み込めない場合	72
EDF が少しも読み込めない場合	72
脳波のセンサーの位置が変則的な場合	73
脳波のセンサーからソーススペース出来んの？	73
脳波のセンサーの名前が変則的な場合	74
基準電極	75
トリガーチャンネル	76
bad channel の設定	77
やり方 1	77
やり方 2 (おすすめ)	77
interpolation	77

---

---

maxfilter . . . . .	78
ICA でノイズを取ろう . . . . .	79
ICA コンポーネントのより良い取り除き方 . . . . .	83
自動判定 . . . . .	83
半自動判定 . . . . .	84
Epoch と Evoked . . . . .	85
データの plot . . . . .	87
jupyter の場合 . . . . .	88
pysurfer . . . . .	89
numpy の plot . . . . .	91
多チャンネル抜き出し . . . . .	92
センサーレベル wavelet 変換 . . . . .	94
そもそも wavelet 変換とは何なのか . . . . .	94
wavelet 変換にまつわる臨床的な単語 . . . . .	96
2 つの流儀と MNE . . . . .	97
wavelet 変換の実際 . . . . .	97
データの集計について . . . . .	99
jupyter での R と pandas の連携 . . . . .	101
R での ANOVA について . . . . .	102
Connectivity . . . . .	103
fourier/multitaper モード . . . . .	105
wavelet モード . . . . .	105
plot . . . . .	105
indices モード . . . . .	106
ソースレベル MEG 解析 . . . . .	108
掛け算を作る . . . . .	109
割り算を作る . . . . .	110
その後のストーリー . . . . .	110
手順 0、freesurfer . . . . .	111
手順 1、trans . . . . .	111
手順 2、BEM 作成 . . . . .	113
手順 3、ソーススペース作成 . . . . .	114
手順 4、順問題 . . . . .	115
手順 5、コヴァリアンスマトリックス関連 . . . . .	116
手順 6、逆問題 . . . . .	117
手順 7 ソース推定 . . . . .	118

---

手順 8、前半ラベル付け . . . . .	119
手順 8 後半、label 当てはめ . . . . .	120
三次元 plot . . . . .	120
三次元 movie . . . . .	121
その後のお楽しみ 1、ソーススペース wavelet . . . . .	122
その後のお楽しみ 2、ソーススペース connectivity . . . . .	123
コラム 3-markdown で同人誌を書こう！ . . . . .	123

---

理論編	125
-----	-----

初心者のための波形解析 . . . . .	125
波形解析で得たいものと、必要な変換 . . . . .	125
フーリエ変換とは . . . . .	126
ShortTime フーリエ変換 . . . . .	126
Wavelet 変換とは . . . . .	126
ヒルベルト変換 . . . . .	127
フーリエ級数 . . . . .	127
複素フーリエ級数 . . . . .	128
結局何をしているのか . . . . .	131
スペクトル解析 . . . . .	131
そして wavelet 変換へ . . . . .	132
wavelet 逆変換と bandpass filter . . . . .	133
MorletWavelet とは . . . . .	134
気持ち悪さ対策 . . . . .	135
フーリエ逆変換を使って算出する Wavelet . . . . .	136
直交 Wavelet . . . . .	138
ヒルベルト変換 . . . . .	140
複素フーリエ級数とフーリエ逆変換の復習 . . . . .	141
フーリエ変換の負の位相の性質 . . . . .	142
相方の波を求める . . . . .	142
式の形にする . . . . .	143
さらに形を整える . . . . .	144
畳み込み定理 . . . . .	144
よく見るヒルベルト変換の式の導出 . . . . .	144
コネクティビティ各論 . . . . .	146
PLV と Coherence の問題点 . . . . .	146

---

PLV や Coherence の欠点の克服 . . . . .	147
拡散する前の電流 . . . . .	148
PLV の発展系 . . . . .	148
Coherence の発展系 . . . . .	149
で、こういうのってロバストなの？ . . . . .	149
グラフ理論 . . . . .	149
ソーススペース解析の理屈 . . . . .	150
もう一寸ちゃんと . . . . .	150
もっと良くする . . . . .	153
白色化 . . . . .	156
MAP 推定 . . . . .	157
dSPM の理屈 . . . . .	157
sLORETA の理屈 . . . . .	159
eLORETA の理屈 . . . . .	159
eLORETA の実行 . . . . .	160
割り算を極める☒ . . . . .	162
鶴亀算とは連立一次方程式である 🐢🐢 . . . . .	162
連立一次方程式とは行列の割り算である☒ . . . . .	163
鶴亀算の高級な解き方 🚗 . . . . .	163
実装 (python🐍) . . . . .	164
鶴亀てんとう虫算 🐞🐞🐞 . . . . .	164
無限個の解とその解決策 $\infty$ . . . . .	164
逆行列は作れないこともある 🐍 . . . . .	165
疑似逆行列 📐 . . . . .	166
ここまでで学んだ割り算の応用☒ . . . . .	166
具体的な応用多変量解析入門 . . . . .	167
まとめ . . . . .	168
さいごに . . . . .	168

補講	169
python での高速化のあれこれ . . . . .	169
for 文とリスト内包表記と map . . . . .	169
numpy(独自のメソッドを実装するときとか) . . . . .	170
並列化 (これがやりたかった！) . . . . .	170
OPENMP のオーバーヘッド対策 . . . . .	171
MNEpython の並列化 . . . . .	171

---

---

graph . . . . .	172
おすすめの参考書 . . . . .	173
おすすめサイト . . . . .	174
おすすめ SNS . . . . .	174
おすすめソフト . . . . .	175
MNEpython 実装時の小技 . . . . .	176
メソッド・チェーン . . . . .	176
変数を減らしてみる . . . . .	176
MNE の API 引数多すぎだろ死ね！ . . . . .	177
ここまでのまとめ . . . . .	177
解析失敗したやつをスキップしたいんだが . . . . .	178
file 名じゃなくてフォルダ名が欲しいん . . . . .	179
あとがきとお願い . . . . .	180
令和 2 年 2 月時点のあとがきだ . . . . .	180
令和 3 年 5 月時点のあとがきだ . . . . .	180
令和 5 年 6 月時点のあとがき . . . . .	181

---

## 紹介編

紹介編ではこれから何をしていくのか、何が出来るのかを語ります。  
まあ、適当です。

## 本書の構成

- 紹介編
- 実験編
- 導入編
- 解析編
- 理論編
- 補講

## はじめに

現代では脳は電気で動いている、と信じられています。  
しかし、どのような挙動なのかはまだまだ分かっていません。  
だから、貴方は研究をしたくなります。(それは火を見るより明らかです)  
しかし、脳の解析は難しく、技術的な入門書、特に和書に乏しい現状があります。  
だから同人誌を書くことにしました。  
本書では脳磁図、脳波、MRI 解析を「体で覚える」べく実践していきます。  
さあ、MNE/python、freesurfer の世界で良い生活を送りましょう！

## 本書の目的・限界・対象者・使い方

MNE/python<sup>12</sup>や freesurfer<sup>3</sup>を用いて脳内の電源推定...特にソースベース解析を行うための  
解析環境の構築と解析の基礎を概説します。可能な限り効率的な解析環境を構築し、楽をします。  
僕が個人的に考えている事もちょくちょく書きます。  
僕は脳波計も脳磁計も使用経験があるのですが、脳磁計はレアです。

---

<sup>1</sup>Gramfort, M. Luessi, E. Larson, D. Engemann, D. Strohmeier, C. Brodbeck, R. Goj, M. Jas, T. Brooks, L. Parkkonen, M. Hämäläinen, MEG and EEG data analysis with MNE-Python, Frontiers in Neuroscience, Volume 7, 2013, ISSN 1662-453X

<sup>2</sup>Gramfort, M. Luessi, E. Larson, D. Engemann, D. Strohmeier, C. Brodbeck, L. Parkkonen, M. Hämäläinen, MNE software for processing MEG and EEG data, NeuroImage, Volume 86, 1 February 2014, Pages 446-460, ISSN 1053-8119

<sup>3</sup><https://surfer.nmr.mgh.harvard.edu/fswiki/FreeSurferWiki>



---

僕は elekta 社の MEG を使っていたので脳磁計は elekta 前提で書きますが、MNE-python は大抵の脳波計に対応しているので、実験用脳波計さえあればなんとかなるでしょう。

本書の限界は僕のリソース不足<sup>4</sup>と、これが同人誌であること、MNE 自体の進化のスピードが光の速さであることです。不確実なものとして、疑って読んで下さい。この同人誌は不完全なため、更新があります。内容は結構多いので、辞書的に使うのもいいでしょう。

本書の対象者は以下のとおりです

- 脳波/脳磁図計を使って研究をしたい初心者
- 脳磁図計を使って研究しているけれど、コーディングが苦手な中級者
- 頭部 MRI 研究で freesurfer を使いたい初心者

また、前提条件としてターミナルやプログラミングを怖がらないことがあります。黒い画面を見たら怯える君！ 君は本書の対象外だから出ていきなさい！<sup>5</sup> 黒い画面を触ったことないけど、やる気はある君！ 君は歓迎します。プログラミング未経験者の質問にも出来るだけ答えたいと思います。

脳研究の経験者は MNE/python とはから読んでいけばいいです。MNE/freesurfer 経験者なら OS の準備から読めばいいです。コンピュータは自転車みたいなもので、基本は体で覚えていくしかないと思っています。分からなければググることが大事です。qiita<sup>6</sup>等で検索するのも良いでしょう。

## センサーベース/ソースベース解析とはなんぞや

脳の中の電気信号を調べる方法としては脳波や脳磁図<sup>7</sup>が有名です。脳波や脳磁図のセンサーで捉えた信号を直接解析する方法をセンサーレベルの解析と言います。これは伝統的なやり方であり、今でも多くの論文がこの方法で出ている確実な方法です。

しかし、脳波や脳磁図は頭蓋骨を外して直接電極をつけないと発生源 (僕達はソースと呼びます) での電気活動はわかりません<sup>8</sup>。

---

<sup>4</sup>当初は僕のスキル不足が本書の限界でしたが、MNE-python を魔改造しながら使っているくらいなので、流石に自称中級者名乗っていいと思う。MNE-python ってユーザーのスキルが上がると、自分で魔改造できるんですよ。知ってました？

<sup>5</sup>僕は怖いおじさんなのです。

<sup>6</sup>日本のプログラマ用の SNS の一つです。

<sup>7</sup>脳波は電気信号を捉えますが、脳磁図は磁場を捉えます。電気と違って骨を貫通しやすく拡散しにくいので空間分解能に優れますが、ノイズに弱いです。値段も高いです。 <http://www.elekta.co.jp/products/functionalmapping.html>

<sup>8</sup>動物実験では脳に電極刺す実験はされていますが、人に刺すと警察に捕まります。

---

普段計測している脳波・脳磁図は所詮は

「神経から頭蓋骨を通り抜けて漏れてた信号」に過ぎないのです。

では、一体どうすれば脳内の電気信号を非侵襲的に観察できるのでしょうか？

方法は残念ながら<sup>9</sup>のですが、推定する方法ならあります。

その中の一つの方法として、脳磁図と MRI を組み合わせ、

MNE という python パッケージを使って自ら解析用スクリプトを実装する方法があります。

ソースベース解析というのはあくまで推定であり、

先進的である一方でまだまだ確実性には劣るやり方との指摘もあります。

ちなみに、脳波のソースベース解析はあるにはあるのですが、

脳波は電流であるため磁力と違って抵抗を避けて流れます。

実際、脳波でのソースベース解析とセンサーベース解析の結果が

不一致であったという研究が発表されています。<sup>10</sup>

脳波でやるならば、40 チャンネル以上の脳波計を用意しましょう！

まあ、後述しますが、脳波と脳磁計じゃ他にも得意分野の違いがあります。

## スラングや用語の説明と unix 系のお約束

本書では下記の言葉を使っています。伝統的なスラングを含みます。

適宜読み替えてってください。それ以外にも色々スラングあるかもです…。

何故スラングをそのまま書いているかって？

お前は同人誌にまで正しい日本語を求めるのですか？ そういう人は回れ右。

僕は怖いおじさんなのです。

- hoge: 貴方の環境に応じて読み替えてください、という意味のスラング  
fuga, piyo も同じ意味です。ちなみにこれは日本語です。  
英語が好きな方は foo とか bar とかになりますね。
- 叩く: (コマンドをターミナルから) 実行するという意味の他動詞
- 回す、走らせる: 重い処理を実行するという意味の他動詞
- ターミナル: いわゆる「黒い画面」のこと。Mac ならユーティリティフォルダにある。
- .bash\_profile: ホームディレクトリにある隠し設定ファイルです。  
環境によって .bashrc だったりしますし、両方あることもあります。  
貴方の環境でどちらが動いているか (両方のこともある) 確認して設定してください。
- 実装: プログラミングのことです。プログラムを書くことです。

---

<sup>9</sup>他に脳の活動を調べる方法として磁力を照射する fMRI や赤外線を照射する NIRS などがあります。fMRI は電気信号ってわけでも無さそうです。NIRS は赤外線で脳血流を捉えるのですが、頭皮の血流をいっぱい拾ってしまうので大変です。

<sup>10</sup><http://biorxiv.org/content/early/2017/03/29/121764>

---

本書で「インストールにはこうします」とか言ってコマンドを示した場合は文脈上特に何も無い場合、ターミナルでそれを叩いてくださいという意味です。python の文脈になったら python です。この辺りは見慣れれば判別できます。

## それぞれの研究に必要な環境・特色のまとめ

それぞれの研究の、僕の感想を述べます。

### 脳波センサーレベル研究

- 時間解像度：ミリ秒単位
- 空間解像度：10cm 単位
- 見ているもの：神経活動
- 必要物品：MNE-python

安価で普及していますが、まだ多くの謎が眠っている分野です。脳回や深部の信号に強いですが、脳脊髄液や頭蓋骨を伝わって行くうちに信号が拡散してしまうため、空間分解能が低いです。ポータブル脳波計とかあって、BMI とか出来る夢のある分野ですね。

### MEG センサーレベル研究

- 時間解像度：ミリ秒単位
- 空間解像度：cm 単位
- 見ているもの：神経活動
- 必要物品：MNE-python

ノイズに弱く、脳の深部に弱く、莫大な資金が必要な希少な機器です。ソース推定しないならあまり面白みはないかな。まず、脳波と違って脳溝・浅い部分をよく見れるという特徴があります。

### MRI 研究

- 時間解像度：分単位
- 空間解像度：mm 単位
- 見ているもの：血流や水分

- 
- 必要物品: freesurfer、mricrogl

脳内の水分子に磁場を与えて画像化する凄い機械です。  
画像なのでゆるふわかと思いきや、量子力学の知識を要求します。  
莫大な資金が必要ですが、それなりに普及しています。  
血流を見ることが出来るので「脳機能」というか  
「血が流れているから働いてるんじゃないか？」的な事も大まかに見れます。  
ネタが尽きようとも、新たな理論を持ち出してくる根性の分野です。

## MEG+MRI ソースレベル研究

- 時間解像度: ミリ秒単位
- 空間解像度: 1~数 cm 単位
- 見ているもの: 神経活動

MEG と MRI を組み合わせて脳の電気活動を見るやつです。  
本書の本題です。MRI の空間分解能と脳磁図の時間分解能を備えた  
まさに †最強の解析†...のはずなんですけど、どうなのでしょうね？  
脳波の場合はそこまで空間分解能は高くないっす。モワッとしてます。  
脳磁計を使った場合は空間分解能と時間分解能が合わさり、なかなか凄いです。

## EEG/MRI 同時計測研究

- 時間解像度: ミリ秒単位
- 空間解像度: 数 cm 単位
- 見ているもの: 神経活動

EEG と MRI を同時に計測するという究極の方法です。  
ただし、MRI のノイズは凄まじく、ノイズを取るのが困難です。  
専門家でなくては施行できず、本書では取り扱いません。

## MNE-python とは

脳磁図を解析するための python<sup>11</sup>用 numpy, scipy ベースのパッケージです。  
自由度が非常に高いです。(引き換えに難易度が高いです。)

---

<sup>11</sup> コンピュータ言語の一つ。速度を犠牲にして、読み書きやすさを追求した言語。科学計算の世界では現時点では広く普及しています。MATLAB と似ている...とかいう人もいるけど、僕は似てないと思う。無料でオブジェクト指向の汎用言語なので、応用範囲が Web サーバーとか機械の制御にまで及び、習得して損をすることはまずないでしょう。

---

ソース推定をするためのパッケージなのですが、  
Wavelet 変換とか ICA とか Permutation 検定とか、その他あらゆる事が出来ます。  
出来るのですが...使いこなすためには生理学、数学、工学の知識が必要です。  
ちなみに元来脳磁図用なのですが、脳波を解析することも出来ます。  
というか、脳磁計なんて値段がおかしい割に使い道が狭いので、実質脳波用。  
C 言語で実装された MNE/C というのもありますが、古いバージョンと考えていいです。<sup>12</sup>  
開発は活発で、最近新バージョンは MNE/python 0.16.1 です。  
freesurfer は 6.0 が、python は python3.9 が最新です。  
導入と紹介を書いていこうと思います。  
python2 はマジでオワコンなので特別な理由がない限り使わないように。いいね？

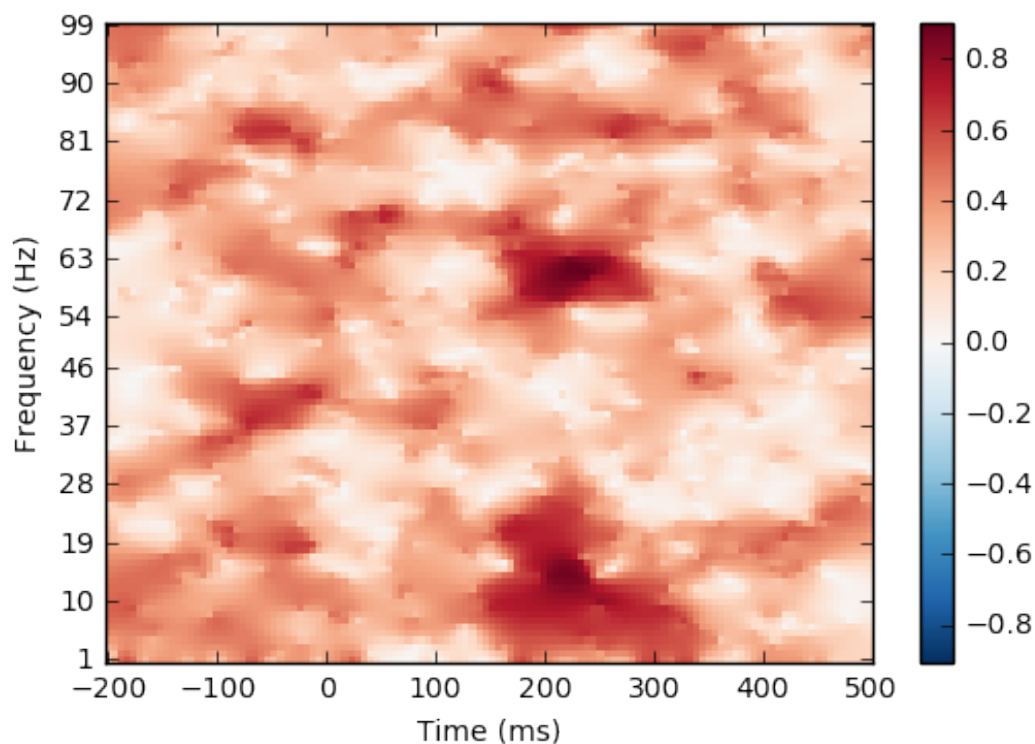


Figure 1: wavelet 変換の出力例

## freesurfer とは

頭部 MRI を解析する為のソフトです。自動で皮質の厚さやボリュームを測れるだけでなく最近では fMRI でコネクティビティの算出が出来るようになるなど、かなり賢いです。特に厚さに強いです。

<sup>12</sup> 良いところもあるらしいです。使ったことないから分かん。

---

反面、計算激重な上にサイズが大きくターミナル使う必要があります。

Unix 系 OS じゃないと動きません。

その上、違う CPU 使ったら結果が変わる仕様があり、正しく扱わないとジャジャ馬と化します。

最近頭部 MRI 研究で勢力を伸ばしつつあり、最早スタンダードの一つだそうです。フリーウェアです。

---

それぞれのソフトの関係性～それ **office** に例えるとどうなの？ ～

実は、以前人に教えようとした時、  
いきなり MNEpython と言われても初心者にはよくわからないと言われました。  
unix 系もコンピュータ言語も触ったことない人には例え話のほうが良いかもしれないので、  
初心者のために、登場するソフトの名前を例え話で話してみます。  
凄く乱暴な例えではあります。

---

MNE	役割	オフィスに例えると？
anaconda, pip, homebrew	ソフトをインストールするソフト	app store, google play, 人事部
spyder, jupyter	実際に色々書いたりするソフト	word, excel, 筆記用具
python	言語	日本語, 命令書の書式, 社内文書
MNE	言語で動く命令セット	excel の関数, 社内文書に従って動く部下
mricron/mricrogl	変換・表示用ソフト	画像変換ソフト, 通訳
freefurfer/freeview	MRI 画像処理ソフト	Photoshop, 絵の具, MNE と違う部署の部下

---

---

## 脳の研究

君は、ここまで読んで「何やら難しそうだが、それで何かできるん？」  
と思ったかも知れない。なので、ここで超簡単なレビューを書いてみます。  
僕は精神病の脳の生理学が専門なので、精神病の誘発 $\gamma$ 波についてすごく簡単に書いてみます。  
もちろん、他にも MMN とか resting とか色々あるんですが、とりあえず雰囲気だけね。  
参考になれば。

### $\gamma$ オシレーション

脳には実際に演算をする装置である灰白質と、演算結果を伝える白質という部分があって、  
脳の表が灰白質、内側が白質であります。で、今回は灰白質の方を見たいと思います。  
理由ですか？ 脳の電気活動を見るときは表面のほうが見やすいのです！  
で、 $\gamma$  波 (30Hz 以上) を見ます。  
理由ですか？ 皮質活動で発生する波は $\gamma$  だからです！  
この $\gamma$  波の事をカッコつけて $\gamma$  オシレーションとか言ったりします。

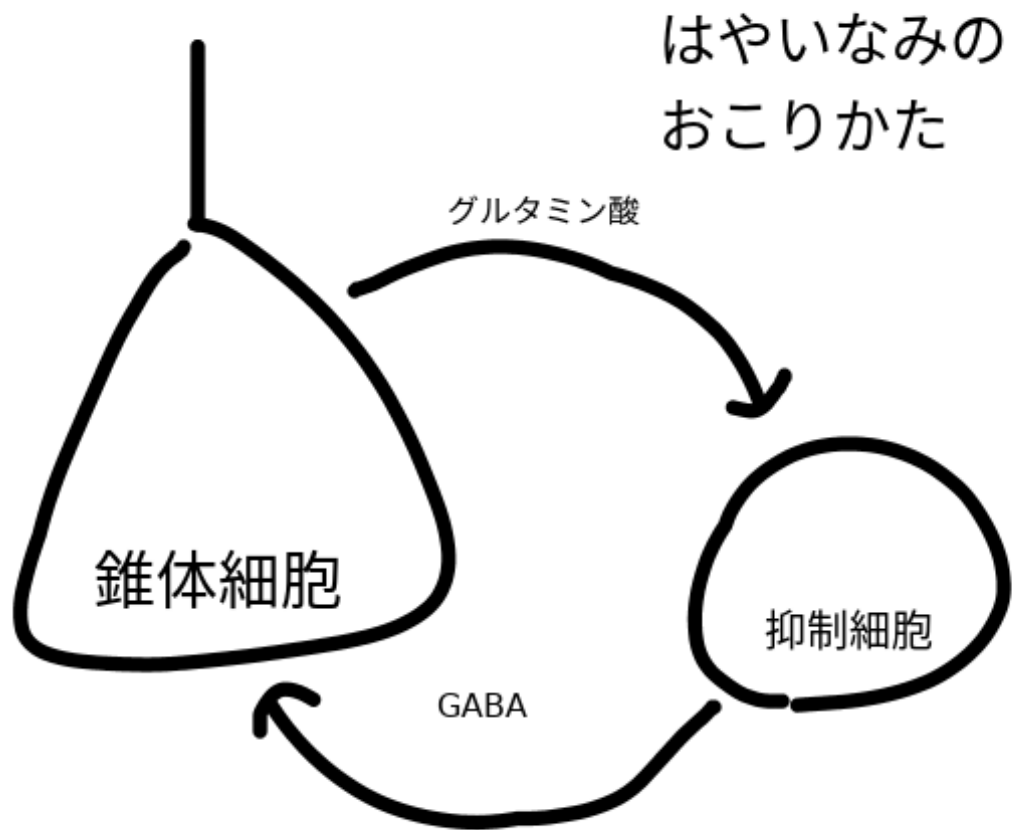
何故皮質は $\gamma$  なん？

脳内には以下の細胞があるのです。

- 興奮性の神経細胞 (錐体細胞)
- 抑制性の神経細胞 (介在細胞)

興奮性のやつはグルタミン酸 [ $\gamma$ ajinomoto] を放出するし、抑制性のやつは GABA を放出します。  
興奮細胞と抑制細胞は互いに接続しています。  
図に示すと、下記の様な感じです。  
で、細胞の興奮というのは一度「ピコーン！」と興奮すると暫く休む性質があります。  
暫く休む性質があるために、神経細胞は興奮の頻度が決まってきます。  
そのピコピコ神経細胞が興奮する速度がだいたい 30Hz 以上ってわけです。





![^ajinomoto]: これは味の素です！

どんな事をする？

統合失調症を見てみましょう。理由ですか？

その分野はいっぱい研究されてて同人誌を書くのに都合が良いからです！

さて、顕微鏡を使った研究があります。

- 聴覚野 (脳において聴覚を司る部位) の錐体細胞に異常がある<sup>13</sup>
  - 軸索 (神経細胞から伸びる枝) の密度が低い<sup>14</sup>
- 等が統合失調症で報告されています。

---

<sup>13</sup>Robert A Sweet, Sarah E Bergen, etal: Pyramidal cell size reduction in schizophrenia: evidence for involvement of auditory feedforward circuits, Biological psychiatry, 55-12:1128-1137,2004

<sup>14</sup>Robert A Sweet, Sarah E Bergen, etal: Anatomical evidence of impaired feedforward auditory processing in schizophrenia, Biological Psychiatry, 61-7:854-864,2007

---

実際は神経の密度とかより先に $\gamma$ 波のおかしさのほうが発見されたんです。  
そして、この神経細胞の異常が $\gamma$ 波のおかしさの原因だろうと言われています。

じゃあ統合失調症にどんな異常があるって？

挙げるの？ 数限りないぞ！

- 計算中に前頭葉の $\gamma$ 振動が減弱している<sup>15</sup>
- 特定の模様を見せた時の $\gamma$ 振動が健常者と違う<sup>16</sup>
- 40Hz の音を聞かせた時の $\gamma$ 波が減弱している<sup>17</sup>

面倒だからもう書かないけど、実は凄くいっぱいある。<sup>18</sup>

で、先のリストを見て君はあることに気づいたはずだ。 $\gamma$ 波は誘発できる。  
って事は、いろんな刺激をして、その刺激中の脳波を見れば  
それで論文書けるんじゃないかね？ ヒャッハウ！ 博士号ゲットだぜ！  
(言うほど簡単じゃないけどな...)

というわけで、顕微鏡を見るだけが能じゃなくて、  
生きている人間の脳活動を直に見ることが出来るということです。

## 診断ツール

さて、40Hz の音を聞かせた時に 40Hz の脳波が出るとか、結構ずるい感じがしますが、  
不思議なことにもっと低い周波数ではあまり差が出なかったり、  
健常者やうつ病では差が出なかったり<sup>19</sup>と、確かにそういうのあるらしい。  
おっ、統合失調症と鬱を分けられる...って思うじゃん？  
躁鬱病でも差が出るらしい<sup>20</sup>。チッ、特異的じゃないじゃん。  
でもまあ、医者<sup>21</sup>としては考える材料が増えるのは嬉しいんですよね。

---

<sup>15</sup>Johanna Kissler, Matthias M Muller, etal: MEG gamma band activity in schizophrenia patients and healthy subjects in a mental arithmetic task and at rest, Clinical Neurophysiology, 111-11:2079-2087,2000

<sup>16</sup>Kevin M Spencer, Paul G Nestor, etal: Neural synchrony indexes disordered perception and cognition in schizophrenia, Proceedings of the National Academy of Sciences, 101-49:17288-17293,2004

<sup>17</sup>Jun Soo Kwon, Brian F O'Donnell, etal: Gamma frequency-range abnormalities to auditory stimulation in schizophrenia, Archives of general psychiatry, 56-11:1001-1005,1999

<sup>18</sup>僕の文献管理ソフトの中には山のように入っているけれど、そんなのここにコピペしてもしょうがないでしょ？

<sup>19</sup>Shuichi Isomura, Toshiaki Onitsuka, etal: Differentiation between major depressive disorder and bipolar disorder by auditory steady-state responses, Journal of affective disorders, 190:800-806,2016

<sup>20</sup>BF O'donnell, JL Vohs, etal: Auditory event-related potential abnormalities in bipolar disorder and schizophrenia, International Journal of Psychophysiology, 53-1:45-55,2004

<sup>21</sup>ちなみに、僕は精神科の医者をやってご飯を食べています

---

つまり？

ま、そういう風な分野のことが出来ます。  
他にも色々出来るでしょうね。

---

## 実験編

### 実験をしよう！

では、実験について色々書こうと思います。  
研究するならもちろん厳密な実験が良いに決まっています。  
というか、最低限の実験の綺麗さを担保しなければ成立しません。  
しかし、実験というものは苦しみに満ちています。  
実験を立ち上げる時に考慮すべき苦しみを  
出来れば対処法とともに順不同で列挙します。  
言い過ぎに見える所もあるかも知れませんが、マジで苦しみに満ちています。  
君は生き残ることができるか？

### 被験者の非協力の苦しみ

被験者が実験に参加しているのは当たり前だと思われるでしょうが、  
そんなことはないです。  
「目をつぶって音を聞く実験」に参加した被験者が  
居眠りをしていないと誰が言えましょうか？  
「画像をみる実験」でちゃんと画像を見ていると保証できますか？  
まずは、その辺りをちゃんと説明して、協力できないと駄目です。  
そのあたりの協力関係を築く事がとても大事です。  
「は？ そんな事が問題になる事があるのか？」って  
貴方はなったかも知れませんが、あるんだなこれが。

### 外来のノイズの苦しみ

MEG も EEG もノイズに弱いです。  
どこからノイズが飛んできているのかちっともわからないことがあります。  
Wifi が電子レンジに弱いなんてやわなもんじゃない。  
もっと深刻なものを体験できます。  
そんな時は EEG 用のシールドルームを用いますが…  
シールドによって得意なノイズが違っていたり、  
シールドルーム自体がやはり高額だったりします。

---

EEG に比べて MEG だと自体は深刻です。  
地球の核から飛んでくる磁場は人間の頭から放出される磁場の 1000 倍程度。  
シールドルームは必須です。  
近くに地下鉄が通っているとそこから出る強力な磁場は  
シールドを一部貫通します。

当然のことながら、MEG のシールドルームに  
視覚刺激提示用の画面を置くと物凄いノイズが乗って酷くなります。  
視覚実験苦しい！  
ノイズの出にくい画面というのがあるのですが、これがまたお高いです。  
というか、ノイズの出にくい画面からもノイズです！  
音の場合は MEG の部屋の外からパイプで伝える事でなんとかはします。  
あと、単に金属が近くにあるだけでノイズになったりします。  
ベルトは外して貰いましょう。  
メガネは視覚実験の場合は検討が必要ですので後述します。  
色々外しますが、どうあがいても銀歯はとれません。  
この場合、金属に強力な磁場を当てることでノイズを散らす方法も  
MEG の場合にはあるのですが、どちらにしる辛いです。

#### 体の動きによる苦しみ

被験者は、実験以上に拘束時間が長いです。  
なぜかって？ EEG や MEG を装着する時間はそれなりに掛かるからです。  
お爺さんとかなら「腰が痛い、動いていいかの？」ってなりますよね。  
体が動くと、ものすごいノイズになります。  
被験者に耐えられるような実験を組まねばなりません。

#### 眼球による苦しみ

眼球が動くと脳波・脳磁図に大きな影響が出ます。  
眼球運動のノイズは脳波などよりずっと大きいです。  
眼球自体が強く帯電していて、目自体が結構ピクピク動くからです。  
本来は、瞼なんか全部切り取って、片目を潰して、  
眼球運動の筋肉を全て切除して、視神経の一部も一部切除したいくらいです。<sup>22</sup>

---

<sup>22</sup>動物実験でしてる人は居るらしいですが、人間にやると警察に捕まります。

---

ほぼ全ての脳波・脳磁図実験に於いて眼球運動は大敵です。  
実はそれをソフトウェアによって取り除くことは出来るのですが、  
そもそもノイズいっぱいデータは良くないです。  
データは確実に劣化するでしょう。

被験者には出来るだけ視線を動かさないように教示しましょう。  
あとは fixation を使う方法があります。  
fixation は「実験中はここを注目しておいてね！」という小さな印です。  
色んな実験の前に、そこに視点を合わせておいて、消してから刺激提示です。  
問題点としては、その印自体が脳に影響するかもということ、  
本当にそれを見ているかも保証できない事、  
視覚実験の最中は出来ないこと…つまり、端的に言うとショボいです。  
しかし、デメリットの少ない方法で有力なので、よく使われる方法ですね。

### 視覚実験の苦しみ (視線)

貴方は被検者さんに画像とかを見せることになりますが  
貴方の提示した画面を被験者さんは本当に見ているのでしょうか？  
見ているとして、本当に真正面から見ているのでしょうか？  
視野が少しでもずれたら大きく結果が変わるんじゃないかと厳しい人は言います。<sup>23</sup>  
視覚実験の場合は眼球運動に加えて  
「この人はちゃんと画像を見ているのか」が問題になります。  
サボって目を閉じて眠っているかも知れない。  
刺激が眩しかったり、嫌な画像から目をそらすかも知れない。  
そこで、下記の二つが有力な方法です。

- Web Camera
- Eye Tracker

Web Camera は要するに Web Camera で被験者の顔を観察して、  
目で見て「お前目をそらしただろ！ 不合格！」と判定するやり方です。  
しかし、どうでしょうか？「貴方は Web Camera 越しにその人の見ている物が  
分かるのですか？ 恣意的な要素が入っていませんか？」  
というツッコミが来たら終わりです。要するにショボいです。

もう一つは眼球運動を監視する為の専用のカメラで眼球を監視して、  
ちゃんと使えるデータだけ使うというやり方です。

---

<sup>23</sup> 厳しすぎない？

---

なんと、こいつぁショボくないのです！ やったね！  
トビーとかが販売していますが、実験用は値段が高いです。

### 視覚実験の苦しみ (メガネ)

もちろんですが、目の悪い人は視覚実験は辛いです。  
メガネをつける事になりますが、MEG の場合は眼鏡の金具が  
ノイズ源になりますので、木のメガネをかけさせてあげる必要があります。  
EEG の場合はそこまで酷くはないですが、それでも金属は帯電するので  
気にはなりますね。  
Eye Tracker を併用する時にメガネって気になりませんか？

### 視覚実験の苦しみ (画面との距離)

下記を気にする必要があります。

- 画面の大きさ
- 画面からの距離
- 画面の明るさ
- 画面のディレイ
- 画面のジッター

前半3つは巻き尺とかを駆使する必要があります。  
後半二つは光センサー等を駆使する必要があります。  
あとでオシロスコープを使ってそれらを測定する方法を書きます。

### MRI 同時計測の苦しみ

MRI と脳波の同時測定ですか？  
そりゃ、やっている人は居るけれど、本書の著者には無理です。  
MRI の磁力と人間の脳から出る磁力は、だいたい 10 の 15 乗以上の差があります。  
MRI の部屋は磁気カードは壊れるし、ボールペンは空を舞う世界です。  
そんな酷い環境で脳波を測定するのは…まさに深淵です。  
専門知識が必要です。

---

## 聴覚実験の苦しみ

まず初めに、辛いことを言います。ヘッドホンではダメです。

聴覚実験の場合、音の大きさが重要になるからです。

音の大きさは音圧計という機械で測ります。

音の大きさが不揃いだと実験になりませんが、

音は反射したり、干渉したりする性質があることはご存知でしょう。

そして、耳の形は人によって全然違います！

ヘッドホンを付けたときのズレも毎回変わります！

耳の形やズレを無視できるデバイスが求められますね。

耳の中に入れるタイプのインサートイヤホンが最適解です。

インサートイヤホンの先っぽを鼓膜の所まで差し込むのです。

これならズレとか耳の形には関係ないですね！

それから、音の大きさはきちんと測っておく必要があります。

音圧計でインサートイヤホンから出る音の大きさを

測定する必要があります。

インサートイヤホンはビニールのケーブルで音を伝えるのですが、

でもインサートイヤホンってゴムのチューブみたいなもので出来ています。

つまり、途中がビヨンビヨンです。

なんと、このケーブルを通すことで音が劣化するのです！

さらに、イヤホンから出た音の大きさを、耳の中と同じ条件で

測ることって出来るでしょうか？ これ、超絶難しい問題です。

耳の中にインサートイヤホンとマイクを一緒に入れるなんて出来ません。

実は無理じゃないらしいんですが、かなり値の張る音圧測定器が

必要になります。

この世は苦しみに満ちています。

## Resting 実験の苦しみ

何も刺激をせずにすればいいじゃないかとなりそうですが、

これもこれで色々考える所があります。

まず、MRI の Resting は Resting じゃないと僕は思っています。

(MRI の Resting 研究の人にブチ殺される発言)



---

MRI なんか、ガンガン音がするんだから絶対純粋な resting じゃないですよ…  
閉所恐怖症の人なら変な感じの脳活動が出てるかも知れません。

なら、脳波って結構 Resting 出来るんじゃない？ って思いますよね？  
ですが、問います。貴方は心を無にすることが出来ますか？  
脳波をつけられて、何も考えるなと言われてじっとしている時、  
貴方は今日の献立や彼氏との喧嘩が頭に浮かんだりしているかもしれない。  
「いけないわ！ 私は今、彼氏との喧嘩を思い出しちゃいけない！」  
そう必死になって心を「無」にしようと足掻く。  
その時に出ている脳波は「無」ではなく「足掻く」脳波かも知れない。

ま、それは実験的には仕方ないのですけれども、  
気に留める必要はあるのではないのでしょうか？

#### 汗と凍えの苦しみ

汗はノイズになります。震えもノイズになります。  
空調がちゃんとなっていないと難しいのです。  
それだけではなく、湿度の管理も大事ですね。  
というわけで、温度と湿度の管理をしましょう！

#### 震えの問題の苦しみ

パーキンソン病の人の実験をする場合、  
体が震えるため、脳波が揺れ動くように見えることがあります。  
そんなの止めることはできないので  
この手の人は限界がありますね。

#### ハゲの苦しみ

ハゲは脳波用のゲルが流れていきやすい性質があります。  
それによって、脳波計によっては短絡のリスクがつきまといます。  
さらに、ハゲ頭は皮が分厚いです。  
だから、電流が通りにくいです。脳波的には不利です。  
キャップ型とかだと中々しんどいです。

---

## デブと心電図の苦しみ

デブは首もデブです。

心臓は心電図が出ますが、首が細ければ＋と－がぶつかりあってノイズが消えます。

デブの場合は心電図のノイズが消えないのです！

心電図はソフトウェアによってあとで取り除くことは出来ますが、それでもノイズが入ること自体が罪です。

## 青筋の苦しみ

頭の血管もまた、脈打ちます。

この、血管にセンサーがちょうど当たったらノイズになります。

この脈波は心電図に比べてソフトで外しにくいです。

## 青二才と老人の苦しみ

歳をとると脳波は変化します。若い人も脳波が幼いです。

20歳とかでも幼い脳波が出る人は居るし、歳をとるだけで

脳波の $\alpha$ 波は遅くなります。

だから、実験の時点で年齢をある程度合わせねばなりません。

永遠の18歳？ 関係ないね！

## 苦しすぎて死にたくなった？

確かに全部致命的だけれど避けにくいのはある。

そういうのがあって事を理解した上で、説得力を出すことが大事なんだよ。

厳格すぎる実験で結果が出ても

~~そんな厳格な実験でしか出ない結果なんて微々たる差だし臨床応用できなくね？~~

私からは以上です。

---

## 実験のディレイの測定

被験者に何かを見せたり聞かせたりしてその反応を脳波や脳磁図で拾ってくる実験をしたいとします。脳はかなりの性能なので、30ms 後には反応が始まります。

さて...ここで困ったことが起こります。実はコンピュータから画面やスピーカへ信号を送る時、一瞬で届いてくれないことがあります。理由は、現代的なコンピュータは様々なタスクを同時進行でやっていたり、性能を確保するために様々な工夫をしますが、それが仇になるのです。実験中に他の処理が割って入ってきたり、刺激に沢山の演算が必要だったりして時間を取られたりすると信号が一瞬で届きません。

なので、どのくらいの時間で出来るかを測定しておく必要があります。

ここでは僕がどうやったかを書いておきます。

僕がやったのは被験者の目の前の画面に図が表示されるまでと、「画面提示」の信号が脳磁図計に届くまでの差を調べることです。

## 必要物品

- オシロスコープ本体  
picoscope2000 というのを使いました。windows で動きます。かなり可愛いオシロスコープですが、入力 2 チャンネル、トリガー出力 1 チャンネル、 $\mu s$  単位の反応速度を持っています。25000 円くらいですが、家電量販店には売ってないです。
- センサー  
これは明るくなると抵抗が減るダイオードです。  
光センサーが RPM22PB で 200 円くらいです。amazon で買ったあと品切れになりました。今買うなら RPT-37PB3F のほうが在庫が多そうです。頑張って探しましょう。  
スペック上 10  $\mu s$  前後の反応速度のようです。
- 乾電池  
上記センサは 5V くらいの電圧では全然問題ないため、  
乾電池二本直列程度なら回路の途中に抵抗は要らないようでした。  
ていうか、2 本じゃ足りないかも。3 本あった方がいいかも。  
この辺りは、スペック表とにらめっこして下さい。
- ヒートシュリンクチューブ  
ドライヤーを当てると縮むチューブです。  
光センサーが壊れたら嫌なのでヒートシュリンクチューブで守りました。

あとはジャンパー線、鱗口クリップ付 BNC 同軸ケーブル、  
半田と半田ごて、電池入れが必要でした。BNC 同軸ケーブル周りはやや入手難度が高かった印象です。

---

どのようにしたかというと、光センサーを電池につなげてオシロスコープに繋がります。  
オシロスコープには2つの入力チャンネルがあるので、もう一方を刺激提示用コンピュータに繋がります。  
さらに、ディレイ測定用コンピュータにも繋がります。  
それで、刺激提示させた刺激を光センサーで捉え、差分を測定用コンピュータで受け取ります。

図にするとこうでしょうか…

もちろん、聴覚実験のときはまた違うと思います。  
先輩と協力して聴覚実験のディレイを測ったときは、  
直接イヤホンジャックからの信号をオシロスコープにブチ込んで測りました。  
音の種類にもよりますが、聴覚実験の方が少し簡単かもです？

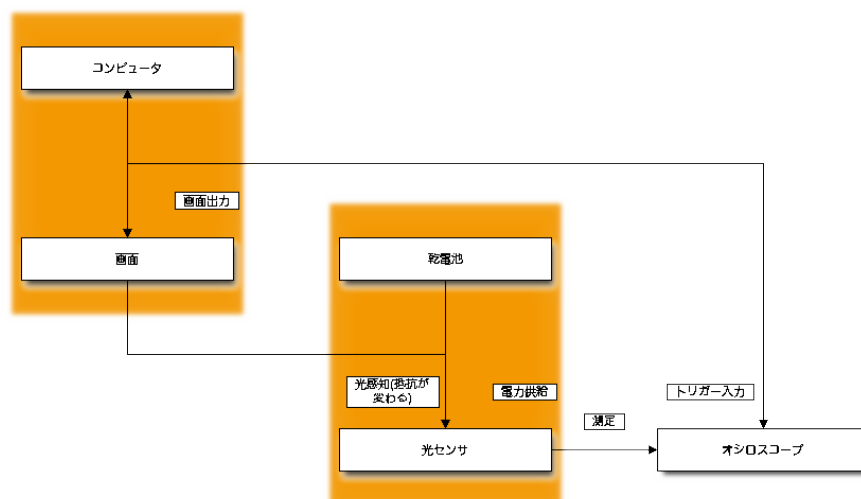


Figure 2: オシロスコープ図

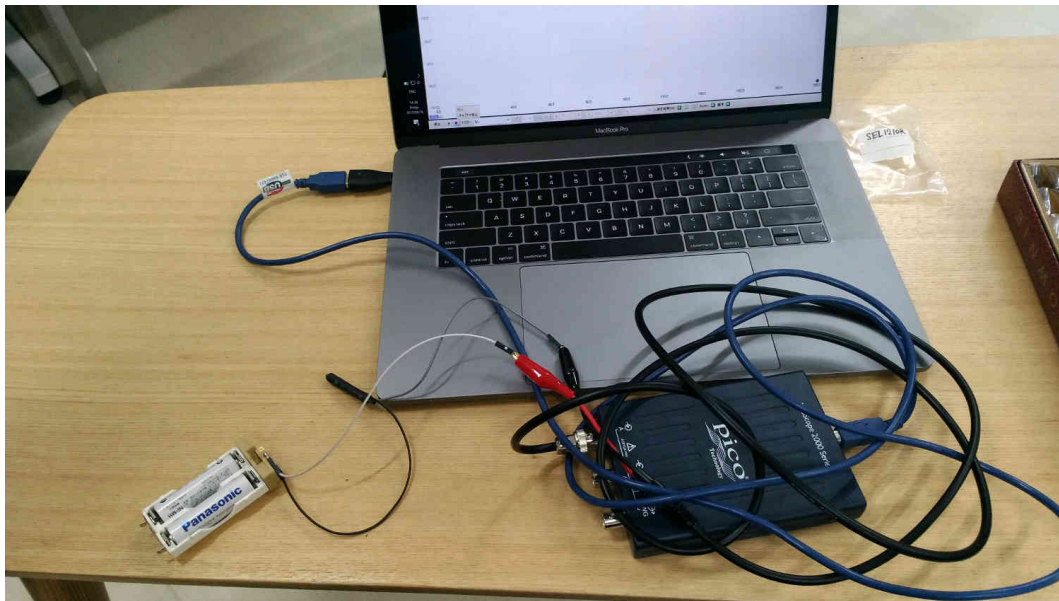


Figure 3: 光センサー+オシロスコープセット。総額3万円くらいしました。半田ごてを使わないといけなので、やけどに注意する必要があります。手前の小さな箱がオシロスコープ、左にあるのが電池、オシロスコープと電池の間にあるのが光センサーを回路にはんだ付けしたものをヒートシュリンクチューブで保護したもの、奥にあるのはコンピュータです。

---

## MRI と脳磁計の準備

必要ですが †億単位の金† が必要なので本書では割愛します。  
読者の中で個人的に買える人が居るなら買うと良いんじゃないかな。

## 脳波計の準備

脳波研究は気軽で良いですね。これは脳磁図研究ほどお金はかかりません。  
臨床応用されている脳波計は †千万単位の金†くらいしか要りません。  
もしお金がなくても、病院にはそれなりにある機械です。  
読者の中で個人的に買える人が居るなら買うと良いんじゃないかな。  
あと、最近は 10 万円くらいで買える脳波計もありますね。openBCI<sup>24</sup>とか。  
性能はまあ…？

## コンピュータの準備

必要な性能はどこまでやるかにもよります。  
センサーベース脳波解析なら普通の市販のノートでも十分です。  
MRI やソースベース解析やるなら高性能なのがいいです。  
また、高性能でも 24 時間計算し続けるような場合ノートではダメです。  
ゲーミングノートで数時間の処理とかならありかな、とは思いますが。  
一日以上やる場合は…小さくてもデスクトップ機を使って下さい。  
ノートは性能に限界があるだけでなく、排熱機構が弱いので  
数日計算し続けると火災が発生する可能性があります。<sup>25</sup>  
メモリいっぱい、CPU は多コアがいいです。今時ブランド選ばなければ割と安いです。  
どの程度のものが必要かは実験系によります。

GPU は基本「あまり効かない」と思っていて下さい。(効く場面もありますが限定的です。)  
入れるなら nVidia 製にして下さい。AMD も頑張ってますが、まだまだ科学計算に弱いです。  
メモリが大量に必要で、GPU より CPU 使う場面が多いです。  
freesurfer は OS や CPU が変わったら結果が変わるという仕様がありますから  
「このコンピュータを使う」と固定する必要があります。

---

<sup>24</sup>ハードウェア、ソフトウェア共にオープンソースという夢広がりがちな脳波計なのですが、使ったこと無いのでどの程度の性能なのかよく知りません。レビュー求む！

<sup>25</sup>あくまで本番環境ではの話です。例えばノートを通してサーバーやワークステーションを動かすとか、スクリプトの雛形を作るという用途であればソースベース解析でもノートは実用性に優れています。

---

## 導入編

### OS の準備

OS というか、コンピュータ自体ですけども。

OS は Linux, MAC OS, Windows どれでもいけます。

MRI をいじるならば Unix 系が必要になります。

それぞれ各論を言いますが、僕個人としては Ubuntu linux が好みです。

### Windows

Windows を知らない人はいないでしょう。

昔は Windows は辛かったのですが、最近は WSL という Windows 上で Linux を動かす仕組みが整っていて、一応こなせます。

MRI をやる時には性能面では劣るかも？

機種を選べば論吉が減りにくいというメリットがあります。

### MacOS

言わずと知れたオシャレ系 OS です。

何かとお財布に優しくないので玉に瑕。

こいつは Unix 系の中でも BSD の系譜であり、

MRI 研究に素の状態で使用できるのです。

### Linux(オススメ)

Unix 系の中でも Linux の系譜に当たる Linux です。

無料なので Windows 機を買ってきて両方使えるようにしちゃうと

ひと粒で二度美味しいです。(デメリットもあります)

さらに、割と素直に全てのソフトが動くのが Linux です。

さて、Linux は単体では使用できません。

Linux 単体だと、マジで黒い画面が表示されるだけで何も出来ません。

なので、ソフトの寄せ集めであるディストリビューションが必要です。

生肉に対する定食みたいなものですね。

この定食的なものが無料で配布されていたりします。

---

この中でも王手として Debian 系と Redhat 系があります。

Redhat 系は最近 CentOS 事件もあって、今は Debian 系が無難と思います。

というわけで、僕は新しめの debian 系 linux ディストリビューションである UBUNTU<sup>26</sup>を使います。

Linux でも新しめのメジャーな Linux ディストリビューションがいいです。

CUDA 等の技術に対応していたり、ユーザーが難しいことを考えなくて良いことが多いからです。

debian 系を使う理由はパッケージ管理ソフトの apt が優秀でユーザーが多いことです。

MAC の場合は apt の代わりに homebrew [https://brew.sh/index\\_ja.html](https://brew.sh/index_ja.html) を用いることになります。

以下、UBUNTU16.04LTS 以上か macos10.12 を想定して書いていきます。

UBUNTU は下記サイトから無料でダウンロードできます。

Ubuntu <https://www.ubuntulinux.jp/ubuntu>

僕自身は少しでも速く処理して欲しいので、誤差範囲かも知れませんが linux では軽量デスクトップ環境に変えています...ここは任意です。

---

<sup>26</sup>UBUNTU は Canonical 社によって開発されているオープンソースの linux ディストリビューションであり、人気があります。debian というディストリビューションをベースに作られています。ちなみに、名前は南アフリカの言葉で友愛という意味だそうです。



---

開発環境の準備（脳波、脳磁図の場合。出来るだけリッチに。）

- freesurfer だけ使う人は開発環境は要りません。読み飛ばして下さい。
- 試すだけだとか、質素な開発環境でいい人も読み飛ばして下さい。
- 玄人も読み飛ばして下さい。

プログラミング得意な人はこのセクションは見なくていいです。

開発環境は MNE 使うなら必要です。

MNE は python で動くため、まずは python の環境を用意してあげないといけないのです。

python の環境は構築の方法がいくらでもあるのですが、それぞれ一長一短です。

詳しい人からは「docker<sup>27</sup>じゃダメなん？」という質問が

来そうですが、セットアップは自分でできなければ困ることもあります。

普通は anaconda<sup>28</sup>を使います。何故なら MNE 公式が anaconda 推しだからです。

anaconda もいい所悪い所色々あってエンジニアの中には毛嫌いする向きもあります。

僕はより pythonista っぽく venv<sup>29</sup>を使っています。

pipenv<sup>30</sup>や poetry<sup>31</sup>を使う方法もあります。

venv と poetry は python 本体までは用意してくれないので、

pyenv と併用する必要があります。

Anaconda <https://www.continuum.io/downloads>

このサイトからインストールプログラムをダウンロードします。

anaconda に jupyter という repl<sup>32</sup>と spyder という IDE<sup>33</sup>が付いてきます。

これらを使うのもまたいいと思います。

それぞれの使用感

あとは、自分の好きな開発環境を調達するといいでしょ。

開発環境は色々あるので軽く紹介します。

---

<sup>27</sup>最近流行りの仮想化環境です。性能が高いのが特徴ですが、反面使いこなすのには力が必要です。

<sup>28</sup>anaconda とは ContinuumAnalytics 社で開発されているデータサイエンティスト向けの python ディストリビューションです。もちろん好みに応じて pipenv などを使うのもありますが、面倒いので書きません。

<sup>29</sup>python 標準の仮想環境です。非常にシンプルで、シンプルである故にトラブルが少ないですが、少し煩雑です。

<sup>30</sup>昔 python 公式がオヌヌメしていた仮想環境作りツールです。かなりリッチです。重いのとトラブリやすいのが欠点です。

<sup>31</sup>pipenv の代替。pipenv が python 界でメジャーになったのですが、割と不安定なところもあったので、開発されたやつです。速くて安定していて、控えめに言って神なのですが大げさかも？

<sup>32</sup>特定言語用の対話型インターフェイスのこと。

<sup>33</sup>統合開発環境のこと。

---

**vscode** vscode は現代的なテキストエディタです。

python 用ではありませんが、プラグインを入れて python の IDE として使うことが出来ます。  
Microsoft 製ですので圧倒的な安心感があります。

**spyder** とても素直な挙動の IDE で ipython の補完機能も手伝って使いやすいです。  
ただし、動作が重めなのと、企業のバックアップがなくなって今後が辛いかもです。  
オススメ度は将来性を考慮するとやや低いです。

**jupyter, ipython** repl というか、shell と言うかちょっと珍しい開発環境です。  
これだけで完結することも出来なくはないレベルの開発環境です。強みとしては

- web ベースなので遠隔操作可能
- cython や R といった他言語との連携が容易
- 対話的インターフェイスがメイン

弱みもあります

- 厳格なコーディングに不向き
- バージョン管理できない
- 気をつけてないと散らかり過ぎて崩壊する
- 処理速度が遅くなる

これだけでやろうとするのはやめたほうが良いです。

弱みが割と致命的になりがちで、初学者がこれだけに慣れると  
コーディング技術が伸び悩みます。

使うなら、必ず他のテキストエディタと併用しましょう。

僕の今のおすすめは？

初心者の場合

- anaconda
- vscode
- jupyter

という組み合わせです。mne のインストールは anaconda に任せちゃいます。

基本は vscode でスクリプトを書きますが、状況に応じて jupyter でチェックしたりします。

python を書いたことは無いけれど他の言語の玄人の人にはこうです。

- venv または poetry と pyenv
- 好みのテキストエディタ

---

え？ 僕ですか？ 僕は `venv` と `vim`<sup>34</sup> と `ipython` の CUI でやっています。

逆にお勧めしないもの

Windows の `notepad.exe` はお勧めしません！  
力不足です！

Mac のテキストエディットは最悪です！  
勝手に貴方の書いた文字列を書き換えてしまいます！

`jupyter` 一本で挑むのはやめて下さい！  
初心者は `jupyter` 依存症になって伸び悩みます！

構築

では、上記の初心者用環境を整える為の準備をしていきましょう。  
`vscode` はまあ、導入するのは楽勝なのでググって下さい。

---

<sup>34</sup> 圧倒的なテキストエディタの一つ。僕は脳波解析も、同人誌執筆も、すべて `vim` で行っています。

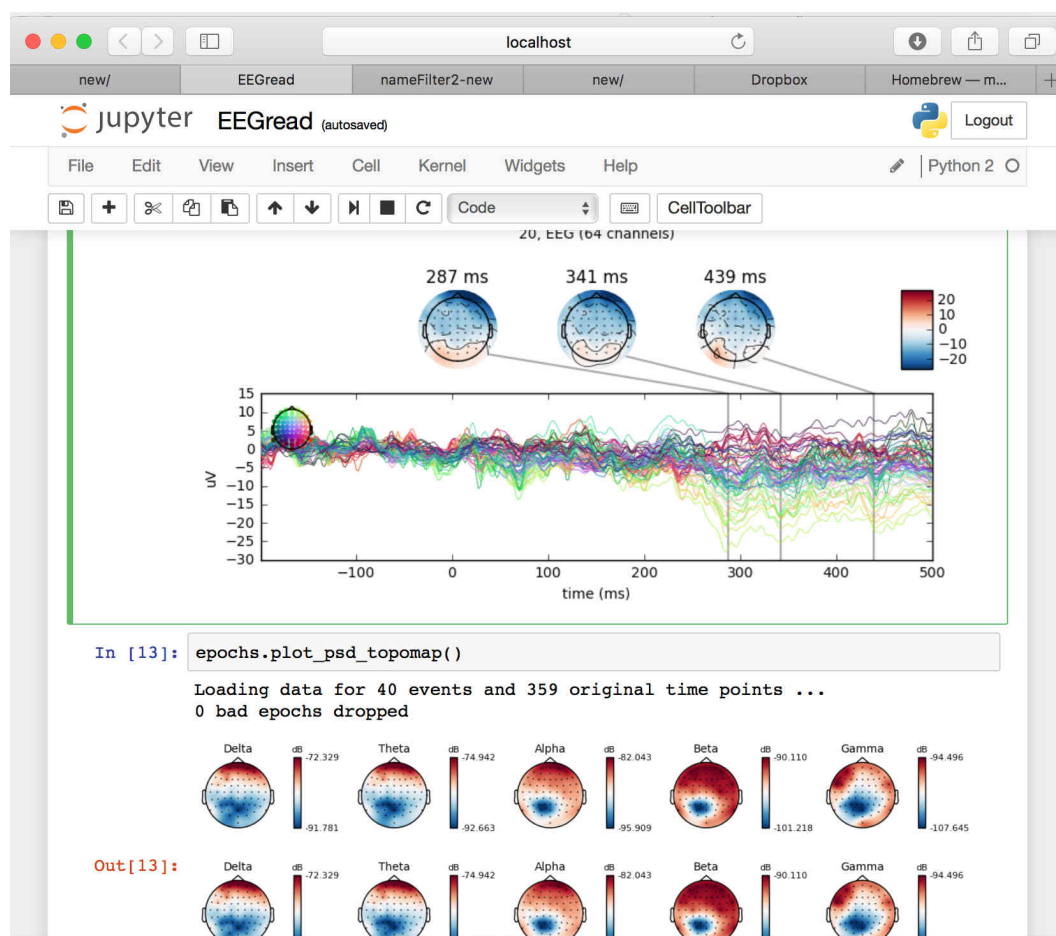


Figure 4: jupyter の画面。web ベースでインタラクティブにコーディング・共有できる。まあ、触ってみればわかります。

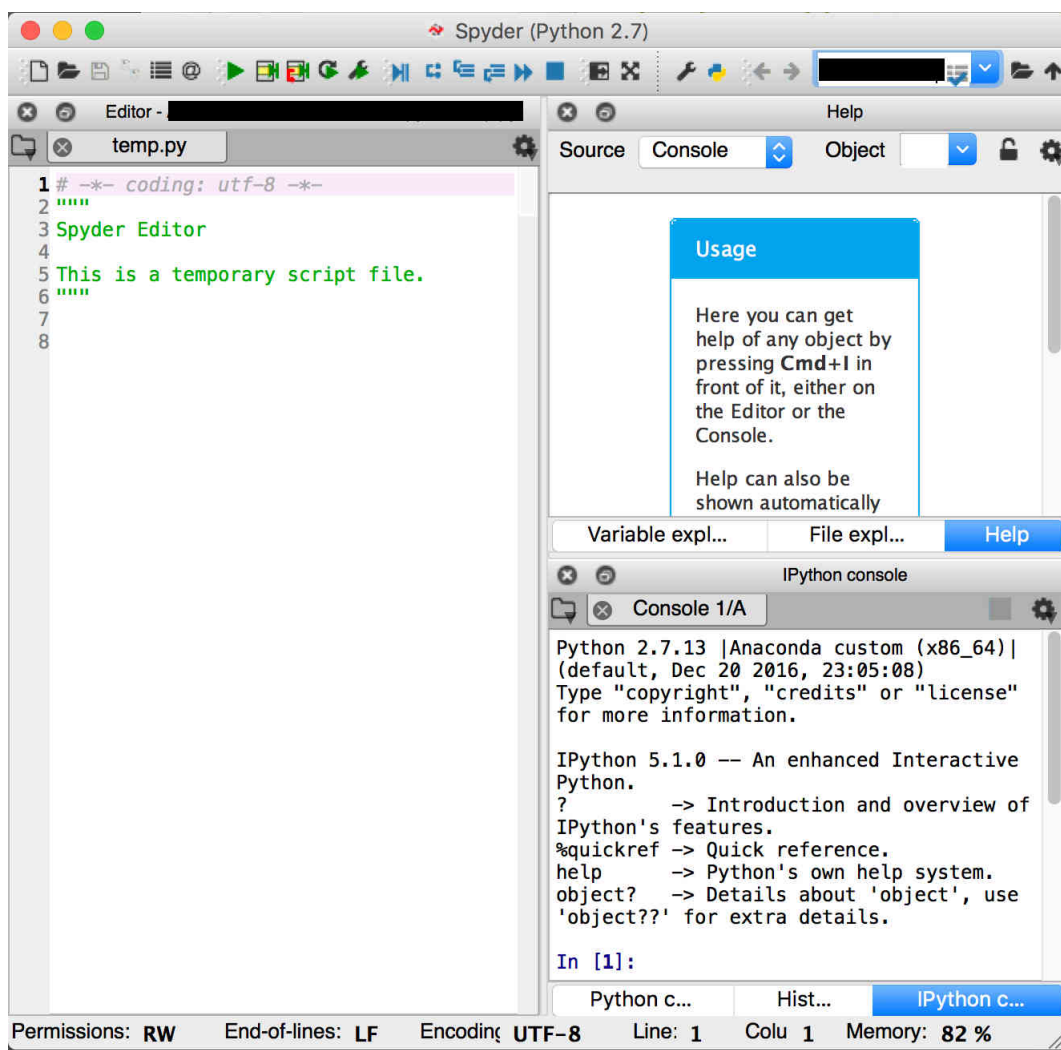


Figure 5: spyder の画面。ごく普通の素直な挙動の IDE。

MAC は anaconda のインストーラーをダウンロードしてクリックしていけばどうにかかります。  
linux では anaconda はダウンロード後、ターミナルで以下のようにコマンドを叩いてインストールします。bash です。ただの sh じゃインストールできません。

```
1 bash Anaconda3-hoge-Linux-x86_64.sh
```

インストール先はホームフォルダでいいとか、色々質問が出てきますが、そのままホームフォルダにインストールするのが気持ち悪くてもスムーズに行くかと思います。気持ち悪くて死ぬ人は pipenv でも使って下さい。

---

## Jupyter の起動

```
1 jupyter notebook
```

すると、ブラウザが起動し、画面が表示されるはずです。

起動しなければ、下記 URL にアクセスしてください。

`http://localhost:8888`

jupyter はブラウザで動かすものですが、別にネットに繋がるものじゃないです。

ちなみに、下記の様にして起動すると、lan 内で別の jupyter に接続できます。

```
1 jupyter notebook --ip hoge
```

ip は、多分ターミナルに表示されていると思います。

そうじゃないなら、例えば Ubuntu なら

```
1 ip addr
```

で表示されます。

jupyter はターミナルで `ctr-c` を二回叩けば終了できます。

では、左上の new ボタンから python を起動しましょう。

### jupyter の設定

jupyter を使うなら、折角なので拡張しておきましょう。ターミナルで下記を叩いてください。

```
1 conda install -c conda-forge jupyter_contrib_nbextensions
2 jupyter contrib nbextension install --user
3 ipcluster nbextension enable --user
```

これで extension が使えるようになります。

## jupyter での plot

jupyter は plot の方法を指定できます。

表示したい場合は、予め下記コードを jupyter 上を書いておいてください。

jupyter 上に直接出力したい時

```
1 %matplotlib inline
```

python3 環境下で別ウィンドウで表示したい時 python3 と python2 は使う qt のバージョンが違うので qt5 が必要になります。

```
1 %matplotlib qt5
```

---

三次元画像をグリグリ動かしながら見たい時  
(mayavi 使用)

```
1 %gui qt
```

これについては後でまた詳しく記載します。

## anaconda 仮想環境

anaconda は仮想環境を作れます。仮想環境はガンガン使いましょう。

何故かって？ mne のバージョンが上がることがあります。

バージョンが上がるときに困るのは、バージョンを上げると

過去の解析環境が失われてしまい、再現性が損なわれることです。

そこで、大事なのは仮想環境を作り、その環境の中でやっていくことです。

MNE は anaconda を推奨しています。

anaconda は python の仮想環境<sup>35</sup>を作ることが出来ますのでそれを利用するのが楽です。

では、ipython からやっていきましょう。

ここでは、hoge という名前の python3.6 環境を jupyter 上に作ってみましょう。

```
1 ipython kernel install --user --name py3
2 conda create -n hoge python=3.6 anaconda
3 conda activate hoge
4 ipython kernel install --user --name hoge
5 conda info -e
```

1 行目から順に何をやっているか述べます。

1. 今の python の環境を jupyter に「py3」という名前で載せておく
2. conda で別バージョンの python 環境を作る
3. 切り替える
4. jupyter に「hoge」という名前で組み込む
5. 確認

conda activate コマンドで python の環境を切り替えられます。

これで jupyter で色んな環境を切り替えられると思います。

ちなみに間違えて環境を作った場合は以下のコマンドで消せます。

```
1 conda remove -n python3 --all
```

---

<sup>35</sup>仮想環境にも色々あります。例えば、pipenv、poetry などです。anaconda も同じ様な感じで使えます。

---

## R を jupyter で動かすために

anaconda を使っているなら下記で R がインストールできます。

```
1 conda install libiconv
2 conda install -c r r-essentials
3 conda install -c r rpy2
```

これにより R が動くようになり、貴方は少しだけ楽になります。  
何故なら、実験結果を同じ環境で動く R に吸い込ませられるので、  
「実験結果を入力するだけでワンクリックで統計解析結果まで出る」<sup>36</sup>ような  
スクリプトが実現できるからです。具体的には jupyter 上で

```
1 %load_ext rpy2.ipython
```

とした後

```
1 %%R -i input -o output
2 hogehoge
```

という風に記述すれば hogehoge が R として動きます。plot も出来るし、引数、返り値も  
上述のとおり直感的です。さて、この -i ですが、通常の数字や一次元配列は普通に入りますが、  
R ならデータフレームからやりたいものです。その場合は pandas というモジュールを使って  
受け渡しをします。例えばこのような感じです。

```
1 import pandas as pd
2 data=pd.DataFrame([二次元配列])
```

```
1 %%R -i data
2 print(summary(data))
```

python と R をシームレスに使いこなすことがこれで出来るようになります。

---

<sup>36</sup>同様に、matlab や C 等と連携をすることが簡単なのが jupyter の強みの一つと思います。



---

## CUDA

CUDAをご存知でしょうか？

GPUを科学計算に用いる方法の1つで、Nvidia社が開発しているものです。

GPGPUと呼ばれる技術の一種ですね。これはMNEpythonでも使うことが出来るので、やってみましょう。

つっても、今の所フィルター関連だけなんですけどね...

しかもあまり速くない上に、ドライバ入れるのが結構面倒いとか

そういうツラミが出てきていてあまりおすすめではないです。

あと、ついにMacでNvidiaがサポートされなくなったりしています。

このインストールも詰まるとそれなりに面倒です。

結構環境によるので「頑張ってね」としか...

まずは、Nvidiaのサイトからインストーラーをダウンロードします。

Nvidia <https://developer.nvidia.com/cuda-downloads>

このサイトには色々なOSに対応したCUDAが置いてあります。

僕はubuntuならdeb(network)をお勧めします。面倒臭さが低いです。

インストーラーをダウンロードしてダブルクリックするだけではダメで、

ダウンロードのリンクの下にある説明文を刮目して読みましょう。

こんな感じに書いてあります(バージョンによって違います)

```
1 sudo dpkg -i cuda-repo-ubuntu1604_9.1.85-1_amd64.deb
2 sudo apt-key adv --fetch-keys http://hoge.hoge.pub
3 sudo apt-get update
4 sudo apt-get install cuda
```

こんな感じのがあるはずなので、実行して下さい。もちろん、バージョンとかは読み替えるべし。

そして、これが大事なのですが、bashrcにパスを通す必要があります。

これはCUDAのインストールガイドに書いてあります。

インストールガイドへのリンクは先程の説明の下に小さく書いてあります。

具体的には下記のような感じです。

```
1 export PATH=/usr/local/cuda-9.1/bin${PATH:+:${PATH}}
2 export LD_LIBRARY_PATH=/usr/local/cuda-9.1/lib64\
3     ${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}
```

これでCUDAへのリンクが貼れたはずですが、bashを再起動しましょう。

MNEpythonのCUDAインストールのページに従ってコマンドを叩きます。

[http://martinos.org/mne/stable/advanced\\_setup.html#advanced-setup](http://martinos.org/mne/stable/advanced_setup.html#advanced-setup)

```
1 sudo apt-get install nvidia-cuda-dev nvidia-modprobe
```

---

```
2 git clone http://git.tiker.net/trees/pycuda.git
3 cd pycuda
4 ./configure.py --cuda-enable-gl
5 git submodule update --init
6 make -j 4
7 python setup.py install
8 cd ..
9 git clone https://github.com/lebedov/scikit-cuda.git
10 cd scikit-cuda
11 python setup.py install
```

これでインストールできてたら成功です。python で

```
1 import mne
2 mne.cuda.init_cuda()
```

としたら Enabling CUDA with 1.55 GB available memory...  
的なメッセージが出たりします。そして、一番確実なのは MNEpython に付属した  
テストツールを回してみることです。

```
1 pytest test_filter.py
```

このテストツールは MNEpython の中にあります。  
場所的には anaconda の中の lib/python3/site-package/mne/tests  
的な場所にあると思うのですが、環境によって違うかもです。  
このテストがエラーを吐かなければ...おめでとうございます！  
貴方は MNEpython を CUDA で回すことが出来ます！ つっても、今の所フィルター関連だけなんですけどね...

---

## バージョン管理 git

バージョン管理を知っているでしょうか？

貴方はスクリプトを書くことになるのですが、ちょっとしたミスでスクリプトは動かなくなります。

そんなリスクを軽減するために、貴方はスクリプトのコピーを取ります。

コピーを取り続けるうちに、貴方のコンピュータはスクリプトで埋め尽くされ、収集つかなくなります。

さらに、他の人がスクリプトを手直する時、引き継ぎとかも大変です。

そんな貴方は git を使うと幸せに成れます。

git を知らない人は、とりあえず github desktop とか source tree をダウンロードして

体でそれを知ってください。詳しくは git でググってください。

こことか参考になります。

git-guide <http://www.backlog.jp/git-guide/>

## git サーバー

git 単体でもいけるのですが、git サーバーというのもあります。

最近 Microsoft が気前よく github のプライベートリポジトリを無料化したので、

それを使うのもいいでしょうね。

自分の研究用スクリプトをアップしたくないなら自前で鯖立てするのもいいし、

そもそも鯖立てなくても十分便利です。

ただ、一つ言えるのは、これ間違って public として個人情報を

github に上げちゃったりすると捕まりますので、これだけは注意しましょう。

普段は git サーバーは要らないと思うよ。

## jupyter で作ったスクリプトのバージョン管理 (小技)

jupyter のファイルは git しにくい上にすごい散らかるので

きちんとコーディングする場合はオススメしません。

普通にテキストエディタ使いましょう。

あくまでサブとして使う事をおすすめします。

重いしね...

### 方法 1(超面倒くさい)

```
1 jupyter notebook --generate-config
```

このコマンドで jupyter のコンフィグファイルが作成されます。

場所は/home/user/.jupyter です。

---

その上で、下記 URL に記載されている通りに書き加えます。

<http://jupyter-notebook.readthedocs.io/en/latest/extending/savehooks.html>

すると、jupyter で編集したファイルが python のスクリプトとしても保存されます。

あとは git<sup>37</sup>などで管理すればいいです。

ただし、この方法は計算結果がファイル内に残りません。

しかも散らかりますし、面倒くさいです。

方法 2(あまりおすすめしない) gitを使いますが、git 側の設定だけでもどうにかかります。

まず、jq をインストールします。.gitattibute に書きを書き加えます。無ければ作ってください。

```
1 *.ipynb diff=ipynb
```

そして、下記を.git/config に

```
1 [diff "ipynb"]
2 textconv=jq -r .cells[] |{source,cell_type}
3 prompt = false
```

下記を.gitignore に

```
1 .ipynb_checkpoints/
```

これで jupyter notebook のファイルを git で管理しやすくなります。

色んな理由でおすすめはしませんけどね...

---

<sup>37</sup>プログラミング用バージョン管理ソフト。敷居は高いが多機能で超速。GUI クライアントも豊富。

---

## maxfilter(elekta のやつ)

maxfilter というフィルタが MEG 研究ではほぼ必須です。

これは外から飛んでくるノイズを数学的に除去するフィルタです。

これについては MNEpython にもあるのですが、elekta 社の maxfilter もあります。

一長一短ですが、何も考えずに使うなら elekta 社でしょうか…。

僕は以前は elekta のを使っていましたが、最近 MNE に移行しました。MNE のは後で解説します。

それぞれの特徴としては

Elekta 版

- Elekta 社の MEG 部門が売却されたよ…将来性どうなん？
- Redhat 系 linux でないと動かないのがクソ (Docker 使うところかな？)
- 自動で bad チャンネル見つけてくれるのが超最高

MNE 版

- 臨床には使っちゃいけないという縛りあり
- 使用環境を選ばないのが超最高
- まだ改良中？？？

ELEKTA 社を使いたい場合は DANA というソフトと

maxfilter というソフトを ELEKTA 社から貰う必要があります。

また、環境は Redhat5 または CentOs5 の 64bit 版を使うことになっています。

えっ、linux は Ubuntu を使うんじゃないの？ と思ったでしょう？

ELEKTA 社のソフトは redhat linux 系が前提なのです。

あとは企業の人に聞いたほうが良いです。細則があります。

…正直、よく分かんないし、あんまり語れないんですよね、

オープンソースじゃないものですね。

---

## freesurfer のインストール

freesurfer をインストールしましょう。

下記の url からダウンロードできます。

<https://surfer.nmr.mgh.harvard.edu/fswiki/rel7downloads>

Ubuntu、CentOS、MACOS がありますね！

Windows 版？ ハッw そんなものはない。

もしどうしても君が Windows を使わねばならぬなら

WSL(Windows Subsystem for Linux) を使えばいいかもしれないです。

というか、僕の手元では動いたので動くと思います。

Mac なら下記のソフトも必要です。Xwindow システムですね。

<https://www.xquartz.org/>

素で Xwindow システムが動くのが Linux ディストリビューションなので、  
ここが僕が Linux ディストリビューションを気に入っている理由です。

Ubuntu なら

```
1 sudo dpkg -i hoge
```

でパッケージをインストールできます。

Mac なら GUI のインストーラーあります！

ね、簡単でしょう？ でも、まだ終わっていません。このままでは動きません。

もしかすると、追加で下記が必要かも知れません。Ubuntu なら

```
1 sudo apt install tcsh csh
```

Mac なら

```
1 sudo brew install tcsh csh
```

あとは、設定をしないとイケないのです。

設定ファイルはホームディレクトリにある隠しファイルです。

テキストエディタは何でも良いですが、とにかく編集しましょう。

「隠しファイルなにそれ」な人は、unix 系の勉強をしましょう！

僕はとても優しいので教えますが、「.」で始まるファイル名は隠しファイルになります。

freesurfer のダウンロードページに、Setup & Configuration という所があります。

四角で囲んである部分をコピーして、隠しファイルの .bash\_profile に追記しましょう。

場合によっては .bashrc のこともあるかも知れませんね。

---

貴方が使っているシェルに応じてどれをどんな風にコピペするかが決まるのですが大抵は bash か zsh と思います。

linux なら

[https://surfer.nmr.mgh.harvard.edu/fswiki//FS7\\_linux](https://surfer.nmr.mgh.harvard.edu/fswiki//FS7_linux)

Mac なら

[https://surfer.nmr.mgh.harvard.edu/fswiki//FS7\\_mac](https://surfer.nmr.mgh.harvard.edu/fswiki//FS7_mac)

を見て下さい。

要するにこんなふうなのを設定ファイルに書き加えるんですね。

```
1 export FREESURFER_HOME=$HOME/hoge
2 source $FREESURFER_HOME/SetUpFreeSurfer.sh
```

1 行目を見て下さい。

これは Freesurfer がホームディレクトリの 'hoge' にインストールされている場合ですね。

で、コピペし終わったら、保存して閉じるんですが、MRI の解析結果の保存先 (subject\_dir) を決めてあげたい場合は下記のようにします。

```
1 export SUBJECTS_DIR=fuga
```

これは決めてあげたほうが良いです。何故なら、標準の subject\_dir は読み書きに管理者権限が必要だったりするからです。

あと、毎回同じ MRI 研究をするわけでもないので、プロトコルが変わるなら場所も変えてあげたいですよ！

最後にライセンスキーを入れましょう。

freesurfer の公式サイトに登録して、ライセンスキーをメールでもらい、freesurfer のディレクトリに突っ込みます。

面倒いので、あとは freesurfer のサイトを読んで下さい。

---

## MNE/python のインストール (脳波、脳磁図をする場合)

こちらは MNE の公式では anaconda の存在下でやるようになっています。  
anaconda が嫌いな人 (結構いらっしゃるかと思います) は既に十分な知識をお持ちのことと思います。

例えば、下記の意味が分かる人にはこのセクションは不要です。  
あとはその都度必要になったものを適当に入れます。

```
1 python -m venv env
2 source env/bin/activate
3 pip install mne h5io matplotlib numpy mne-connectivity
```

ちなみに、意味的には

「env って名前の仮想環境を作って、その仮想環境に行って、色々インストールするよー」という意味です。

実は僕はいつもこの方法をとっています。

ま、python は何通りも導入方法あるので。

あと、最近 mne からコネクティビティ解析関連が mne-connectivity として独立したので、使いたいならば入れて下さい。

### 公式のインストール方法

長く mne-python をヲチしているけれど、  
mne-python のインストール方法、変わりすぎて草。  
バージョン変わる毎にインストール方法変わるので注意です。  
現行バージョンは 1.4。

現時点で公式サイトでは anaconda を使うことになっています。  
理由はよく分かりませんが、多分「どこでも動くから説明が楽」なんだと思います。  
インストール方法はここにあります。

[https://mne.tools/stable/install/manual\\_install.html](https://mne.tools/stable/install/manual_install.html)

ちなみにこういうゆるふわ簡易版なのもあります。

<https://mne.tools/stable/install/installers.html#installers>

さて、上記の簡単版を使わないなら、下記のようにします。

```
1 conda install --channel=conda-forge --name=base mamba
2 mamba create --override-channels --channel=conda-forge --name=mne mne
```



---

これで新しい'mne' という仮想環境が出来るようです。

そう、仮想環境で構築することになります。

このやり方のメリットは、いつでも同じ環境を整える事ができるので、

ソフトのバージョンが変わっても対応しやすいということです。

反面、毎回仮想環境に入らないといけないという小さなデメリットがあります。

だけど、バージョン揃えるのは貴方の責任です。

つまり、使いましょう。(圧力)

公式サイトをみながら頑張りましょう。

```
1 conda create\  
2     --strict-channel-priority\  
3     --channel=conda-forge\  
4     --name=mne\  
5     mne-base h5io h5py pymatreader
```

-name=mne がありますよね？

これで mne という名前の仮想環境が整います。

下記のコマンドで mne の環境に入れます。

```
1 conda activate mne
```

今後は mne を使うときは必ず上記のコマンドを打って下さい。<sup>38</sup>

面倒くさい？ どうしても打ちたくないです？

それならば、.bashrc や .bash\_profile に下記を追記してください。

```
1 conda activate mne
```

大まかにはこれで完結です。

仮想環境は複数作っておくほうが良いと思います。

hoge って環境作りますね。

```
1 conda create\  
2     --strict-channel-priority\  
3     --channel=conda-forge\  
4     --name=hoge\  
5     mne-base h5io h5py pymatreader
```

hoge の環境に入るには

```
1 conda activate hoge
```

---

<sup>38</sup>昔は source activate コマンドでしたが、このコマンドは anaconda 以外の仮想環境ツールと衝突してクラッシュするという不具合がありました。控えめに言って糞仕様ですね。今後は conda activate コマンドを使うのがいいでしょう。

---

です。ちなみに、出るのは

```
1 conda deactivate
```

mac なら下記も必要です。

## MAYAVI がインストール出来ない

mayavi は mne-python のインストールの鬼門です。

僕はしばしば mayavi をインストールすることに失敗します。

mayavi のドキュメントに「mayavi」のインストールなんて簡単とか書いてありますが騙して悪いがあれは嘘である。

pip でインストールできないときのとっておきをお伝えします。

ようこそ、UNIX の世界へ。

まずは、git をインストールします。

git のインストールは mac なら

```
1 brew install git
```

ubuntu なら

```
1 sudo apt install git
```

で簡単です。そのうえで、mayavi のソースコードを貰ってきます。

```
1 git clone https://github.com/enthought/mayavi.git
```

で、この中の setup.py を走らせましょう。

```
1 cd mayavi
2 python setup.py install
```

ご注意ください。問答無用で今使っている python の中に入ってしまうです。

もし、これでうまく行かないならば、なにかコンパイラとかが欠けているはずです。

例えば ubuntu なら

```
1 sudo apt install build-essential
```

とかやれば入るかも？（検証してません）

---

## HF5 をインストールしたい

時間周波数解析をする場合は、  
HF5 をインストールする必要があることがあります。  
上記の anaconda の方法なら全部入りますが、pip とかでやると  
入らないので一応言います。

```
1 pip install h5io
```

でおkです。

## jupyter kernel

jupyter を使うのであれば、上記の環境を jupyter に登録する必要があります。  
まずは、仮想環境に入って下さい。

```
1 conda activate mne
```

では、登録しましょう。下記は「今いる環境を jupyter に登録する」やつです。  
user というのは「コンピュータ全体向けじゃなくて、僕向けにやるよ」  
name は仮想環境の名前ですね。

```
1 jupyter kernel install --user --name hoge
```

もし、要らなくなったら

```
1 jupyter kernelspec uninstall hoge
```

ですね。

## CUDA

CUDA<sup>39</sup> (GPGPU) についてもそのサイトに記載があります。  
CUDA は nvidia の GPU しか動きません。インストールについては nvidia のサイトも参照して下さい。  
まあ、各種波形フィルタでしか使えない上にさほど性能よくないです。  
正直、CPU だけで十分です。<sup>40</sup>

僕の環境では下記二行のコマンドを予め入れていないと動かないです。  
.bash\_profile や .bashrc に書き加えておけばいいでしょう。

---

<sup>39</sup>nVidia の GPU を使った高速な計算ができる開発環境。うまく使いこなせれば CPU の 10 から 100 倍速いです。

<sup>40</sup>MNE python は GPU の使い方が下手くそです。

---

```
1 export LD_PRELOAD='/usr/$LIB/libstdc++.so.6'
2 export DISPLAY=:0
```

さらに、jupyter 内で下記を実行しないといけません。

```
1 %gui qt
```

## MNE/C のインストール

これは mne-python を普通に使うなら不要です。

つまり、レガシな物を使いたい人が使うやつです。

要するに不要です。

残念ながら MNE/C を使ったことがないので僕には何もわからないのです…。

下記サイトにメールアドレスを登録し、ダウンロードさせていただきます。

MNE-C [http://www.nmr.mgh.harvard.edu/martinos/userInfo/data/MNE\\_register/index.php](http://www.nmr.mgh.harvard.edu/martinos/userInfo/data/MNE_register/index.php)

ダウンロードしたものについてはこのサイトの通りにすればインストールできます。

MNE-C [http://martinos.org/mne/stable/install\\_mne\\_c.html](http://martinos.org/mne/stable/install_mne_c.html)

僕はホームディレクトリに入れました。

```
1 tar zxvf MNE-hogehoge
2 mv MNE-hogehoge MNE-C
3 cd MNE-C
4 export MNE_ROOT=/home/fuga/MNE-C
5 . $MNE_ROOT/bin/mne_setup_sh
```

これで MNE-C も動くようになるはずです。

## コラム 1-SNS の活用

- 1 皆さんは SNS はしていますか？ SNS には様々な効能と副作用があります。
- 2 時に炎上する人だって居ます。廃人になる人も居ます。
- 3 しかし、最先端の科学にとって、SNS は大変有用なのです。
- 4 twitter で MEG や MRI の研究者をフォローしてみてください。
- 5 いい情報、最新の情報がピックアップされ、エキサイティングです。
- 6 僕は新着情報は twitter で研究者、開発者、有名科学雑誌のアカウントを
- 7 フォローしてアンテナはってたこともありました。
- 8 (筆者の脳の疾患が増悪して今はしてない)
- 9 ちなみに、若いエンジニアはよくするらしいです。

---

## freесurfer を使う (MRI)

ターミナル使える人のための TLDR;  
インストールしたら以下で終わり。

```
1 recon-all -i ./hoge.nii -subject (被験者番号) -all
```

ちなみに、recon-all というのは複数のコマンドを束ねたコマンドです。  
その複数のコマンド群は下記に記載されています。

<https://surfer.nmr.mgh.harvard.edu/fswiki/ReconAllDevTable>

つまり、上記の細かいやつを参照すれば自分が何をしているのか分かるのですが、  
ユーザーが難しいことを考えなくてもソフトの作者が  
「デフォルトではこうだから全部並べとくね」ってやって下さってる。  
昔だったら自分でこれを実装しないといけなかったらしい。  
凄い親切！ recon-all ってなんて楽ちんなコマンドなんだ！  
圧倒的感謝！ 謝々！ カムサハムニダ！ 現場からは以上です。

とはいえ、ソフト自体はものすごく正確というわけではありません。  
塗り損ないが出来ることがあります。トラブルシュートは後で書きます。  
ちなみに、並列計算出来るんですが、公式曰く「バグがあるから使うな」とのこと。  
(なのに何故か無効化されていない。何故だ...)

さて、俺は優しいのでターミナル使ったことのない人への解説も書きます。  
下記はターミナルを操るための必要最低限の bash のコマンドです。

- cd : 閲覧するフォルダへ移動する
- ls : 今開いているフォルダの内容を確認する

まず、ターミナルを開き MRI の画像データがある場所まで移動します。  
例えばフォルダの名前が DATA なら下記のようにします。

```
1 cd DATA
```

辿って行って、目的のファイルを見つけたならば、freесurfer で解析します。  
例えばファイルの名前が hoge.nii なら下記です。

```
1 recon-all -i ./hoge.nii -subject (患者番号) -all
```

このコマンドを走らせると、完遂するのにおよそ丸1日かかります。

で、やっている事は、頭蓋骨を取り除き、皮質の厚さやボリュームの測定、標準脳への置き換え、  
皮質の機能別の色分け等、色々な事をしています。詳しくは freесurfer のサイトを見て下さい。

---

## recon-all 同時掛け (freesurfer)

recon-all はマルチスレッドで処理をすることができます。  
しかし、効率はあまり良くないし<sup>41</sup>、公式が使うなって言っています。  
つまり、マルチコア機なら一例ずつマルチスレッドでかけるより、  
同時多数症例をシングルプロセスで掛かける方が速く済みます。  
ターミナルを沢山開いて処理させたりすると速いですが煩雑です。  
なので、スクリプトを書いて自動化することをおすすめします。

MNEpython を使う人はプログラミングの習得は必須なので良いとして、  
freesurfer しか使わない人でもスクリプトは書けるようになる方が便利です。  
僕のおすすめは python、sh のいずれかです。<sup>42</sup>  
下記は sh の一例です。

```
1 recon-all -i ./hoge1.nii -subject hoge1 -all &
2 recon-all -i ./hoge2.nii -subject hoge2 -all &
3 recon-all -i ./hoge3.nii -subject hoge3 -all &
4 recon-all -i ./hoge4.nii -subject hoge4 -all
```

こんな感じでテキストファイルにして、'hoge.sh' とでも名付けます。  
で、以下のとおりです。

```
1 sh hoge.sh
```

「シェルで並列するなら xargs コマンドを使えよ」という声が  
聞こえてきそうですが、本当はそれが王道です。  
やりましょうか。

```
1 SUBJECTS='1
2 2
3 3
4 4'
5
6 echo "$SUBJECTS" | \
7   xargs -P 4 -I{} recon-all ./hoge{}1.nii -subject hoge{} -all
```

これで、CPU4 つを使って同時に解析できます。

---

<sup>41</sup>理由は openMP というライブラリを使った並列化だからです。openMP はマルチスレッドを簡単に実装する優れたライブラリ  
なのですが、メモリの位置が近い場合にスレッド同士がメモリ領域の取り合いをしてしまうため速度が頭落ちになるので  
す。このケースではマルチスレッドよりシングルプロセスをいっぱい並べる方が良いように思います。

<sup>42</sup>ちなみに、僕は vimmer なので vim を使って sh を直書きしています。vim でシェルを扱うときの必殺技があるのです。です  
が、本書は vim の本ではないので書きません w w w w w w w

---

## freесurfer の解析結果の表示

freeview というコマンドで解析済みの画像を表示できます。

上から解剖的に分けたデータを乗せることで部位別の表示ができます。

コマンドラインでは以下のようにすればいいですが、freeview と叩いてから画面上からやっていってもいいと思います。

(多くの人は普通の画面上からしたほうが分かりやすいでしょう)

```
1 freeview -v <subj>/mri/orig.mgz \  
2 hoge/mri/aparc+aseg.mgz:colormap=lut:opacity=0.4
```

orig.mgz というのはオリジナル画像。グレイスケールで読みこみましょう。

aparc+aseg.mgz は部位別データ。部位別データには色を付けて読み込みましょう。

画面左側に表示されているのは読み込んだ画像一覧です。

上に半透明の画像を重ねあわせていって上から見ています。

色々できますので、遊んで体で覚えるのが良いと思います。

## 解析結果のまとめ

recon-all が終わった時点で、下記コマンドを入力しましょう。

```
1 asegstats2table --subjects hoge1 hoge2 hoge3 ...\  
2 --segno hoge1 hoge2 hoge3 ... --tablefile hoge.csv
```

subject には subject(つまり解析済みデータの通し番号) を入れます。

segno には見たい位置を入力します。その位置というのは

\$FREESURFER\_HOME/FreeSurferColorLUT.txt に書かれていますので参照しましょう。

これで hoge.csv というファイルが出力されます。

このファイルの中には既に脳の各部位のボリュームや皮質の厚さ等、

知りたい情報が詰まっています。しかし、このまま使うのは危険です。

freесurfer は時にエラーを起こしますので、クオリティチェックと修正が必要です。

## 画像解析の修正

個別な修正は freeview を用いてすることになります。下記を参照して下さい。

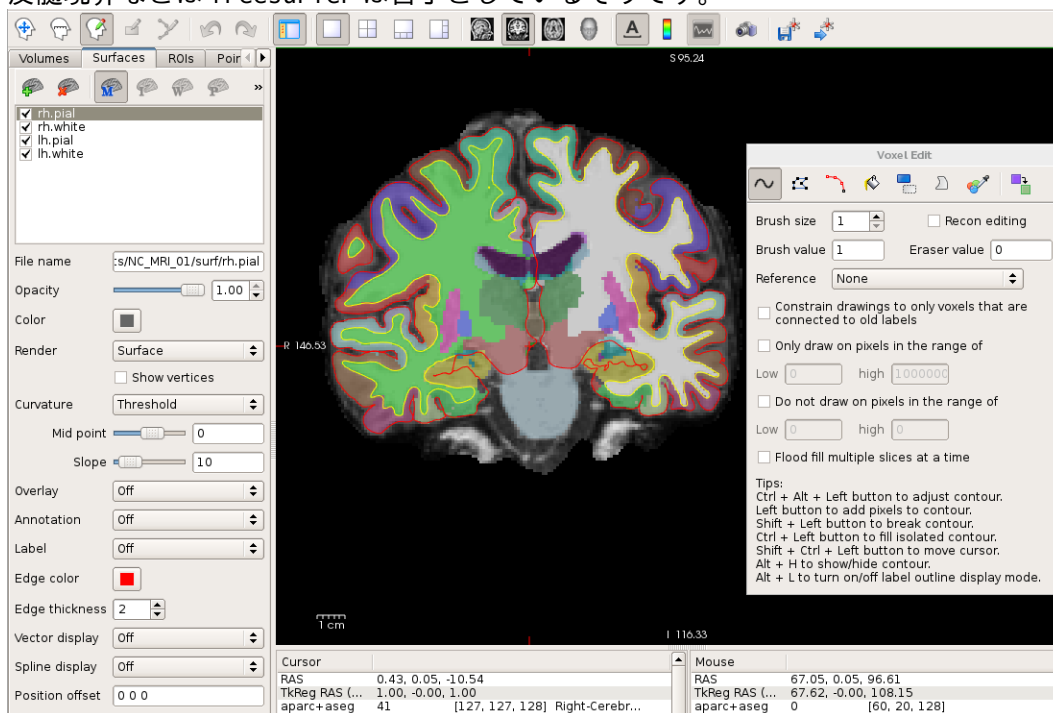
Tutorials <http://freесurfer.net/fswiki/Tutorials>

この freесurfer のサイトには、説明用のスライドと動画があり、とてもいいです。

以下、要約です。

- 脈絡叢や各種膜を灰白質と間違える
  - freeview で修正して recon-all(オプション付き)
- 白質の中で低吸収域を「脳の外側」と間違える
  - freeview で修正して recon-all(オプション付き)
- 白質の中で薄い部分を灰白質と間違える (controlpoint より小さい部分)
  - freeview で修正して recon-all(オプション付き)
- 頭蓋骨をくりぬく時に間違って小脳などを外してしまう
  - recon-all(オプション付き)
- 白質を freesurfer が少なく見すぎてしまう
  - freeview で controlpoints を付け加えて recon-all(オプション付き)

これは、問題にぶつかった時に上記サイトのスライドでも見ながら頑張るのが良いと思います。  
皮髄境界などは freesurfer は苦手としているそうです。



**SkullStrip** のエラー    Freesurfer は脳だけを解析するために Skull Strip という作業をします。要するに、頭蓋骨を



---

外してしまうわけです。この時に watershedmethod<sup>43</sup>という方法を使うのですが、頭蓋骨を切り取ろうとして脳まで取ったり逆に眼球や脈絡叢まで脳と間違えることがあるので修正が必要です。

脈絡叢の巻き込み 脈絡叢 (血管の束とか) を巻き込んでいる場合は brainmask.mgz を編集します。Brush value を 255、Eraser value を 1 にして Recon editing shift キーを押しながらマウスをクリックして脈絡叢を消していきます。編集がおわったら

```
1 recon-all -s <subject> -autorecon-pial
```

とします。

眼球が白質と間違われた時 上記と同様にして、編集がおわったら

```
1 recon-all -s <subject> -autorecon2-wm -autorecon3
```

頭蓋骨と間違って脳をえぐっているとき 頭蓋骨と間違って脳実質まで取られた画像が得られた場合は

```
1 recon-all -skullstrip -wsthresh 35 -clean-bm -no-wsgcaatlas -s <subj>
```

で調整します。この-wsthresh が watershedmethod の閾値です。標準は 25 なのですが、ここではあまり削り過ぎないように 35 にしてます。

白質の内部に灰白質があると判定されるとき 時々、白質の中の低吸収域を灰白質とか脳溝と間違えることがあります。これも freeview で編集します。

wm.mgz を開いて色を付け、半透明にし、T1 強調画像に重ねます。

Brush value を 255、Eraser value を 1 にして

Recon editing をチェックして編集します。

```
1 recon-all -autorecon2-wm -autorecon3 -subjid <hoge>
```

白質が厳しく判定されているとき 実は、freesurfer は brainmask.mgz で白質を全部 110 という色の濃さに統一します。

しかし、時々これに合わない脳があります。

そんな時は brainmask.mgz にコントロールポイントをつけて recon-all をします。

---

<sup>43</sup>脳に水を流し込むシミュレーションをすることで切っていいところと悪い所を分ける処理

---

File -> New Point Set を選びます。

Control points を選んで OK して、選ばれるべきだった白質をクリックしていきます。そして下記でいいそうです。

```
1 recon-all -s <subject> -autorecon2-cp -autorecon3
```

なんか、コマンド難しくて死にたくなるな？ 「ああ！ もう！ シェルスクリプト作っちゃえ！」と思いましたので、一筆。

```
1 recon-me () {
2   recon-all -s $1 -autorecon2-wm -autorecon3
3 }
4
5 recon-myaku () {
6   recon-all -s $1 -autorecon-pial
7 }
8
9 recon-hone () {
10  recon-all -skullstrip -wsthresh $2 -clean-bm -no-wsgcaatlas -s $1
11 }
12
13 recon-gray () {
14   recon-all -autorecon2-wm -autorecon3 -subjid $1
15 }
16
17 recon-gray () {
18   recon-all -s $1 -autorecon2-cp -autorecon3
19 }
```

これらを bashrc にでもぶち込んでおけば良いんじゃないかな？

---

## mricon/crogl(MRI を使う場合)

MRI の形式って色々あります。例えば誰かの脳の画像を手に入れて、それを解析できなければ悲しいですね？

そこで、変換ソフトが必要になります。

ここでは mricon と mricrogl を紹介しますが、

僕は後者のほうが新しいしトラブルも少ないのでいいと思います。

実は他に MRIconverter とか言うのもあるらしいですが、

僕は MRI の専門ではないのでよく分かりません…。

mricon は UBUNTU なら

```
1 sudo apt install mricon
```

MAC なら <http://www.mccauslandcenter.sc.edu/crnl/mricon/> から

インストーラーをダウンロードします。この mricon ファミリーの中にある dcm2nii というソフトが MRI の形式の変換に大変有用です。

さて、新しい mricrogl ですが、ここからダウンロード出来ます。

<http://www.mccauslandcenter.sc.edu/mricrogl/>

これはたまたま mricon では変換できないものを変換することが出来ます。

ちなみに、Debian 系 Linux は神なので、Ubuntu なら下記で大丈夫です。

```
1 sudo apt install dcm2niix
```

以上で freesurfer/MNE/python のインストールは終了しました。

これでゴリゴリ計算していくことができます。

## MRI のファイルの変換

mricon も mricrogl も mri の画像の閲覧が出来るソフトです。

ちゃんと変換の前に内容を見ましょう。

MRI って結構撮り損ないがあるものです。

この中に dcm2nii というソフトがあるはずなので、そのソフトを起動します。

mricon なら dcm2niigui、mricrogl ならメニューから import 辺りを探して下さい。

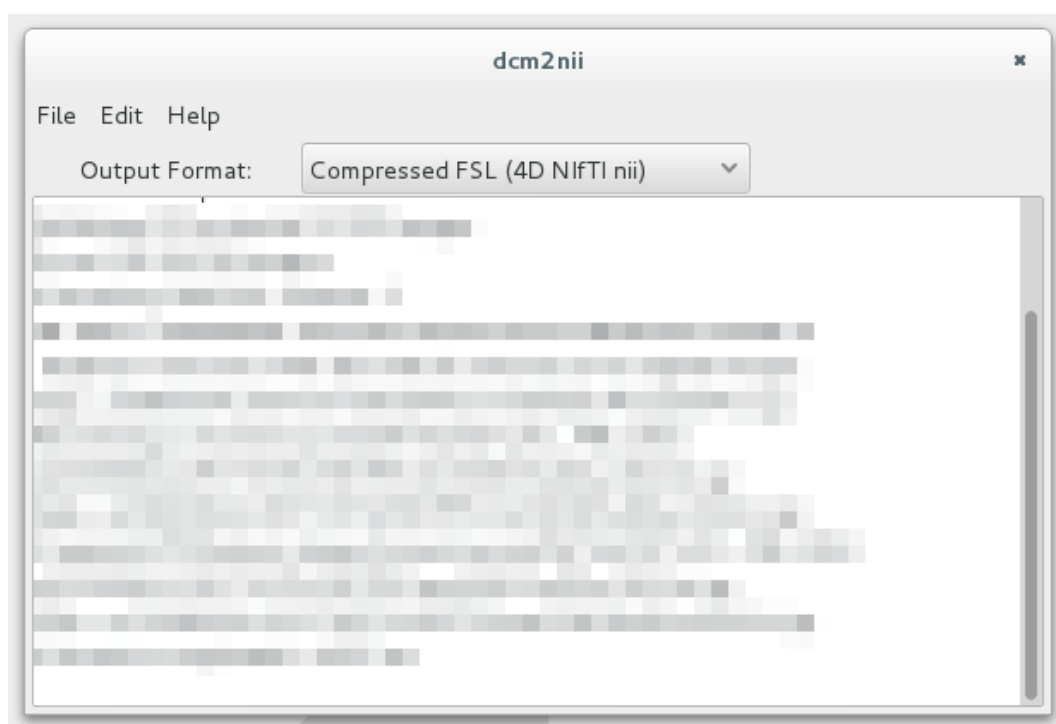


Figure 6: dcm2nii の画面

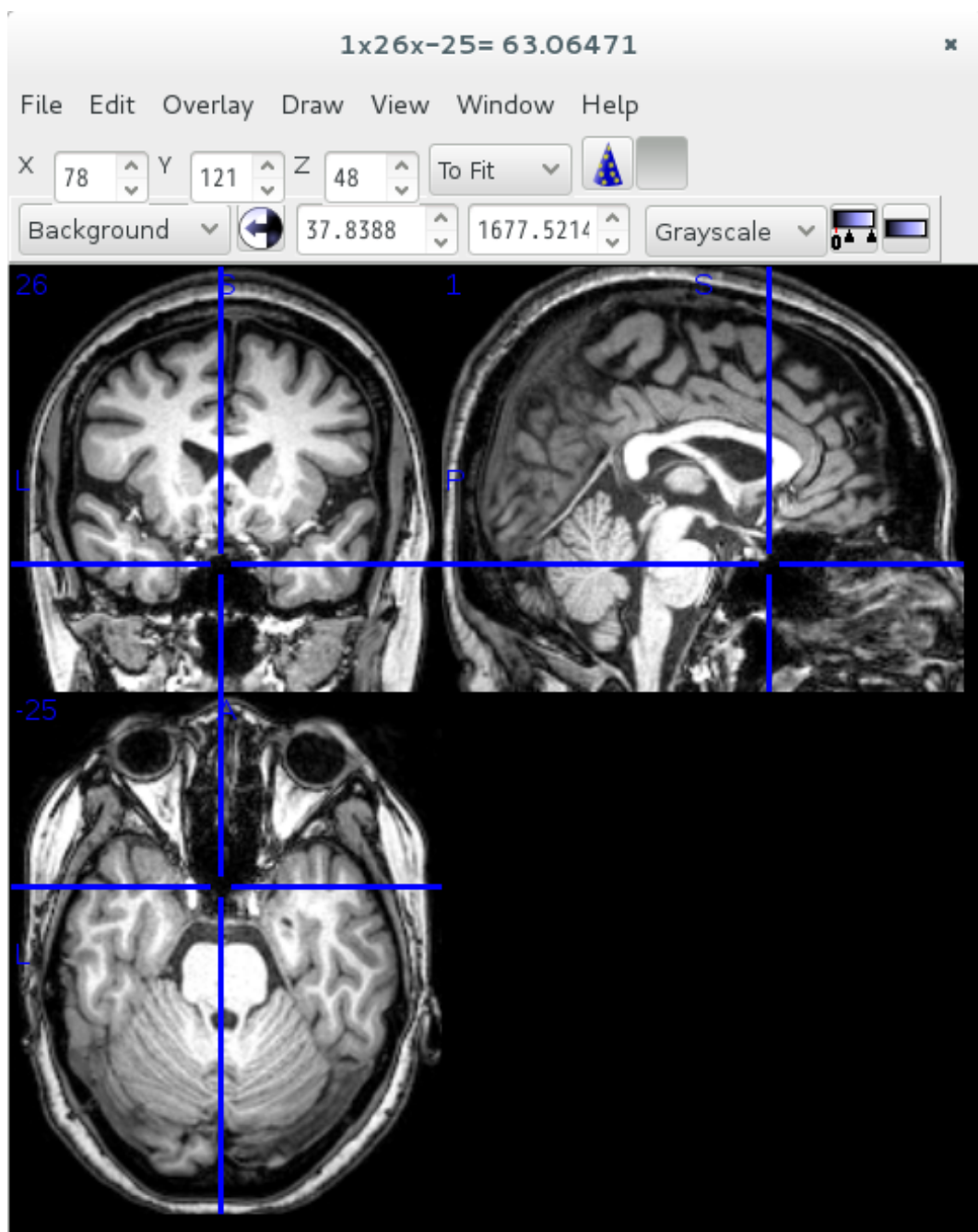


Figure 7: mricon による 3DMRI 画像の閲覧

例えば手元にある MRI の形式が dicom ならば、方言を吸収するために NIFTI 形式に直した方が僕の環境では安定していました。dcm2niigui の画面に dicom のフォルダをドラッグしてください。ファイルが出力されるはずです。

---

さて、出力されたファイルですが、3つあるはずです

- hogehoge:単純に nifti に変換された画像
- ohogehoge:水平断で切り直された画像
- cohogehoge:脳以外の不要な場所を切り取った画像

となります。どれを使っても構わないと思います。

でも、大事なことがあります。使うソフトや変換の方法は合わせて下さい  
なにやら、それぞれが微妙に違うのだそうです。MRI に詳しい人が言ってた。

---

## 解析編

いよいよ解析の準備に入ります。以下、MNE の公式サイトのチュートリアルなのですが...  
初心者が見るには敷居が高目です。要らない部分が結構あるし。

一回はそれをなぞろうと思いますが、その後は噛み砕いてシンプルに紹介します。

[https://mne.tools/stable/auto\\_tutorials/index.html](https://mne.tools/stable/auto_tutorials/index.html)

## 肩慣らし

いきなり応用は無理だよ、という人に対する補習を行います。

玄人は読み飛ばして下さい。

## MNEpython を使う前に学んでおくべきパッケージ

とりあえず、python と numpy<sup>44</sup>の基礎を学ばねばなりません。

これは最低限のことです。これを書けないのであれば mne/python は無理です。

他に学んでおくべきパッケージは matplotlib でしょう。

scikit-learn もググってください。

初心者は毎日何らかの課題に向けて python スクリプトを書きましょう。

毎日書いて下さい。大事なことなので 2 回言いました。

指が覚えます。適当にググって良いサイトを見つければいいでしょう。

Python 入門から応用までの学習サイト

<http://www.python-izm.com/>

## python を綺麗に書くために

プログラミング出来る人には釈迦に説法ですが、初心者の人に伝えたいことがあります。

python に限らずプログラミングは中々奥が深いので、

ある程度指が覚えたところで「綺麗なコード」を書かねばなりません。

なぜなら、3 ヶ月後に自分の書いたコードを読めなくなるからです。<sup>45</sup>

僕のおすすめを書きます。

- mypy (python に静的型付けを導入するもの)

---

<sup>44</sup>python 用行列計算ライブラリ。科学計算に広く用いられています。

<sup>45</sup>信じられないでしょうが、本当です。

- 
- flake8(python の補完をしてくれるもの)
  - pep8(python を書くときのコーディング規約。即ちお作法)

この辺りはエディタによって導入方法が違うので書きません。

## numpy で遊ぼう

詳しくはググって下さい。numpy は本書では語りつくせるわけがありません。以上です。

...ではあんまりなので、ほんのさわりだけ紹介しておきます。

numpy とは行列計算のための python 用パッケージです。

行列計算とは、高校数学の旧カリキュラムにおける数 III の行列です。

大学数学で皆に憎まれている、あの行列です。

目を逸しそうになった人は、行列の意味を知らねばなりません。

行列の意味は色々ありますが、ここでは

「連立方程式とは四則演算である」という意味であります。

本書に「割り算を極める」というセクションを書きましたので読むと良いでしょう。

Python の数値計算ライブラリ NumPy 入門

<http://rest-term.com/archives/2999/>

初心者の君は今思ったでしょう。「python なら list があるから良くね？」

ダメです！ numpy は C や FORTRAN で実装されており、超高速です。

多分、100 倍くらい。さらに、様々な数学的手法が実装されており、お得です。

それ以前に、MNE-python 自体が numpy で実装されているので必須です。

```
1 import numpy as np
2 a = np.array([5, 6])
3 b = np.array([7, 8])
```

解説します。

1 行目は numpy を使うけれども長いから np と略して使うよ、という意味です。

ここはほぼほぼ慣習です。

二行目と三行目で、a と b に 5, 6 と 7, 8 を代入しました。ここから下記を入力します。

```
1 print(a+b)
```

結果

```
1 [12, 14]
```



---

このように計算できます。

ちなみに、numpy の配列と素の python のリストは違うものであり、素の python ならこうなります。

```
1 a = [5, 6]
2 b = [7, 8]
3 print(a + b)
```

結果

```
1 [5, 6, 7, 8]
```

numpy と普通の list は list 関数や numpy.array 関数で相互に変換できます。  
他に numpy.arange 等非常に有用です。

```
1 import numpy as np
2 np.arange(5, 13, 2)
```

結果

```
1 array([5, 7, 9, 11])
```

これは 5～13 までの間、2 刻みの数列を作るという意味です。

そのほか、多くの機能があり MNEpython のベースとなっています。

出力結果が numpy 配列で出てくるので、MNE があるとはいえ使い方は覚える必要があります。

numpy は多次元配列を表現できますが、その次元は  
shape というメンバー変数に入っています。

```
1 import numpy as np
2 data = np.arange(5, 13, 2)
3 print(data.shape)
```

結果は敢えて書きません。手を動かさなければ  
君は決して numpy を使いこなせるようになりません。

解析を始める前の **warning!**

ここまでは単なる unix 系の知識だけで済んでいましたが、この辺りからは数学の知識、  
python を流暢に書く技術、脳波脳磁図のデータ解析の常識等、色々必要です。

python を書くのは本気でやればすぐ出来ますが、

微分だとか行列計算を全部理解して応用するのはかなり面倒くさいです。

時に、mne-python のソースコードを読む必要にかられます。<sup>46</sup>

---

<sup>46</sup>この同人誌にも、ソースコードを読んで書き換えた所があります。

---

同人誌で完璧に説明するのは無理なので、一寸だけしかしません。  
また、データ解析の常識は進化が速いうえにその手の論文を  
読めていないと正確なところは書けません。  
僕の知能はチンパンジーレベルなのでここからは不正確な部分が交じるでしょう。  
本書は純粋な技術書であることに留意し、最新の知識を入れ続けましょう。

---

## jupyter 用図用おまじないセット

このへんのおまじないは素の python 使っているならいりませんが、jupyter や ipython のときは必要でしょう。

下記は jupyter/ipython のコマンド

```
1 %matplotlib inline
2 %gui qt
```

%matplotlib inline については、この設定なら jupyter 上に表示されます。

もし、別窓<sup>47</sup>を作りたいなら、inline を変えてください。

python3 の場合

```
1 %matplotlib qt5
```

python2 の場合

```
1 %matplotlib qt
```

となります。

下の%gui qt は mayavi による 3D 表示のためのものです。

mayavi が python3 で動くかどうかは僕はまだ確認してないです。

他に、こういうものもあります。

```
1 import seaborn as sns
```

matplotlib の図を自動で可愛くしてくれるゆるふわパッケージです。

---

<sup>47</sup>生の波形を見たいときなどにはそのほうが向いてる

---

## データの読み込みとフィルタリング・リサンプル (公式サイト版)

ついに MNE を使い始めます。まずは下記リンクを開けてください。

[http://martinos.org/mne/stable/auto\\_tutorials/plot\\_artifacts\\_correction\\_filtering.html](http://martinos.org/mne/stable/auto_tutorials/plot_artifacts_correction_filtering.html)

ちょっと小難しい文法を使っているように見えます。

小難しい部分は初心者は混乱するだけなので無視してください。

難しいなら読み飛ばして、次に移ってください。簡単にまとめています。

公式サイトでは脳磁図前提としていますが、ここではついでに脳波の読み込みの解説もやります。

是非脳波、脳磁図のファイルを手元において、読み込んだり

フィルタを掛けてみてください。(でないと、覚えられません)

ここでは

- データの読み込み
- パワースペクトル密度のプロット (以下 psd)<sup>48</sup>
- notch filter と low pass filter を使って要らない波を除去する
- サンプリングレートを下げて処理を軽くする

をしています。

はじめのパラグラフ<sup>49</sup>で大事な関数は以下です。

- `mne.io.read_raw_fif`: 脳磁図のデータを読み込みます。  
ここではつけていませんが、通常 `preload=True` をつけた方がいいです。  
`preload` をつけると、メモリ上に脳磁図データを読み込み、  
色々と処理ができるようになります。(付けないと処理できないです...)  
脳波を解析するなら下記公式サイト Reading raw data セクションに  
各種形式に対応した読み込み関数がありますから、読み替えてください。  
[http://martinos.org/mne/stable/python\\_reference.html](http://martinos.org/mne/stable/python_reference.html)

読み込みの詳細は後で書きます。

- `mne.read_selection`: 脳磁図の一部を取り出しています。
- `mne.pick_types`: データの中から欲しいデータだけ取り出します。
- `plot.psd`: psd プロットを行います。

基本は体で慣れるしかありませんが、大抵のエディタでは  
tab キーとかを押せば補完してくれるので、入力自体は楽です。

---

<sup>48</sup>各周波数ごとの波の強さをあらわしたもの。フーリエ変換の結果算出されるものの1つ。

<sup>49</sup>ipython/jupyter 独自の単位で cell といいます。通常のプログラミングでは行ごと、関数ごとですが、jupyter では数行をひと塊りにしてプログラムを書きます。色々悪いところもありますが、解説用コードを書けるのが jupyter の強みです。

---

例えば「raw.」と書いて tab を押せば、plot 関数だけでも色々出てきます。  
だから、色々プロットして遊んでみてください。

次の cell では notch filter をかけています。

- notch\_filter

これは特定の周波数を削除するフィルタです。

何故それをするかというと、送電線の周波数が影響するからです。  
西日本では 60Hz、東日本では 50Hz です。それを除去できます。<sup>50</sup>

関数内に np.arange と書いてあるのは numpy の関数。

60 から 241 までの間で 60 ごとの等差数列を返すものです。

つまり、ここでは 60Hz を除去しています。

次に low pass filter をかけます。

- filter

これは分かりやすいでしょう。ある周波数以上、以下の波を除去します。

バンドパスフィルタと言います。

ERP をする時は遅い周波数成分を除去するときは注意が必要です。

その場合は 0.1Hz 未満とするのがいいのかもしれませんが。

また、ICA でノイズ除去する時は 1Hz くらいでかけるといいです。

最後にサンプリングレートを変えています。

理由は今後処理がかなりのものになるので負担を軽くしたいからです。

- resample

ここでは 100Hz まで下げていますが、最低見たい周波数の 2~3 倍以上の周波数が必要です。

また、周波数は元の周波数の約数である必要があります。

というか、普通 resample とか必要ないんですが、何かここではやっていますね。

以上...MNE の公式サイトは一寸詳しいです。初心者にはちょっとつらかったですね...

よしよし。

## データの読み込みと filter, resample(僕の解説)

公式サイトは python をバリバリ書ける上に生理学をきちんと理解できている人向けに感じます。

本書はあくまで初心者向けです。

先ずは大雑把に理解して体を動かすべきと思うので、以下は極めて乱暴な僕なりのまとめです。

大まかに理解した上で公式サイトに取り組みれば良いのではないのでしょうか？

---

<sup>50</sup>敢えて除去しない研究者もいます。notchfilter によって、除去する周波数の周辺の信頼性が失われるとのこと。

---

極めて乱暴にまとめると、ノイズ取りの第一段階はこうです。

```
1 raw = mne.io.read_raw_fif('hoge',preload=True)    #読み込み
2 raw.filter(1,100)    #0.1~100Hzの波だけ残すバンドパスフィルタ
3 raw.notch_filter([60,100])    #この場合、60と100Hzを消してる
4 raw.save('fuga')
```

…ていうか、このくらい雑な例のほうが良いと思う。

ちなみに、第0段階があります。それは `badchannel` の指定、`interpolation`、`maxfilter` 等ですが、とりあえず読めなきゃ話にならないので。

- 1行目で読み込みます。脳波と脳磁図では読み込み方が違うので、次セクションを参照。  
`preload` を `True` にしてください。そうしないとメモリ上に読み込んでくれません。<sup>51</sup>  
`fif` というのは適当に読み替えて下さい。
- 2行目でバンドパスフィルタかけてます。1Hz未満の波と、100Hz以上の波を消しています。<sup>52</sup>
- 3行目で送電線のハムノイズを取っています。<sup>53</sup>
- 4行目でデータを間引いて処理を軽くしています。必ず元データの約数に設定し、  
`wavelet` 変換するならば `wavelet` 変換の最高の周波数の2~3倍以上の周波数にしてください。  
というか、僕なら `resample` はしません。データが荒くなるからです。
- 5行目で掃除した結果を `'fuga'` という名前で保存しています。

あとは、`plot` を色々してみてください。

以下、本書ではこのような乱暴な解説をしてとりあえず手で覚えた後、理屈を覚えていくスタイルにしていきます。

## 脳波読み込みの問題

脳波はすんなり読み込めたでしょうか？ そうでもないかもしれないですね。

なにしろ、脳磁図と違って脳波は沢山の形式があるのです。

例えば、ヘッダーファイルを要求する形式があったりもしますし、

モンタージュや眼球運動チャンネルの設定を追加せねばならぬ場合もあります。

このセクションは試行錯誤が要求されます。

ちなみに、`mne` は `mne.io.read_raw_hoge` って感じで色々なデータに対応しています。

やたら対応できるのは多いので適当に見て下さい。

---

<sup>51</sup>`preload` しないと各種処理が出来ないので、ほぼ必須です。何故 `preload` が標準で `False` なのかはよくわかりませんが、`False` も使いみちがあります。例えば生波形を素早く表示するだけならば `preload` は `False` が軽いです。

<sup>52</sup>バンドパスフィルタについては賛否両論と思います。何故なら、時間周波数解析をすると要らない周波数は消えちゃうので、意味が無いという考えがあるからです。詳しくは参考文献の `analyzing neural` ...を読んで下さい。個人的にはソースレベル解析の場合はした方がいいと思います。

<sup>53</sup>`notch` フィルタもバンドパスフィルタ同様賛否両論です。

---

さて…脳波は色々な企業が参入していますが、  
脳波のファイルには以下の情報が入ってたり入ってなかったりです。

- 波形データ
- チャンネル名と空間データ
- 測定条件

このあたりは脳波計のユーザーが設定できる所もあったりするので、  
脳波計の管理者に聞いたりするのが早いかもしれません。  
また、モンタージュ (センサーの空間情報) を指定せねばならぬ事もあります。  
例えば biosemi の場合は下記のように書きます。

```
1 raw = mne.io.read_raw_edf(  
2     filename,  
3     preload=True,  
4     montage='biosemi64',  
5     eog=['eye-l', 'eye-r'],  
6     exclude=['X1', 'X2', 'X3', 'X4']  
7 )
```

解説します…。

- filename  
これは問題ないですね
- preload  
これも問題ないです。前のセクションを御覧ください。
- montage  
これは場合によっては問題ありですのであとで解説します。
- eog  
これは、眼球運動は何番目のチャンネルだよ、というやつですね。単純。
- exclude  
これは、このチャンネルはいらないよ、というやつです。  
余りチャンネルが有ることは日常茶飯事です。数字で指定もできます。

## 日本光電

最近 mne では日本光電の形式を読めるようになりました。  
これで病院づとめの人も脳波解析が出来ますね！  
mne.io.read\_nihon 関数です。  
なんか、日本国を呼んでいる感じがして名前がデカいですね！

---

## EDF で event 情報が読み込めない場合

EDF 形式はかなりメジャーな脳波の形式です。

event 情報が文字列として入ってたりします。

そんな時は MNEpython では読めません。なので、別のソフトを使います。

使うソフトは pyedflib です。インストールしましょう。

```
1 pip install pyedflib
```

そして、コードを書くのですが、たいへん面倒いです。

```
1 import pyedflib
2 edf = pyedflib.EdfReader('hoge.edf')
3 annot = edf.read_annotation()
```

これで annot(list 形式) にイベント情報が入ります。

しかし、annot の中を覗くと分かると思いますが、たまーにこの annot の中に 2 行で 1 つのイベントとかが入ってたりして、そいつを 1 つのイベントとして書き直すスクリプトを書かないといけなかったりするので面倒臭いです。

頑張って書いて下さい。

## EDF が少しも読み込めない場合

EDF 形式はメジャーなのですが、そんな中にも色々な形式があります。

EDF+C だとか EDF+D だとか。EDF+D は凄く読み込みにくいです。

pyedflib は EDF+D を読めません。しかし、万事休すではありません。

open source のいいソフトがあります。edfbrowser というソフトです。

<https://www.teuniz.net/edfbrowser/>

このサイトには windows 版が公開されていますね。

このソフトは tools メニューから EDF+D を EDF+C に変換する事ができます。

mac や linux の人はコンパイルしてください。

このサイトには mac のコンパイルの仕方が書いてありませんが、それはこのようにします。

まず、xcode を app\_store からインストールします。

そして、homebrew をインストールします。ググってください。

その上で、下記のようにして git と qt をインストールします。

```
1 brew install qt
```



---

```
2 brew install git
```

そして、ソースコードをダウンロードします。  
ソースコードのフォルダの中で、

```
1 qmake
2 make
```

とすると、バイナリが出来上がります。

脳波のセンサーの位置が変則的な場合

さて…montage の話をします。montage は要するにセンサーの空間情報です。  
この世には色々な脳波の取り付け方があります。「は？ 10-20 法しかねえよ！」  
と言われそうですが、あるものは仕方ないのです。センサーの数の違いもありますし。

MNEpython では出来合いのモンタージュセットがあります。  
10-20 法ならだいたいセンサーはこの辺だよ、というやつですね。  
それは上記のように文字列で指定できます。大抵はこれで事足ります。  
しかし、時々凄くニッチなセンサー配置の脳波計があったりします。  
そういうのは MNEpython で対応できないこともしばしばです。  
そんな時には文字列じゃなくてモンタージュ情報を別途読み込んで、  
montage 変数に入れなきゃなりません。めんどいです。  
詳しくは [mne.io](http://mne.io) の解説記事をみて下さい。形式ごとの解説記事があります。  
ちなみに下記のように raw を読み込んだ後で指定する事も可能です。

```
1 from mne.channels import read_montage
2 mont = read_montage('standard_1020')
3 raw.set_montage(mont)
```

脳波のセンサーからソースベース出来んの？

これについては大きな声では言いたくないのですが、出来ます。  
ちょっと捻ったやり方が必要です。  
ただし、脳波のセンサーを位置情報としたソースベース解析が  
どの程度の精度を持っているかは…お察しください。

まず、上記の raw.set\_montage(mont) に一つオプションを入れます。

```
1 raw.set_montage(mont, set_dig=True)
```

---

これをすると、raw.info の中に raw.info["dig"] という項目が入ります。  
この dig の中に位置情報が入りますから、これを使って位置合わせが出来ます。

さて、これを僕がやってみた時に、  
どういうわけか meg の montage 情報に比べて縮尺がやたら大きかったです。  
僕がやったときはなんと千倍のサイズでした ()  
この大きさになると無理感が出てくるので、ちっちゃくしちゃいました。

```
1 for n in raw.info['dig']:
2     n['r']=n['r']/1000
```

無理矢理感あふれるやり方ですね…。  
こうすることにより、MEG と同じ様に mne coreg が出来るようになりました。  
皆さんは皆さんの条件に合わせて格闘して下さい。  
mne coreg については後述します。ソーススペース解析の所を御覧ください。

脳波のセンサーの名前が変則的な場合

もう一つかなり面倒くさい問題があります。  
MNEpython はチャンネルの位置情報を自動で設定する時に  
ファイルの中に記述されているチャンネルの名前を参照して  
位置情報を当てはめていきます。これの何が困るのでしょうか？

脳波計が montage の 'Fp1' という風な普通の名前で出力してたら良いのですが、  
例えば 'EEG-Fp1' という風な名前だったら名前を変えてあげないと読めないのです。  
名前は大事なのです。

変える方法としては、raw.rename\_channels 関数を使う方法があります。  
mne.channels.read\_montage の解説記事を開いてみて下さい。

まず、チャンネルの名前を表示しましょう。  
いっぱいモンタージュ情報が書いてありますが、ここでは 10-20 法を見えます。

```
1 mont = mne.channels.read_montage('standard_1020')
2 print(mont.ch_names)
3 mont.plot()
```

凄くたくさんのチャンネル名と図が出てきましたね？  
次に、読み込んだ脳波のチャンネルリストを見てみましょう。

```
1 print(raw.ch_names)
```

---

チャンネル名が同じ名前になっているでしょうか？  
なっていなかったら書き換えていかねばなりません。  
書き換えるには、python の辞書形式を利用します。  
2つのチャンネル名をよく見比べて変えていって下さい。  
下記のような辞書を作っていきます。

```
1 channel_list = {
2     "EEG Fp1-Ref": "Fp1", "EEG Fp2-Ref": "Fp2",
3     "EEG F3-Ref": "F3", "EEG F4-Ref": "F4",
4     "EEG C3-Ref": "C3", "EEG C4-Ref": "C4",
5     "EEG P3-Ref": "P3", "EEG P4-Ref": "P4",
6     "EEG O1-Ref": "O1", "EEG O2-Ref": "O2",
7     "EEG F7-Ref": "F7", "EEG F8-Ref": "F8",
8     "EEG T3-Ref": "T3", "EEG T4-Ref": "T4",
9     "EEG T5-Ref": "T5", "EEG T6-Ref": "T6",
10    "EEG Fz-Ref": "Fz", "EEG Cz-Ref": "Cz",
11    "EEG Pz-Ref": "Pz", "EEG A1-Ref": "A1",
12    "EEG A2-Ref": "A2"}
```

脳波の基準電極や眼球運動や心電図もこんな風に辞書にして下さい。  
では、この辞書を使ってチャンネル名を変えましょう。

```
1 raw.rename_channels(channel_list)
```

これで、脳波のチャンネルの名前を変え終わりました。  
最後に、用意した montage と脳波をくっつけます。

```
1 mont=mne.channels.read_montage('standard_1020')
2 raw.set_montage(mont)
```

これで上手くいけば普通に MNEpython で解析できます。

## 基準電極

基準電極の設定は下記のような感じでできます。

```
1 raw = mne.set_eeg_reference(
2     raw, ref_channels=['LMASTOID'])[0]
```

が、普通脳は研究では全体の平均で設定することが多いようですから、下記のようなのが普通でしょうか。

```
1 raw2=mne.set_eeg_reference(raw)[0]
```

---

ちなみに、初期設定では全体の平均を基準電極としていますから、この設定は実は不要です。

末尾の [0] はこの関数が list 形式で結果を出してくるから必要です。

詳細は

[http://martinos.org/mne/stable/python\\_reference.html](http://martinos.org/mne/stable/python_reference.html)

を見て、各自読み替えてください。

このようなスクリプトははじめは面倒ですが、  
一度書いてしまえば後は使いまわしたり自動化出来ます。

### トリガーチャンネル

もう一つの鬼門がトリガーチャンネルです。つまり、刺激提示の時刻を記録したものです。  
これは通常下記で表示できます。

```
1 mne.find_events(raw)
```

raw 中の刺激提示チャンネルが読めない場合はどうにかしてテキスト形式とかで書き出してください。  
そこからは…貴方はもちろん pythonista なので書けるはずです。

例えば、pandas を使って

```
1 import pandas as pd
2 shigeki=pd.read_csv('hoge.csv')
```

後はゴリゴリスクリプト書いてください。

僕もこのようなトリガーチャンネルについて苦労しました。

僕の場合はトリガーが脳波と同じように波形として記録されていたのです。脳波の波形は

```
1 raw.get_data()
```

で出力することが出来ます。内容はチャンネルごとの numpy 形式の数値です。  
サンプリング周波数を鑑みてがんばってください。

トリガーチャンネルは信号が入ったら波形が跳ね上がっていたので、  
僕はその跳ね上がりを検知するようなスクリプトを書くことで解決しました。

---

## bad channel の設定

苦行その 1 です。次にダメなチャンネルの設定や眼球運動の除去を行います。

[http://martinos.org/mne/stable/auto\\_tutorials/plot\\_artifacts\\_correction\\_rejection.html](http://martinos.org/mne/stable/auto_tutorials/plot_artifacts_correction_rejection.html)  
これには 2 つのやり方があります。

### やり方 1

`raw.plot()` でデータを見ながらひたすら下記のように  
`badchannel` を設定してってください。それだけです。

```
1 raw.info['bads'] = ['MEG 2443']
```

`badchannel` は、例えば明らかに一個だけ滅茶苦茶な波形…  
振幅が大きくて他のとぜんぜん違う動きしているとか、  
物凄い周波数になっているとか、毛虫っぽいとか、そういうやつを選んでください。

### やり方 2(おすすめ)

```
1 raw.plot()
```

をした上で、画面上でポチポチクリックしていけば、`raw` に `bad` が  
入っていくように出ています。便利ですね！  
もちろん、あとで保存しないとちゃんと残りません。

```
1 raw.save('hoge.fif')
```

python の対話モードを使って毎回一々やっていくのは超絶面倒なのでスクリプトにしたいかと思いま  
す。

しかし、その場合 `plot` し終わったらすぐ python が終了して図が即消えます。  
それを防ぐには以下の一行を入れましょう。

```
1 input()
```

## interpolation

選び終わったら、`badchannel` を補正します。  
隣接するチャンネルを平均したようなやつで置き換えることになります。  
それには下記を走らせるだけでいいです。

---

```
1 raw.interpolate_bads()
```

後で badchannel を無視した ICA を掛けるとか、色々出来るわけです。

## maxfilter

ここは EEG には不要です。

しかし、MEG は EEG と違ってノイズがのりやすいです。

何故なら、地球の磁力は脳みその磁力の 1000 倍だからです！

MEG 使う人は maxfilter を使ってノイズを除去することが出来ます。

maxfilter は elekta 製のと mne 製のがあります。

ここでは mne 製のを紹介します。

[https://mne-tools.github.io/stable/generated/mne.preprocessing.maxwell\\_filter.html](https://mne-tools.github.io/stable/generated/mne.preprocessing.maxwell_filter.html)

さて、maxfilter には 2 つファイルが必要です。

この 2 つのファイルは、それぞれの施設によって違うものです。

一つは calibration 用の dat ファイル、一つは crosstalk 用の fif ファイルです。

これについては elekta の機械ならあるはずなので、そこから抜き出すといいでしょう。

ここについては僕は詳しくないので、周囲の賢者に聞いて下さい。

もう一つ、MNE の maxfilter には特徴があって、badchannel を設定してあげないとうまく動きません。

因みに、elekta のは自動で badchannel を設定しちゃうそうです。

```
1 from mne.preprocessing import maxwell_filter
2 cal = 'hoge.dat'
3 cross = 'fuga.fif'
4 raw = maxwell_filter(raw, calibration=cal,
5                       cross_talk=cross, st_duration=10)
```

この maxwell\_filter 関数で行います。

calibration と cross\_talk は見てのとおりと思いますが、st\_duration も大事なやつです。

MNEpython の標準の設定では st\_duration は None なのですが、

実際は数値を設定しないと酷いことになります。

公式サイトには「俺たちの MEG はキレイだから None で良いんだ」と

ドヤ顔していましたが、町中の MEG だと地下鉄通るだけで酷いことになるので、

大草原の小さなラボとかでないなら設定してあげましょう。

元祖 elekta maxfilter ではここが 10 になっています。

この st\_duration の数字は実は highpass filter の役割も果たします。だから、注意が必要です。

1/st\_duration 以下の周波数がカットされるので、遅い周波数を見たい人は気をつけて下さい。

---

その他、いろいろな理由で `st_duration` は出来れば大きな値が良いそうですが、計算コストが上がるという欠点がありますので、程々に。

## ICA でノイズを取ろう

苦行その 2 です。

ICA は日本語で言うと独立成分分析と言い、古典的機械学習の一種です。

つっても、波形に関する機械学習でいえば大変有用です。<sup>54</sup>

10 個の耳で聞いた一つの音を、10 個の「有効っぽい成分」に分けちゃう方法です。

何故 ICA をするかというと、ノイズ取りです。

前回やったノイズとは違うノイズを取ります。

例えば眼球運動や心電図、場合によっては筋電図です。

眼球運動や心電図は特徴があるので、ICA で分けた時分かれてくれるのですね！

順を追って内容を説明します。

```
1 from mne.preprocessing import ICA
2 from mne.preprocessing import create_eog_epochs, create_ecg_epochs
```

まずは、ICA のモジュールをインポートします。

```
1 picks_meg = mne.pick_types(raw.info, meg=True,
2                             eeg=False, eog=False,
3                             stim=False, exclude='bads')
```

次に、どのような波に ICA をかけるか選びます。基本、解析したい脳磁図 (脳波) に ICA をかけるので、それを `True` にします。`badchannel` も弾きます。

```
1 n_components = 25
2 method = 'fastica'
3 decim = 4
4 random_state = 9
```

`n_components` は ICA が分ける波の数です。

ICA で分ける波の数は何個が良いのか僕にはよく分かりませんが、

あまり少なすぎないほうがいいでしょう。

ここではひとまず適当に 25 個にしています。

`method` は `ica` の方法です。方法は三種類選べます。API 解説ページをご参照ください。

---

<sup>54</sup>DeepLearning が今の世の流行りですが、DeepLearning は画像が得意だという点があります。これは CNN という手法があるからですが、波形を CNN に適用する手法もあります。でも、これが筋が良いかというと色々意見もあろうかと。

---

decimはどの程度詳しくICAをかけるかの値です。

数字が大きくなるほど沢山かけますが、数字を入力しなければ最大限にかけます。

random\_stateは乱数発生器の番号指定です。

pythonでは乱数テーブルを指定することが出来ます。

そうすると、再現可能な乱数(厳密には乱数ではない)が生成できるようになります。

実はICAは乱数を使うので、結果に再現性がないのですが、

この擬似乱数テーブルを用いることにより再現性を確保しつつ乱数っぽく出来るのです。

(狡いけど) 便利ですね！

```
1 ica = ICA(n_components=n_components,
2          method=method, random_state=random_state)
3 raw = raw.filter(1, 100)
4 ica.fit(raw, picks=picks_meg, decim=decim, reject = dict( grad=4000e-13))
```

ここで、rawに周波数フィルタをかけています。これはICAが低周波の影響を受けやすく、ノイズをとることに失敗してしまうのを防ぐためです。

icaで特徴を抽出し、データに適用しています。

この時点ではまだ何も起こっていません。下記を実行してください。

jupyterなら先に%matplotlib qtと入力した上でです。

分離した波(特徴)が表示されます。

```
1 ica.plot_sources(raw)
```



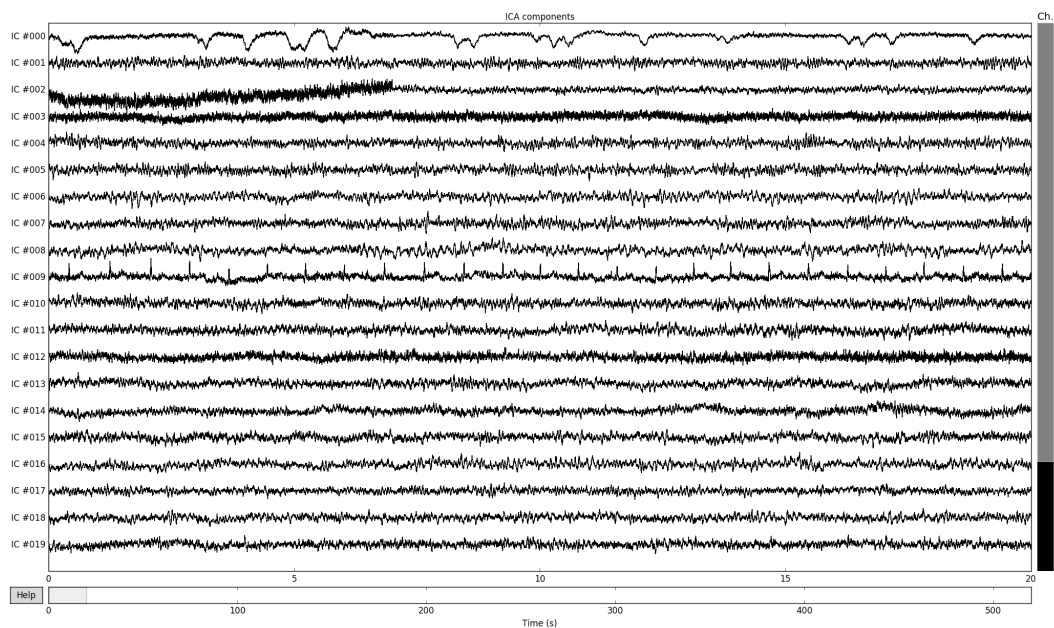


Figure 8: ica で分離した波。明らかに眼球運動や心電図が分離された図が出てくると思います。

チャンネル名のところをクリックすると topomap が表示されます。

個人的には生波形を見るのが明快で好きです。

ちなみに、これを凄く詳しく見るには下記のようになります。重いですが、これも結構良いです。

```
1 ica.plot_properties(raw, picks=0)
```

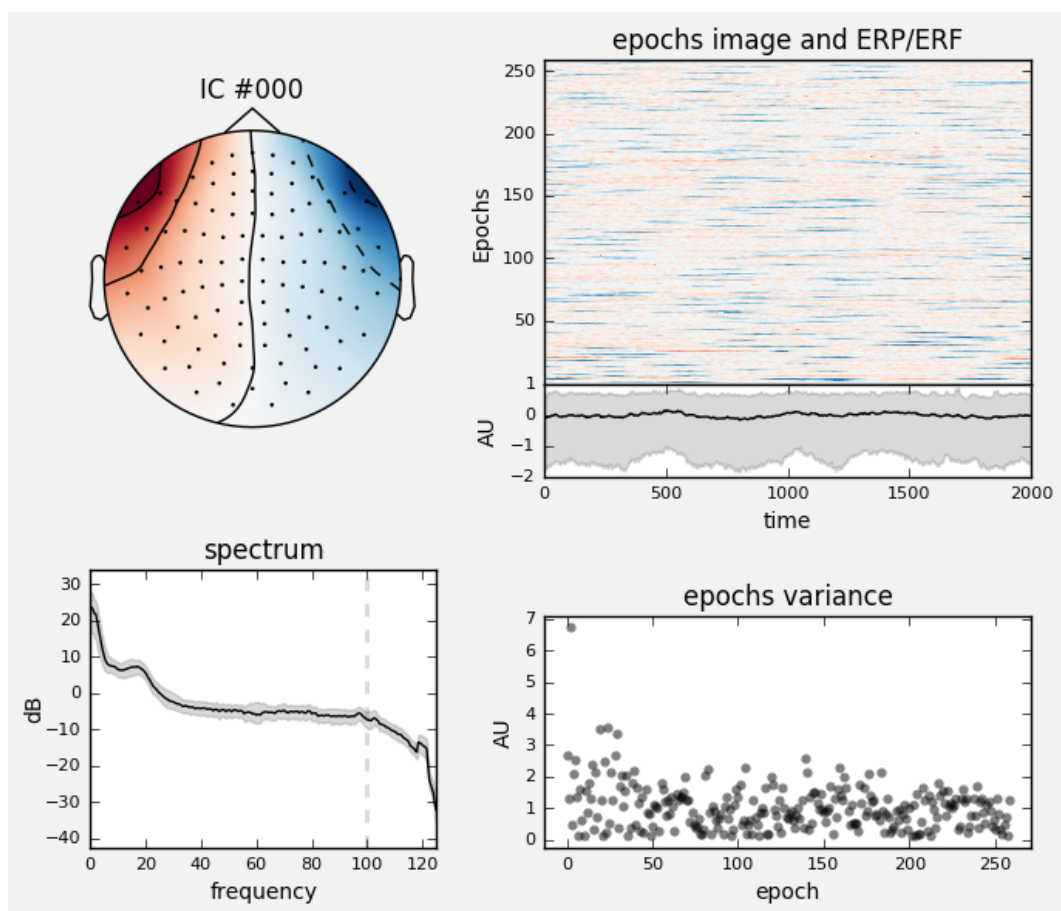


Figure 9: ica property の図。左上を御覧ください。これこそが典型的な眼球運動の topomap です。

最後に、0 番目と 10 番目の波を raw データから取り除きます。

```
1 filtered_raw=ica.apply(raw,exclude=[0,10])
```

ここでの raw はフィルタを掛けていない raw...で良いのか僕はよく分かりません。

ともかく、これで ica は終わりです。

上記の出力結果や取り除いたチャンネル、random\_state は保存しておきましょう。

random\_state を保存しておくことで ICA に再現性が生まれます。

(狡いんですけどね)

---

## ICA コンポーネントのより良い取り除き方

実際に上記を手動でやるのは恣意的になったり、再現性が無かったり、面倒臭すぎたりして、なにより面倒くさいので僕は嫌いです！

(大事なことなので二回言いました)

それはともかく、抜く波が恣意的になるのはいただけません。

例えば病人だけに出る波が欲しい時に健常者から波を抜きまくってしまう姑息な事をする輩が居るかも知れません。それはいただけません。

そこで、もっとクールなやり方が2つあります。

### 自動判定

眼球運動チャンネルや心電図をとっていたら、それに似てるやつを自動判定してくれる機能が MNE-python にはあります、やったね！やり方は以下のとおりです。

まずは、眼球運動がある場所を眼球運動によって epoch を作ります。

```
1 from mne.preprocessing import create_eog_epochs
2
3 eog_epochs = create_eog_epochs(raw, reject=reject)
4 eog_inds, scores = ica.find_bads_eog(eog_epochs)
```

簡単ですね！ eog\_inds は眼球運動に超似ているチャンネルの番号リストです。scores はどれだけ似ているかの度合いです。とりあえず、plot しましょう。

```
1 ica.plot_scores(scores, exclude=eog_inds)
```

どれが悪いコンポーネントかが plot されたかと思います。  
では、どの程度浮き立っているか確認しましょう。

```
1 ica.plot_sources(eog_epochs.average(), exclude=eog_inds)
```

浮き立っている度合いがわかったかと思います。では、詳しく見てみましょう。

```
1 ica.plot_properties(eog_epochs, picks=eog_inds)
```

詳しいですね！ いい感じであれば一網打尽にしていましょ。

```
1 ica.exclude = eog_inds
2 ica.apply()
```

心電図については殆どこいつが ecg になっただけだから、もう解説はしません。

---

## 半自動判定

眼球運動チャンネルや心電図をそもそも取っていない時はどうするのでしょうか？

その時は一部のコンポーネントを「根本的なノイズだよ」と指定して、それに似ているコンポーネントを一網打尽にすることが出来ます。では、やっていきましょう。

まずは、ICA のオブジェクトをいっぱい作ります。上記の `ICA.fit()` で出来るやつですね！

で、それらを沢山並べてリストにします。

リストにしたものを作る時、きっと時間がかかるので、`ICA.save` で保存してから読み込むほうが良いでしょうね。

超絶面倒なので `map` 関数を使います。(沢山の物に同じ関数を適用する関数)

```
1 from mne.preprocessing import read_ica
2 ica_paths = ['hoge.fif', 'fuga.fif', 'piyo.fif']
3 icas = list(map(read_ica, ica_paths))
```

で、この `ica` のリストの中から典型的なノイズを選んできます。

例えば 5 番目の `ica` の 3 番目のコンポーネントがノイズっぽい場合はこうします。

```
1 template = (5, 3)
```

で、`corrmap` という関数にぶち込みます。

```
1 from mne.preprocessing import corrmap
2 corrmap(icas, template, threshold='auto', label=None,
3         ch_type='eeg', plot=True, show=True,
4         verbose=None, outlines='head',
5         layout=None, sensors=True, contours=6, cmap=None)
```

- `icas`: 要するにさっきのリストです
- `template`: さっきのテンプレートです
- `threshold`: どのくらい似てるものまで引っ掛けるかです。  
標準は 'auto' なのですが、'auto' では中々何も引っかかりません。
- `label`: 引っ掛けたやつにつけるラベルです。文字列入れて下さい。
- `ch_type`: eeg なら eeg ですし、meg なら mag とか grad になります。

だいたい、そんな感じです。`corrmap` を `plot=True` の条件でかけると、いっぱい似てるやつが引っかかってきます。

`label` に何か入れていれば、`ica` にラベルがつきます。

`ica.labels_` に格納されており、`label` の情報は辞書形式です。

```
1 {'eog': [1], 'ecg': [2]}
```

---

この例では、label を 'eog' と 'ecg' の二回分 corrmap をまわしたときの結果みたいなもんですな！

どの程度の閾値にすれば適切か分からないので試行錯誤しましょう。

corrmap は違うラベルでやれば、違うラベルがどんどん追加されていきます。

ところで、ラベルに情報が入っても、print(ica.labels\_) みたいに

しないと貴方はそれを見れません。

plot してくれないのです...これでは実際の sources がどんな感じが分かりませんね？

```
1 raw = Raw('hoge.fif') # ダメな例
2 icas[0].plot_sources(raw)
```

label に情報が入るだけなのでこのままではダメです。

ica.exclude は List 形式なのでこれをどうにかしたいですね。

まあ、せいぜい工夫して下さい。僕ならこうします。

```
1 from operator import add
2 from functools import reduce
3
4 ica.exclude = list(set(reduce(add, ica.labels_.values())))
```

set は重複のない値を格納するオブジェクト、reduce は調べて下さい。

python 初学者は面食らうやり方ですね。こういう風にベタにかいてもいいですね。

```
1 for n in ica.labels_.values():
2     if n not in ica.exclude:
3         ica.exclude += n
```

こうしてやれば plot\_sources したときに悪いコンポーネントは赤く表示できるようになります。

いい感じであれば ica を保存するといいでしょう。

良くない感じなら閾値を変えたりチャンネル変えたりしてやり直しです。

このやり方をする時のコツとしては、「こいつこそが典型的な眼球運動だ！」という奴を選ぶことと、HighpassFilter を掛けた上で作業を行うことです。

## Epoch と Evoked

初心者にはなんのことやら分かりにくい単語ですが、波形解析には重要な用語です。

Epochs はぶつ切りのデータそのものです。Evoked は「誘発された波」です。

元データ (raw) に「ここで刺激したよ！」という印を付けておいて、

後からその印が入っているところだけ切り出していきます。

---

切り出してきたものが Epochs です。

Evoked は切り出したものを加算平均して算出します。

例えば元データ (raw) に刺激提示したタイミングを記録しているならば、下記のコードでその一覧を取得できます。

```
1 events=mne.find_events(raw)
```

この events 情報からほしいものを抜き出してきて、epoch や evoked を作ります。

上記 events の内容は例えばこうなります。

```
1 221 events found
2 Events id: [1 2 4 7 8]
3 Out[205]:
4 array([[ 15628,      0,      2],
5        [ 18053,      0,      2],
6        [ 20666,      0,      4],
7        [ 23131,      0,      1],
8        [ 25597,      0,      8],
```

この場合刺激チャンネルには 1, 2, 4, 8 という刺激が入っています。

このうち、刺激情報 1 を使って切り出したいときは下記です。

```
1 epochs = mne.Epochs(raw, event_id=[1], events=events)
```

先程の events を使っています。

event\_id は配列にしてください。ここは [1, 2] とかも出来るのでしょうか。

evoked を作るのはとても簡単で、下記のとおりです。

```
1 evoked = epochs.average()
```

---

## データの plot

是非自ら plot してみてください。何をやっているのか理解が早まると思います。

```
1 epochs.plot()  
2 evoked.plot()
```

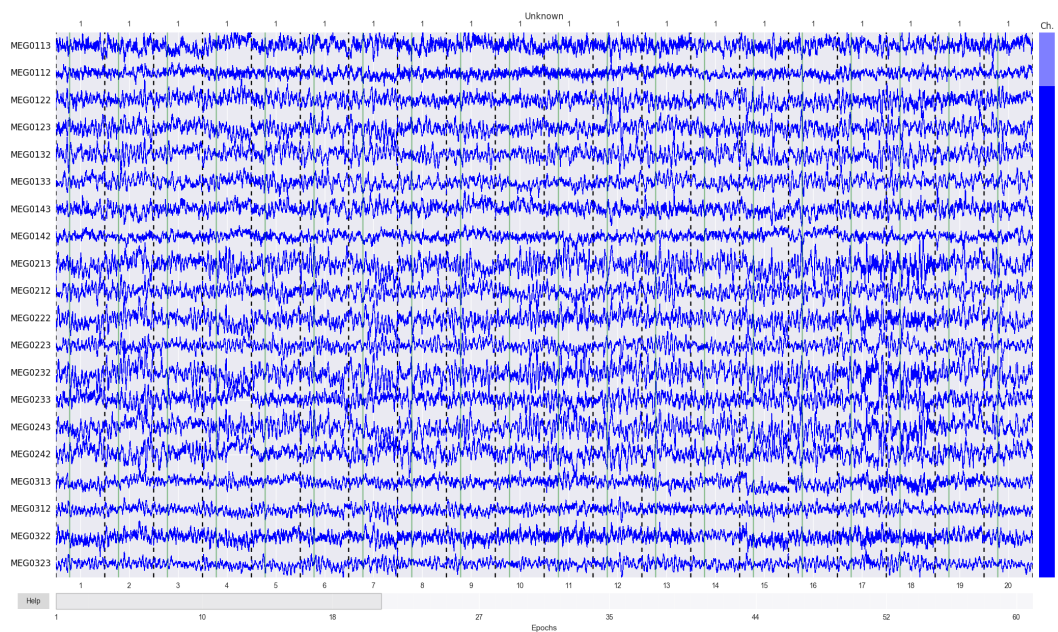


Figure 10: epochs の例

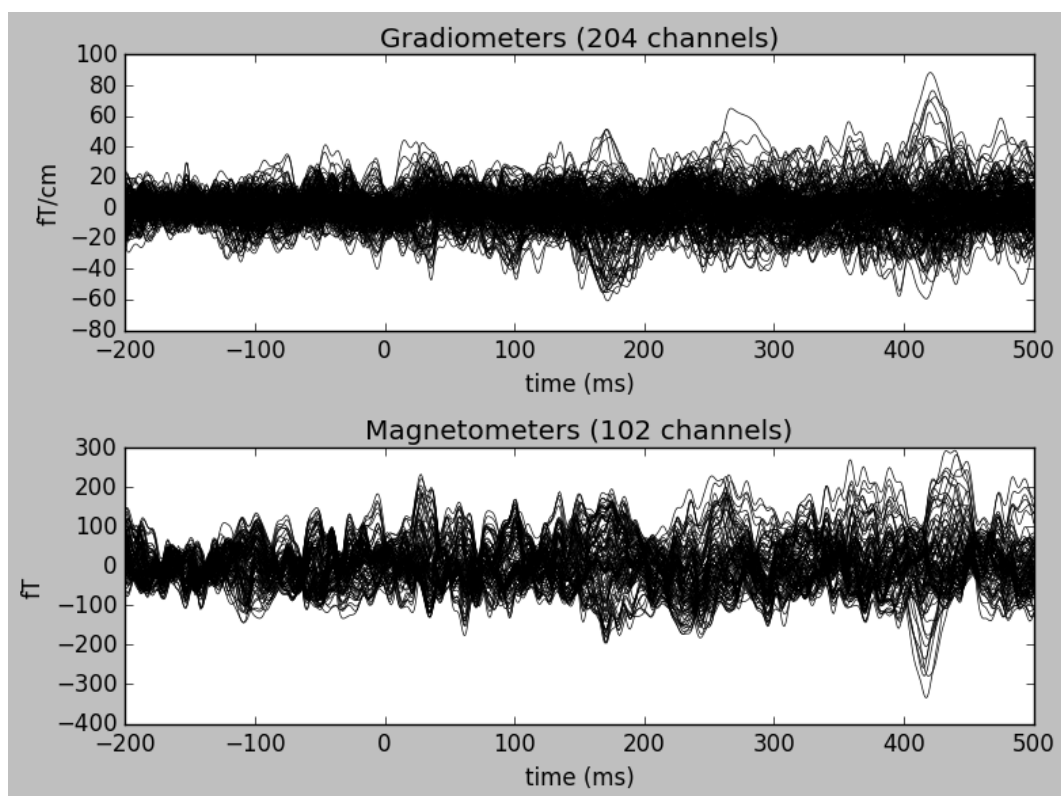


Figure 11: evoked の例

epochs や raw をプロットしたとき、どうなったでしょうか？

### jupyter の場合

jupyter ではどのように表示するかを選ぶことができます。

jupyter にそのまま表示したい場合は下記を先に jupyter 上で実行してください。

```
1 %matplotlib inline
```

別の window に表示したいときは下記のようにしてください。

```
1 %matplotlib qt
```

また、3D 画像を表示したい場合は

```
1 %gui qt
```



---

jupyter に表示するメリットは jupyter 自体を実験ノート風に使えること、別ウィンドウに表示するメリットは raw や epoch 等大きなデータを表示する時にスクロールさせることが出来ることです。

(反面、バージョン管理がしんどい)

実は jupyter 上でスクロール出来る表示もあるのですが、重くてあまり良くないです。詳しくは qiita で検索してください。親切な記事がいくらでもあります。

## pysurfer

また、PySurfer については例えば下記のような感じです。

これは mac の場合ですが、ubuntu も同じ感じです。

subject や subjects\_dir は freesurfer の設定で読み替えてください。

jupyter で下記の呪文を唱えましょう。

```
1 import surfer
2 %gui qt
```

そしてこうです。この場合ブロードマン1を赤く塗っています。

```
1 brain = surfer.Brain(subject, "lh", "inflated",
2 subjects_dir=subjects_dir)
3 brain.add_label("BA1.thresh", color="red")
```

注意すべき点として、拡張子や左右半球にかんしては add\_label 関数では省略して入力する必要があります。

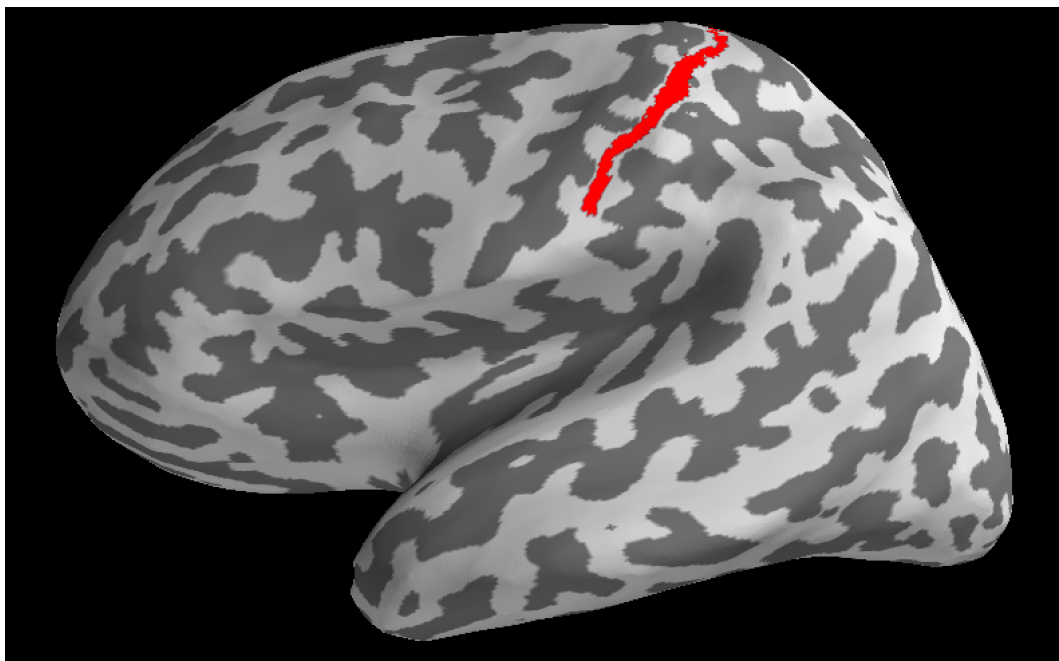


Figure 12: pycortex で表示した freesurfer のラベルファイル

ちなみに、label ファイルはそれぞれの subject の中の label フォルダの中にあります。  
この label についてはブロードマンの脳磁図ベースの古典的なものが多いですね。  
新しい系は annot ファイルの中に多いです。

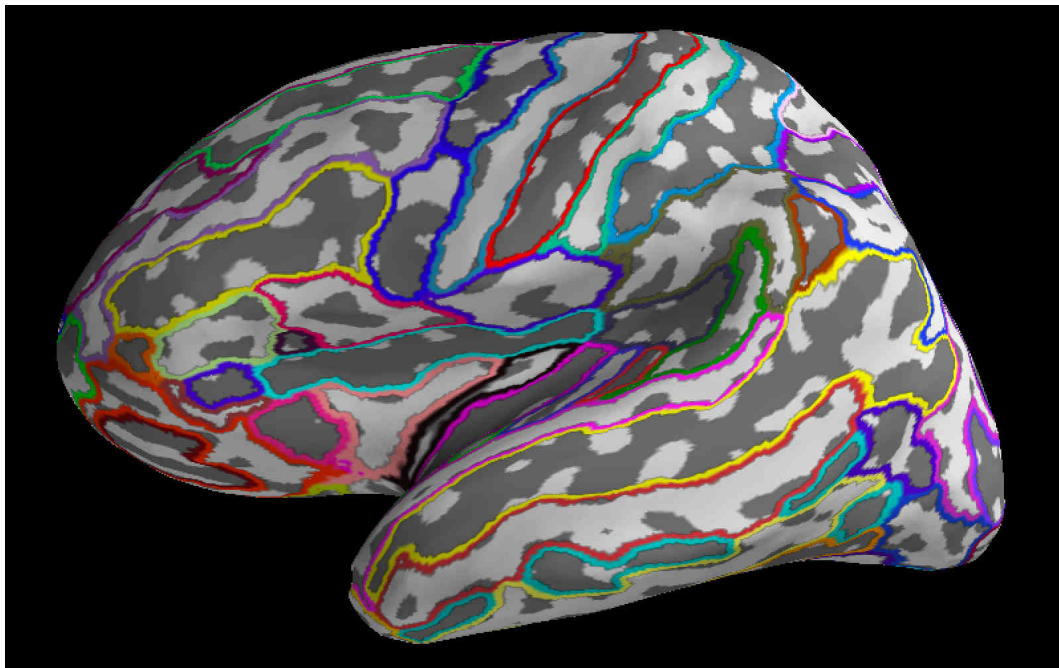


Figure 13: pysurfer で表示した freesurfer の annotation ファイル

```
1 brain = surfer.Brain(subject, "lh", "inflated",
2 subjects_dir = subjects_dir)
3 brain.add_annotation('aparc.a2009s')
```

沢山表示されていますね。僕はちょっと気持ち悪いなあと思いました。  
一つだけ表示すればいいなら以下のようにします。

```
1 labels = read_labels_from_annot(hoge, 'aparc')
2 labels = list(filter(lambda x: label_name in x.name, labels))
3 b.add_label(label)
```

hoge には Subject を入れてください。

さて、実は脳みそだけでなく、脳みその中の活動の動画も撮ることが出来ます。  
それについてはソース推定のセクションで書こうと思います。

## numpy の plot

これ、結構面倒くさいです。では、表示していきましょう。numpy の情報を data とします。  
しかし、例えば wavelet 変換をした情報なんかなら、

---

時間軸、周波数軸、チャンネルというふうに、多次元です。  
二次元のほうが皆さん見やすく好きですね？ では、二次元にします。

```
1 data_mean = data.mean(axis=0)
```

mne では三次元配列を多用しますが、とりあえず axis=0 でうまくいくことが多いですね。  
ここは適当ですが、いい感じに調整して下さい。

```
1 import matplotlib.pyplot as plt
2 def make_and_save_fig(data, fname)-> None:
3     fig, ax = plt.add_subplot()
4     ax = plt.imshow(data, vmax=0.25, cmap='rainbow')
5     ax.set_yticks(np.arange(85, 0, -5))
6     ax.set_yticklabels(np.arange(15, 100, 5))
7     ax.set_xticks(np.arange(0, 1000, 100))
8     ax.set_xticklabels(np.arange(-300, 700, 100))
9     ax.invert_yaxis()
10    plt.savefig(fname)
11    plt.clf()
```

二次元の画像データを plot するやつですね。  
set\_yticks はデータのどの部分に目盛りをつけるかを指定したもの。  
set\_yticklabels はデータの目盛りに書き込む内容です。  
ここでは、15 から 100Hz の周波数について解析して、5Hz ずつ目盛りをつけていったのですね。  
matplotlib は突然 plt として出てきていますが、これは仕様です。  
ax に吐き出したものは plt で色々するんですね。詳しくはググって下さい。

## 多チャンネル抜き出し

もし、多チャンネルの evoked を平均したものを割り出したいなら貴方は numpy を使うことになります。  
ここでは脳波の evoked を例にしておきます。他のデータでも応用ききます。  
下記のチャンネルを選択したいとします。

```
1 channels = ['Fz', 'FCz', 'FC1', 'FC2',
2             'Cz', 'C1', 'C2', 'F1', 'F2']
```

python の配列では、中の項目を逆引きで探し出す .index() 関数があります。  
加工した波形データは .data 変数の中に格納されています。その一番初めの情報が  
チャンネル別なので、1 チャンネル...例えば 'Fz' なら下記のようにすれば割り出せます。

```
1 evoked.data[evoked.info['ch_names'].index('Fz')]
```

この 'Fz' を for 文で書きかえていけば良いのです。

---

```
1 data = []
2 for channel in channels:
3     wave = evoked.data[evoked.info['ch_names'].index(ch)]
4     data.append(wave)
```

---

## センサーレベル wavelet 変換

これは解析のゴールの一つと言えましょう。  
特定の周波数の波の強さとか、揃い具合を数字に出来れば論文が  
書けるってわけです！ ヒャッハウ！

そもそも wavelet 変換とは何なのか

波形解析の方法です。画像変換にも使えます。  
ここでは特定の周波数の波の強さや位相を計算する方法として使います。  
フーリエ変換という言葉をご存知でしょうか？  
フーリエ変換は全ての波を sin 波の合算として解釈することで波を一つの式として表す方法です。  
余程やバイ波じゃなければ、ほぼ全ての波はフーリエ変換によって近似的に変換できるのです。  
凄いですね！ しかし、これには欠点があります。不規則な波の変化に対応できないのです。  
何故なら、sin 波は未来永劫減衰しない波だからです。  
フーリエ変換において、波は未来永劫つづくのが前提なのです。  
(擬似的に切り取ることは出来るし、普通にそれだけでも研究は成立する)

そこで、減衰する波を使って波を表す方法を使います。これが非直交 Wavelet です。<sup>55</sup>  
そのため、減衰する波を単純な数式で表現する必要があります。

これを理解するためには高校数学を理解する必要があります。<sup>56</sup>  
なんと！ この同人誌は高校を卒業した人向けの同人誌であったのか！？  
まあ、中卒の人でも高校数学の特定の数学を勉強すれば十分いけます。勉強すべきなのは

- 三角関数
- 複素数
- 指数・対数
- 微分方程式

と、このあたりです。詳しくは後半の「初心者のための波形解析」を御覧ください。

---

<sup>55</sup> もしも完全に消えちゃうのが欲すれば直交 Wavelet というのもあるのですが、ギザギザしており、解析にはあまり向きません。理屈編で語ります。

<sup>56</sup> 高校を卒業した事のある人は目をそらさないでいただこうか。

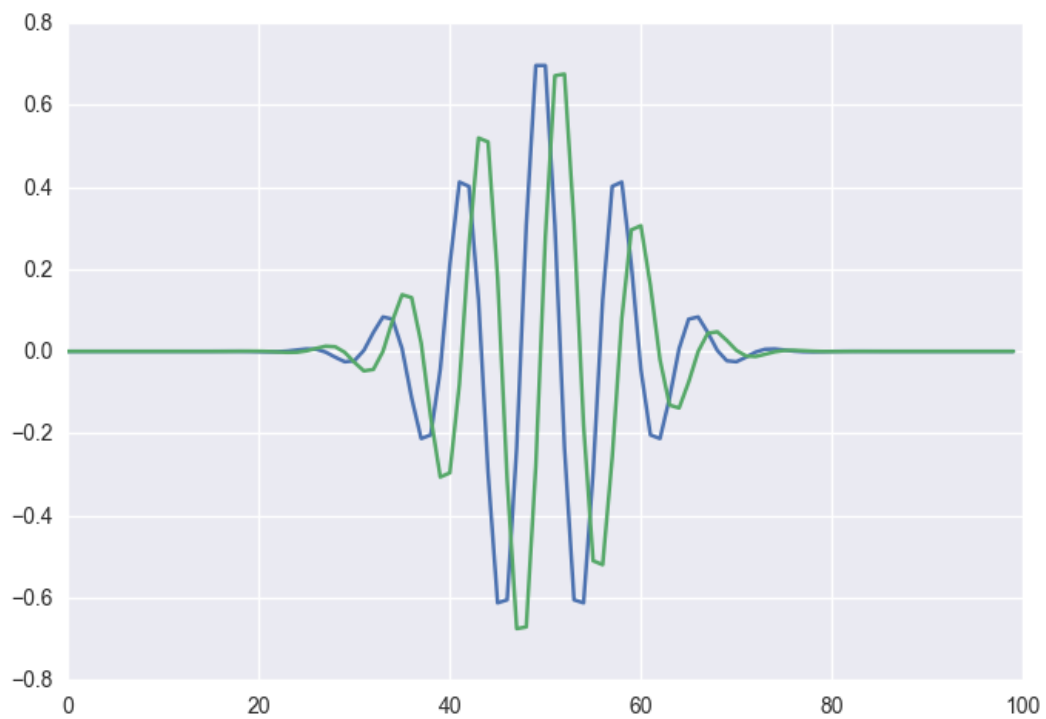


Figure 14: wavelet の例。これは morlet wavelet という種類。morlet はモルレと読む。青は実数部分、緑は虚数部分。

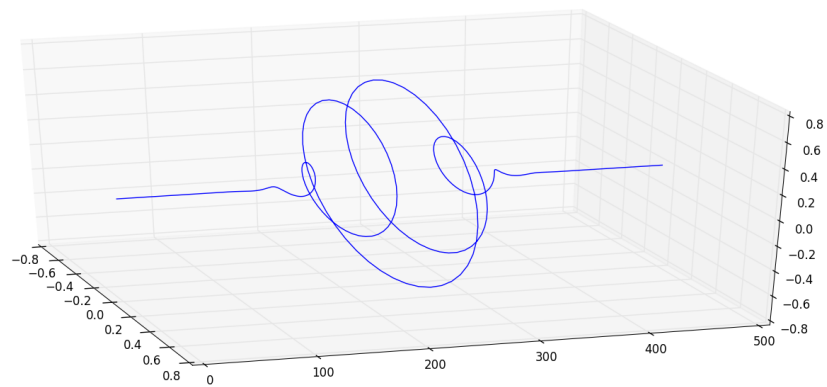


Figure 15: morlet wavelet の実数軸、虚数軸、角度軸による 3d plot。

---

## wavelet 変換にまつわる臨床的な単語

波の強さや揃い具合を測れると言っても、まずは用語がないとどうしようもありません。  
脳活動に関する用語としては以下のようなものがあります。(ただし、コネクティビティ系は除く)

単語	内容
total power	何らかの刺激を受けて出てくる power
evoked power	何らかの刺激を受けた直後に同期して出る power
induced power	何らかの刺激を受けて刺激に同期せずに出る power
phase locking factor	同一部位での位相同期性

どういうことでしょうか？

つまり、何か刺激を受けた時に「受け取った直後にすぐ起こる」のが evoked  
何か刺激を受けた後「なんかよく分からんけど生じる不規則な波」が induced  
上記の合算が total です。

total power は簡単に計算できます。単に wavelet 変換して  
結果の絶対値を二乗すればそれで終わりであります。

evoked power は二種類の出し方があります。まあ、流派みたいなものでしょうか。

induced power もまた、二種類の出し方があります。

これは後で書きます。

このなかで、Phase Locking Factor は別名 Inter Trial Coherence(itc) といいます。

MNEpython では itc という言い方しています。<sup>57</sup>

それぞれ生理学的には違うものを見ているらしいです。

本書では itc という言い方にしておきましょうか…。

MNEpython ではセンサーベースならどれも実装されています。

ソースベースでは induced power と itc の計算方法が実装されています。<sup>58</sup>

では Evoked Power, Induced Power, Inter Trial Coherence について解析を行いましょう。

---

<sup>57</sup>ちなみに phase locking value という全然別のものがあります。これはコネクティビティ用語ですので分野が違います。あとで書きます。

<sup>58</sup>これは実質 itc と似たようなもの…という考え方もあります。



---

## 2つの流儀と MNE

InducedPower の計算の仕方に 2 つの流儀があります。  
名前については適当につけました。どっちが正しいとかはない。

**生波形引き算派** 生波形を合算していくと、そのうちキレイに Evoked の波形が出ます。  
この波形を生波形から引き算します。しかる後、Wavelet 変換して、これを InducedPower とします。  
その上で、TotalPower から InducedPower を引き算します。

**EvokedPower 引き算派** 生波形をまず Wavelet 変換して、TotalPower を出します。  
そして、生波形の合算も Wavelet 変換して EvokedPower を出します。  
TotalPower から EvokedPower を引き算して InducedPower を計算します。

### wavelet 変換の実際

morlet のやり方は臨床研究的にメジャーなやり方と僕は思っています。  
まずは EvokedPower からやりましょう。下記のスクリプトで実行できます。  
この場合は合算の後に Wavelet 変換ですね。

```
1 from mne.time_frequency import tfr_morlet
2 freqs=np.arange(30,100,1)
3 n_cycles = 6
4 evoked_power = tfr_morlet(evoked, n_jobs=4, freqs=freqs,
5                           n_cycles=n_cycles, use_fft=True,
6                           return_itc=False, decim=1)
```

- freqs : どの周波数帯域について調べるか。  
上の例では 30Hz から 100Hz まで 1Hz 刻みに計算しています。
- n\_cycles : 一つの wavelet に含まれる波のサイクル数。  
5~7 という値で固定する方法がよく用いられます。  
MNE ではこのサイクル数を可変にすることも出来ます。
- n\_jobs : CPU のコアをいくつ使うか。重い処理なのです。  
ちなみに、n\_jobs を大きくするよりも、n\_jobs を 1 にして  
同時にたくさん走らせたほうが速いです...が、メモリは食います。
- use\_fft : FFT による高速 wavelet 変換を行うかどうか。  
数学の話になるので、詳しい所は本書では扱いません。  
要するに速く計算するかどうかです、True でいいかと。

- `decim` : この値を大きくすると処理が軽くなりますが、出力結果がちょっと荒くなります。
- `return_itc` : これを `True` にすると `phaselocking factor` も算出してくれます。

この関数は `evoked` も `epochs` も引数として取ることが出来ます。

`return_itc` が `True` か `False` かで大きく挙動が違います。

挙動の組み合わせについてですが、下記のとおりです。

<code>return_itc</code>	引数	返り値 1 つ目	返り値 2 つ目
<code>False</code>	<code>evoked</code>	<code>EvokedPower</code>	なし
<code>False</code>	<code>epochs</code>	<code>TotalPower</code>	なし
<code>True</code>	<code>epochs</code>	<code>TotalPower</code>	<code>InterTrialCoherence</code>

Induced が無いじゃないか！ ってなりましたね？

Induced を計算したければ、`Epochs` のオブジェクトで以下のようにします。

```
1 epochs.subtract_evoked()
```

これをするとう波形から `Evoked` が引かれて計算できるようになります。または、

`itc` を計算したい時は返り値が 2 つになりますから、下記の通りです。

```
1 freqs=np.arange(30,100,1)
2 n_cycles = 6
3 total_power, itc = tfr_morlet(epochs, n_jobs=4, freqs=freqs,
4                               n_cycles=n_cycles, use_fft=True,
5                               return_itc=True, decim=1)
```

ここで一つ注意点があります。

`wavelet` 変換は基準になる波を実際の波に掛け算して行うのですが、

波の始まりと終わりのところだけは切れちゃうはずですが、そこは十分注意して下さい。

どの程度の `wavelet` の波の長さなのかについては、勉強して適当に計算して下さい。

---

## データの集計について

データの集計についてですが…実は結構面倒くさいです。

MNE は個人個人のデータを解析するモジュールだからです。

それぞれのデータは MNE のオブジェクトの中に入っています。

貴方は個人個人のデータを MNE で解析した後、

そのデータを自分で集計する必要があります。numpy を使う必要性はここで出てきます。

MNE のオブジェクト (itc, power, evoked, epochs, raw 等) は

ユーザーがいじることが出来るようになっています。

中の実データはそれぞれのオブジェクトの中の data という変数か、

または get\_data 関数で抽出してくることになります。

power なら power.data に、raw なら raw.get\_data() に入っています。

こうして出してきた配列は numpy 形式の配列です。

ピックアップした情報は多次元配列ですから、内容は膨大です。直接見ても整理つきません。

そこで便利な変数が numpy にはあります。例えば evoked のデータを作ったならば

```
1 evoked.data.shape
```

とすればデータの構造が確認できます。

この場合は [チャンネル、時間] になろうかと思います。

データの構造としてはこんな感じのようです。括弧がついているのはオブジェクト内の関数です

形式	データ	1 次元目	2 次元目	3 次元目
raw	raw.get_data()	チャンネル	波形	
epochs	epochs.get_data()	チャンネル	波形	
evoked	evoked.data	チャンネル	波形	
itc	itc.data	チャンネル	周波数	波形
power	power.data	チャンネル	周波数	波形

揃っていませんね…。

(どうせ使うのは evoked 以下くらいなので大して困りません。)

まあ、itc や power は波形データじゃないので揃ってなくてもいいですが。

それぞれのオブジェクトは

---

`object.save(filename)`

とすれば保存できます。

読み込みは多くの形式に対応する必要があつて一寸複雑です。

形式	読み込み関数	備考
raw	<code>mne.io.Raw()</code>	脳磁図の場合。脳波とかは公式サイト API 参照
raw	<code>mne.io.read_raw_hoge()</code>	様々な形式がある。hoge は読み替えてね。
epochs	<code>mne.read_epochs()</code>	
evoked	<code>mne.read_evoked()</code>	条件によって list で返されることがあり
itc	<code>mne.time_frequency.read_tfrs()</code>	条件によって list で返されることがあり
power	<code>mne.time_frequency.read_tfrs()</code>	条件によって list で返されることがあり

例えば

```
1 itc = mne.time_frequency.read_tfrs('/home/hoge/piyo')[0]
```

という感じで読み込みます。行の最後についている `[0]` は上記のごとく

条件によって list で返されることがある関数だからです。

そうじゃない関数の場合は `[0]` は不要です。実際に手を動かして練習すればわかると思います。

さて、実データのみではサンプリング周波数やチャンネルの名前が分からず

困ったことになりますが、`mne/python` ではこれらは

それぞれの object 中の `info` という python 辞書形式変数に入っています。

例えば `print(itc.info)` とか `print(itc.info["ch_names"])` とかで

読めたりしますから確認してみてください。僕はこの `info` を使ってチャンネルを抽出したりします。

ここまでの知識で、自分で `numpy` 形式で脳波脳磁図を扱えるようになります。

あとは下記のようにすれば良いと思います。

1. `power` なり `itc` なり波形なり、個人レベルで計算する
2. `numpy` 形式で 1 チャンネル抜き出したり数チャンネルの平均取ったりする
3. 個人個人で数字が出てくるので、それを保存する
4. R でその数字を統計解析する

例えば `hoge` チャンネルの `fugaHz` から `piyoHz`、

`foo` 番目から `bar` 番目 (秒 × サンプリング周波数) の反応までの

実データを抽出したいなら、

---

```
1 itc.data[hoge, huga:piyo, foo:bar]
```

何番目ってアレですね...なら、foo と bar を秒単位にして、こうしましょう。

```
1 sfreq = 1000
2 itc.data[hoge, huga: piyo, foo / sfreq: bar / sfreq]
```

です。ちなみに、wavelet 変換時に `decim` の値を設定している場合は  
(秒 × サンプル周波数 / wavelet 変換の `decim` の値) となります。  
API ページで `time_frequency.tfr_morlet()` 関数をご参照ください。

2 は numpy の `mean` 等で実現します。

`import numpy as np` の後

```
1 np.mean(itc.data[hoge, huga:piyo, foo:bar])
```

などとすれば良いと思います。

3 は python の基本構文通りなので解説しません。

4 はどのようにしたいかは人によって違うかと思います。

最近僕は単純に csv 形式に書き出しています。pandas なんかはとても素敵です。

numpy でも普通の list でも csv に変換してくれます。こうすればいいです。

```
1 from pandas import DataFrame
2 DataFrame(hoge).to_csv(filename)
```

## jupyter での R と pandas の連携

「R を jupyter で動かすために」である程度書きましたが、再掲します。

jupyter 上で

```
1 %load_ext rpy2.ipython
```

とした後

```
1 %%R -i input -o output
2 hogehoge
```

という風に記述すれば hogehoge が R として動きます。

このデータの受け渡しにも pandas を使うのが良いです。

項目には名前をつけることが出来ます。

```
1 from pandas import DataFrame
```

---

```
2 data = Dataframe(data
3                     columns=('group',
4                               'hemisphere',
5                               'test', 'value'))
```

これで、横軸に columns のラベルの付いたデータフレームが出来ます。  
こいつを to\_csv を使ったり jupyter とかで R にぶちこみます。

```
1 %%R -i data
2 result <- aov(
3   df$value ~ df$group * df$hemisphere * df$test,
4   data=df))
5 cat(result)
```

だいたいこんな感じです。

## R での ANOVA について

python の scipy での統計もいいのですが「なんで統計ソフト使わないん？ 舐めてるん？」  
と reject を食らう可能性もありますから辞めましょう。

どうせ多重検定することになるんですから、それについて一寸。

だいたい ANOVA を用います。aov が R の ANOVA 関数です。

これを summary 関数に読ませることで結果を簡単にまとめます。

さらに、cat 文を使うことで画面上に表示します。

中の式は、データフレーム内の掛け算になっています。

ANOVA 詳しい人は知っていると思いますが、これは相互作用を算出するものです。

相互作用を計算しない場合は '+' 演算子を使ってください。結果が算出されると思います。

あとは ANOVA の本でも読んで下さい。本書では割愛します。

R によるやさしい統計学という本が僕のおすすめです。

---

## Connectivity

Connectivity を脳波でやってみましょう。Connectivity は要するに、脳のあちこちの繋がり具合を調べる指標です。MRI とかでよくされている手法ですね。

mne-connectivity では脳波と脳磁図でこれを計算することが出来ます。

実装されている計算方法を列挙してみます。

まずは、何はなくても計算しやすくする変換をせねば始まりません。

変換方法は下記の3つが提供されています。

- multitaper
- fourier
- morlet wavelet

フーリエ変換や wavelet 変換をした上で、それぞれの値を比較するのです。

比較の方法は下記のとおりです。

- Coherence: Coherency の絶対値
- Coherency: 純粋に計算で出されたやつ
- ImaginaryCoherence: 同一ソースの影響を除いたもの
- Phase-Locking Value: 純粋に計算で出されたやつ
- Phase Lag Index: 同一ソースの影響を除いたもの
- Weighted Phase Lag Index: PhaseLagIndex に重みを付けたもの

…多すぎですね (´・ω・`)

このうち、Coherence と PhaseLockingValue は敢えて言おう、カスであると。

それ以外のどれが良いとかは…よくわかりません。

色々やってみたり先行研究を見るのが良いかも？

これらの詳細については波形解析の理屈編に一応書きました。

とりあえず、計算方法を書いておきます。まずは、epoch を作ります。作り方は前述のとおりです。

眼球運動や心電図のデータは要らない<sup>59</sup>ので、

pick\_channel や drop\_channel で要らないのを外していきます。

```
1 epochs.pick_channels(['hoge'])
2 epochs.drop_channels(['fuga'])
```

では、始めましょう。

```
1 from mne_connectivity import spectral_connectivity_epochs
2 cons = spectral_connectivity_epochs(
```

---

<sup>59</sup>大抵は、の話です。心臓の鼓動と脳波のコネクティビティの研究も一応あります！

---

```

3     epochs, method='coh', indices=None,
4     sfreq=500, mode='multitaper', fmin=35, fmax=45, fskip=0,
5     faverage=False, tmin=0, tmax=0.5, mt_bandwidth=None,
6     mt_adaptive=False, mt_low_bias=True,
7     cwt_frequencies=None, cwt_n_cycles=7,
8     block_size=1000, n_jobs=1)

```

…基本、我流の僕はソースコードが汚いんですが、今回はあまりにも一行あたりが長すぎて一ページに収めにくかったんです…。やむを得ず圧縮のために sc と短縮しました…。では、解説いきます。

- method: そのまま method です。上記の通り。
- indices: どことどの connectivity を見たいかです。
- sfreq: サンプリング周波数です。
- mode: multitaper, fourier, cwt\_morlet の 3 つを選べます。
- fmin, fmax: 見たい周波数帯域です
- fskip: どのくらい飛び飛びで解析するかです。
- faverage: 最終的に幾つかの周波数を平均した値を出すかどうかです。
- tmin, tmax: どこからどこまでの時間見るか
- cwt\_frequencies: morlet wavelet の時の周波数 (numpy 形式の数列)
- cwt\_n\_cycles: morlet wavelet の波の数
- block\_size, n\_jobs: 一度にどのくらい計算するか

この関数は、中々詰め込み多機能な関数です。

なんと、上記の沢山の method を全部できます。出来るがゆえの大変さもあります。

con は三角行列です。

三角行列というのは、行列の対角より上か下が全部 0 で出来ている行列です。

$$A = \begin{pmatrix} 0 & 0 & 0 \\ 4 & 0 & 0 \\ 6 & 3 & 0 \end{pmatrix}$$

Figure 16: 三角行列の例



---

## fourier/multitaper モード

fourier や multitaper は時間軸がないです。その辺が morlet wavelet と違うところです。  
con の内容は [チャンネル数 X チャンネル数 X 周波数] という三次元配列になります。  
この内、チャンネル数 X チャンネル数 の部分が三角行列になります。  
周波数は、関数が勝手に「これがいいよ」と言って抜き出してきた  
離散的な周波数になります。

ここで、幾つかの周波数について個別にやりたいなら話は違うのですが、  
加算平均したいなら下記のコードで十分です。

```
1 conmat = con.get_data(output='dense').mean(2)
```

これで、conmat に三角行列が入りました。

## wavelet モード

morlet wavelet は乱暴に言うとも fourier に時間軸を与える拡張版です。  
[チャンネル数 X チャンネル数 X 周波数 X 時間] という 4 次元になります。  
この場合は下記のコードで三角行列を作りましょう。

```
1 conmat = con.get_data(output='dense').mean((2,3))
```

三角行列が出来ました。

## plot

さっきの 2 つは三角行列を作るモードでした。  
三角行列を正方行列にしましょうか。

```
1 conmat = conmat + conmat.T
```

ここは使う手法にもよりますね。  
方向が大事な場合にはこのようなことはしないかも知れません。  
では、plot しましょう。

```
1 from mne_connectivity.viz import plot_connectivity_circle
2 plot_connectivity_circle(conmat, epochs.ch_names)
```

plot\_connectivity\_circle は内容的に単に plot するだけなので、解説見ればいいですが、  
n\_lines で大きいコネクティビティだけを plot してくれたりするので嬉しいです。

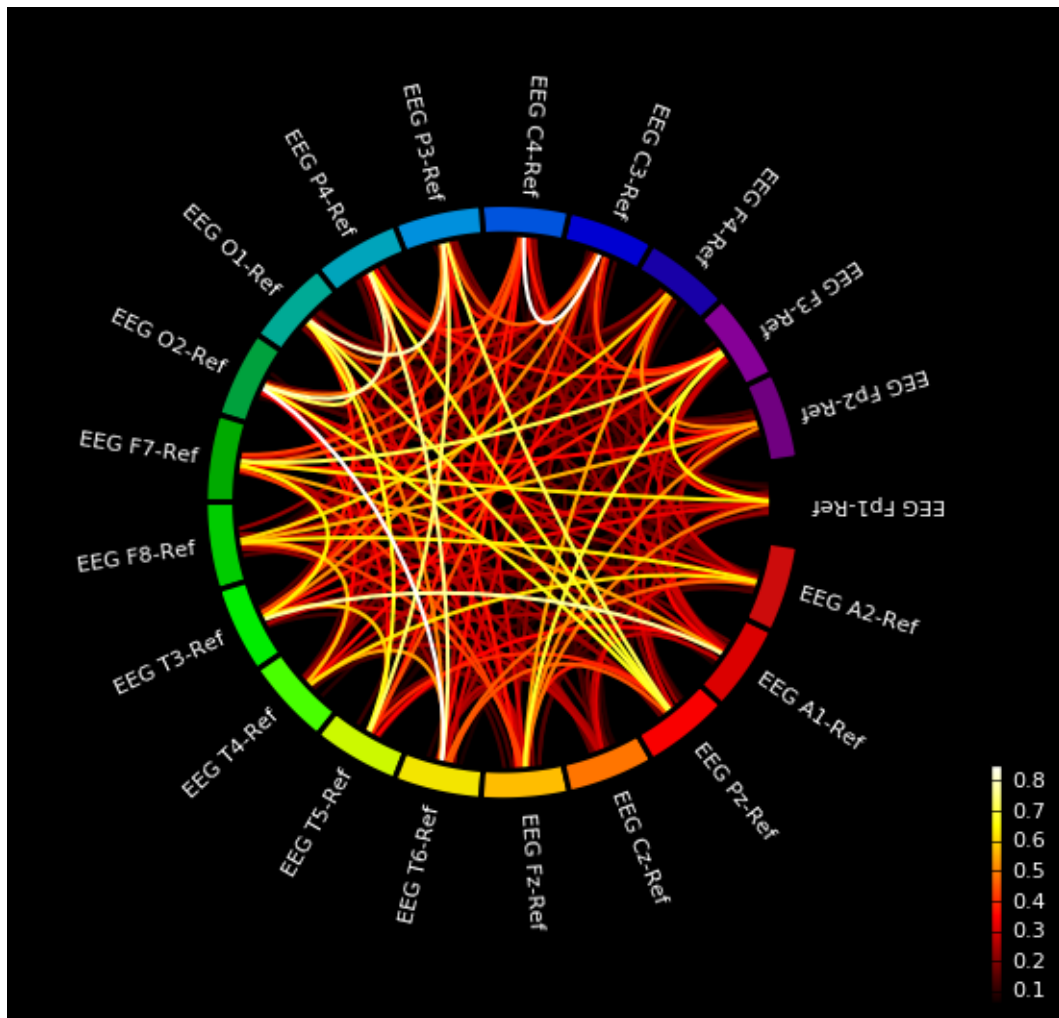


Figure 17: 僕の脳波のコネクティビティの図。花火みたいで綺麗なので好きです。

## indices モード

三角行列関連は、要するに全部入りな感じの計算でした。

indices というところに引数を入れると、特定の connectivity だけ計算してくれます。

```
1 indices = (np.array([0, 0, 0]),  
2           np.array([2, 3, 4]))
```

このように numpy 配列を作ります。1 列目は何番目のチャンネルとそれぞれを見比べたいか。

2 列目はそれぞれのチャンネルです。ここでは

0 → 2 , 0 → 3 , 0 → 4 番目のチャンネルを比べています。

---

で、この `indices` を引数として入れるとどうなるかというと、  
`fourier/multitaper` モードなら [見比べたチャンネルの数 × 周波数] となります。  
`morlet wavelet` モードなら [見比べたチャンネルの数 × 周波数 × 時間] となります。

---

## ソースレベル MEG 解析

ついにソースレベルの解析を行います。これが MNE/python の真髄です。

すこし難しいのです。頑張りましょう。

数学分かる人向けにいきますと、こいつは MAP 推定です。

1. 脳の構造画像からセンサーへの伝わり方を計算する
2. 事後分布として分散共分散行列を用いる
3. MAP 推定する

という手順です。

数学わからない人に適当に言うと

1. MRI とセンサーの位置情報から方程式を作る
2. どの程度の波の大きさを想定するか設定する
3. 上記から大まかなあたりをつけて推定する

という感じ。

ソースレベル解析の性質や仕組みは他の所を見ていただくとして、早速 MRI と MEG をくっつけていきます。

(MRI がない場合は標準脳を使えるけど、あまり感心しない)

目標は「脳内の信号を算出するための式を作る」事です。

式さえできればなんとか計算できるわけです。

必要物品は以下の通り

- 脳の中の見たい場所リスト (SourceSpace)
- センサーの位置情報 (Montage)
- 脳波か脳磁図の結果 (Raw)
- 皮膚や頭蓋骨の抵抗値や、その分布 (BEMmodel)

このなかで、MRI については標準脳を使うのでなければ

freesurfer で recon-all をしたデータが必要です。

前述しましたので、頑張ってください。一晩かかります。

これらを使って何をするかというと、

脳の中の活動と、センサーで捉えた結果で連立方程式を解いてあげるのです。

さて、これは理工系の人には知っているのですが、

実は連立方程式を解くという行為は割り算に他ならないのです。

(ここで文系や医学部の人にはびっくりする)

---

大学数学を習ってなかった人やサボっていた人はあとの方に書いた  
「鶴亀算とは割り算である」をご参照ください。

さて、脳の中の電気の活動量を  $X$ 、センサーで捉える磁場とか電場とかを  $Y$  とすると  
 $AX = Y$  という形式に落とし込めるはずで。

$A$  は適当な定数です。距離とか抵抗で計算できます。電磁気学です。

これは高校物理をちゃんと勉強した人は直感的に分かるはず。

この  $A$  を求めることを ForwardSolution という感じに言います。掛け算を求めるのですね！

ここから  $X = A^{-1}Y$  という風に変えれば  $X$  を計算できます。

これを InverseSolution と言い、 $A^{-1}$  のことを

InverseOperator と言います。割り算ですね！この InverseOperator を求めることが当面の目標です。

手順としては以下のとおりです。

掛け算を作る

まずは、掛け算を作るために、脳の中の位置情報、センサーの位置情報、そして、  
その両者がどのように重なっているかの位置関係を求める必要があります。

必要なのは

- 抵抗値
- 脳みその形
- センサーのセットの形
- 脳みそとセンサーセットの相対的な位置
- それぞれをまとめる方程式

これだけあれば掛け算を作れます。具体的には、下記です。

1. MRI から脳の形を取ってきて、骸骨の抵抗とかも加味して計算できる形にする。  
これを BEMmodel と呼ぶ。これを使って掛け算の形にする。
2. 脳の形から「推定する脳部位の位置」を特定する。  
この脳内の位置情報をソーススペース (Source space) と呼ぶ。  
(鶴亀算の鶴と亀のいる場所を計算する)
3. センサーの位置情報。MNE では Montage クラスのオブジェクト。  
ただし、脳波計によっては Raw の中に組み込まれていて不要なこともある。
4. 「推定するべき脳の部位」と EEG/MEG のセンサーの位置をすり合せて  
両者の位置関係を求める。この作業は手動で行われる。(超絶めんどい)  
この重ね合わせ情報は trans というファイル形式で保存される。

- 
5. 脳の部位情報と頭の形情報とセンサーの位置から、  
脳活動によってどのようにセンサーに信号が届くかを計算する。  
これを脳磁図における順問題 (forward solution) という。  
これにより、掛け算が求められる。

## 割り算を作る

次に割り算を作ります。MNE や sLORETA や dSPM といった何やら難しげな手法は、この割り算を作るときのやり方の違いなのです。

1. 綺麗な割り算をするための Covariance matrix を作る (理屈は後述)。
2. 上記の脳部位とセンサーの関係性から、特定の脳部位での電源活動の波形を推定する。  
これを脳磁図における割りざ...逆問題 (Inverse solution) という。  
割り算は逆数の掛け算と同じであるから、掛け算に置き換えられる。  
この時の逆数のことを InverseOperator という。  
InverseOperator を滑らかにするために CovarianceMatrix を使う。  
この割り算に決まった解答はない。「最も良い解を得る方法」が幾つか提案されている。
3. 脳全体で推定した波形のうち、欲しいものをとってくる。

本当にこれだけ。なんと、現時点での脳の電気活動のソース推定とは割り算なのであった！

簡単に言いましたが、これが割り算である事を数学的にちゃんと理解するにはラグランジュの未定乗数法によって導かれる行列の微分方程式を解かねばなりません。あとでかーく触れます。

その後は色々なストーリーがあるでしょう。

## その後のストーリー

- 推定された波形を wavelet 変換する。
- PSD や ERP をしてみる。
- 脳の各部位のコネクティビティを算出する。
- 何か僕達が思いつかなかった凄いアイデアを実行する。

などなど。でははじめましょう。

---

## 手順 0、freesurfer

脳画像を解析したと思いますが、その脳画像のチェックはしてください。  
ダメダメなやつになっていることもありますので。  
ダメダメだったらトラブルシュートしなければなりません。  
無いなら標準脳でも出来なくはないです。

## 手順 1、trans

まず、脳とセンサーの位置をすり合わせておきましょう。  
GUI での操作となります。2通りの動かし方があります。  
下記のコードを実行すると画面が立ち上がります。

python で

```
1 from mne.gui import coregistration
2 coregistration()
```

bash で

```
1 mne coreg
```

mne coreg コマンド簡単ですね！ 公式でも mne coreg がオヌヌメと書いてありました。  
あんま変わらないとは思いますが。

subject とかの path を指定しない場合は、GUI 上で指定することになります。  
もし 0 から立ち上げた場合、山のようにある MRI の subject から該当の  
subject を探さねばならなくなります。重いので指定してあげたほうが楽です。

python の関数に色々入れてから起動すれば、既にデータが読み込まれているので、楽です。

```
1 coregistration(subject = subject,
2                 subjects_dir = subjects_dir,
3                 inst = file_path)
```

inst は eeg や meg データ…raw でも epoch でも良いらしいですが、どれかを指定して下さい。  
mne coreg の場合は、mne coreg -h とでもやって help を見てください。

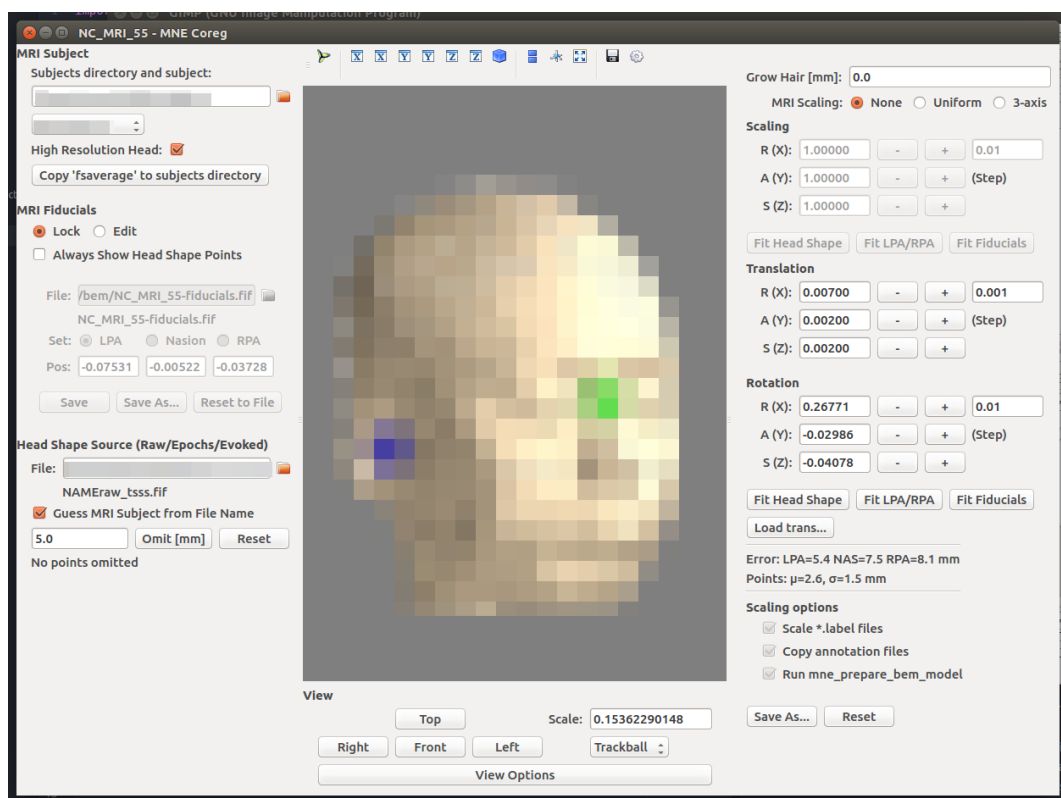


Figure 18: mne coregistration の画面。最悪の苦行。

手順はこうです。

1. 必要ならば、MRI の subject を読み込む
2. 必要ならば、fif ファイルを読み込む
3. 左側、set のところで耳と眉間の位置を入力  
(MEG ならスタイラスでポチるところ)
4. そのの一寸上の所、lock をポチる。
5. 面倒なら右側、Fit LPA/RPA ボタンとかを押す。
6. 表示された黄土色の生首をマウスでグリグリしながら、  
右上の ± ボタンを押して調整。
7. ちゃんと fit したら右下の save as ボタンを押して保存。

あとで、保存した trans を

```
1 from mne import read_trans
2 trans = read_trans('/Users/hoge/fuga/trans.fif')
```



---

みたいな感じで読み込んで使います。

右上のボタンを押した場合は黄土色の生首の大きさが変わってしまうので、freesurfer の subject に別名をつけて保存する必要があります。

ほかに注意点として、脳波とかの場合は表示が projection モードになっていたりして見にくかったりするかもです。色々調整してみてください。

## 手順 2、BEM 作成

脳からセンサーまでの抵抗を計算せねばなりません。

上記の通り、MRI から抽出してくる形データから BEM モデルを作ります。

BEM は脳の全体を包み込むサランラップみたいなデータになります。<sup>60</sup>

頭蓋骨とか皮とか、そういう抵抗が強いものを考慮するために、BEM は三枚一組で出力されます。実装上は 3 枚あるということを意識しなくても大丈夫です。

ちなみに、サランラップみたいなものを使わない方法もあります。

球形を想定してやる方法ですね。今回は書きません。

作するためには freesurfer による解析データが必要となります。

freesurfer を既に使っているなら Subject 関連は既に馴染んだ言葉でしょうか？

もちろん SUBJECT や SUBJECTS\_DIR は読み替えてください。

```
1 mne watershed_bem -s subject -d subjects_dir
```

これにより、freesurfer のサブジェクトの中に BEM が作成されました。

再び python に戻り、下記を入力してみてください。

```
1 from mne.viz import plot_bem
2 plot_bem(subject=subject,
3           subjects_dir=subjects_dir,
4           brain_surfaces='white',
5           orientation='coronal')
```

これで BEM が表示されるはずです。

---

<sup>60</sup> というか、本当はこんな言葉の使い方は間違ってる気がする。BEM は本当は Method の名前です。

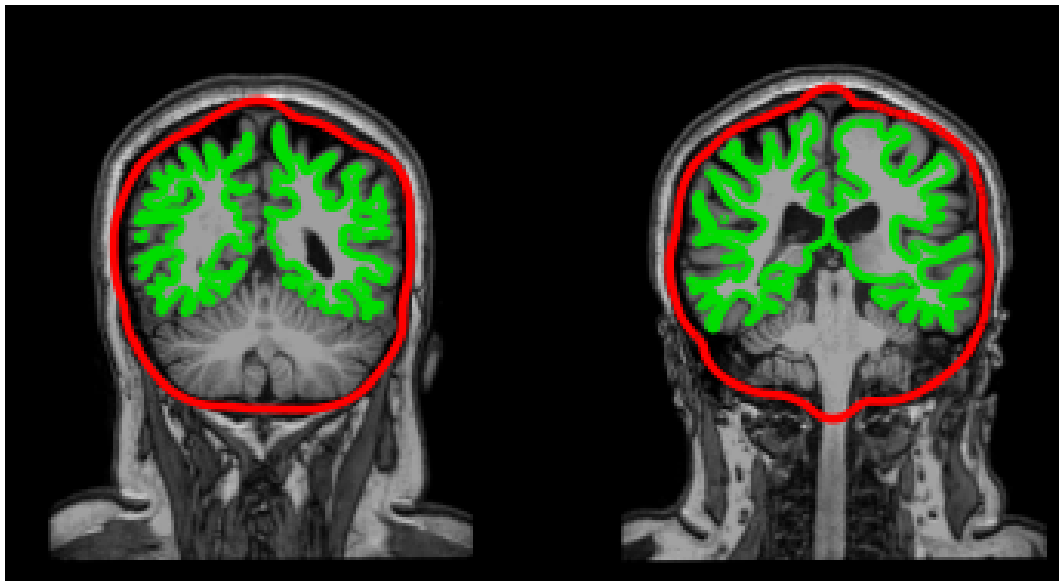


Figure 19: BEM の図示。

もし、標準脳を使うなら、以下のコマンドをターミナルから叩いて下さい。

```
1 mne coreg
```

gui の画面が現れると思います。

'fsaverage → SUBJECTS\_DIR' というボタンを押して下さい。

freesurfer の標準脳である fsaverage が現れます。

以降、subject には fsaverage を入れると標準脳を使うことになります。

### 手順 3、ソーススペース作成

脳内の位置情報を作りましょう。

脳磁図で見れる空間のうち、どの部分の電源を推定するかを設定する必要があります。

その設定がソーススペースです。subjects\_dir は環境変数に設定していれば要らないです。

環境変数ってのは bashrc とか bash\_profile とかに書くやつです。一応前述しています。

```
1 from mne import setup_source_space
2 src = setup_source_space(subject=subject,
3                           spacing='oct6',
4                           subjects_dir=subjects_dir)
```

---

もちろん、標準脳が欲しい場合は黙って `fsaverage`。暫く待ちます。

これで、`src` という変数にソーススペースが入りました。

さて、見慣れぬ単語が出てきました。`oct6` とは何でしょうか？ それはここに書いてあります。

<http://martinos.org/mne/stable/manual/cookbook.html#setting-up-source-space>

ソーススペースを作るためには計算上正十二面体や正八面体で  
区画分けするので、その設定ですね。

やり方によってソーススペースの数も変わるみたいです。

臨床的に意味があるかはわかりません。

標準脳を使う場合は '`fsaverage`' を `subject` に指定して下さい。

ない場合は手順 2 の `mne.gui.coregistration()` でボタンを押して下さい。

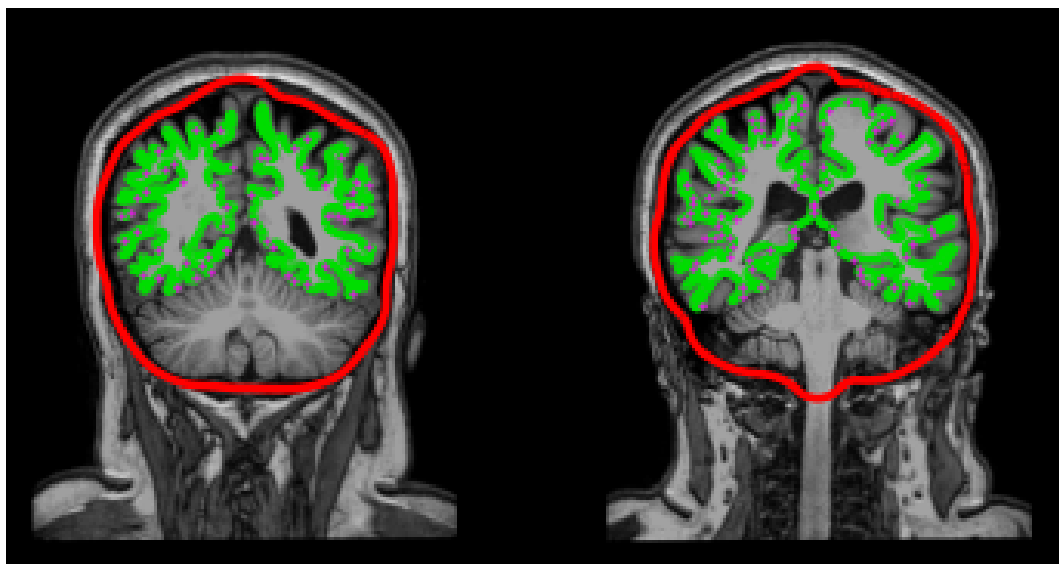


Figure 20: ソーススペースの図示。小さい点々がソーススペース。

#### 手順 4、順問題

まずは掛け算を作ります。

先程作った BEM は 3 枚あります。EEG の場合は 3 枚必要です。何故なら、磁力と違って電力は  
脳脊髄液と頭蓋骨と頭皮を素通りしにくいからです。だから、BEM を三枚仮定するのです。

MEG の場合は一枚だけで十分だそうです。

では、BEM で順問題を解く準備をしましょう。

---

```
1 from mne import make_bem_model, make_bem_solution
2 conductivity = (0.3,)
3 model = make_bem_model(subject='sample',
4                         ico=4,
5                         conductivity=conductivity,
6                         subjects_dir=subjects_dir)
7 bem = make_bem_solution(model)
```

これにより、BEMを読み込み、順問題解きモードに入りました。

icoはどの程度細かく順問題を解くかの数値です。icoの数字が高いほうが詳しいです。

conductivityは電気や磁力の伝導性のパラメータです。

EEGの場合はこれが(0.3, 0.006, 0.3)とかになります。

では、先程作った色々なものと組み合わせて順問題を解きます。

```
1 from mne import read_trans, make_forward_solution
2 trans = read_trans('/hoge/fuga')
3 mindist = 5
4 fwd = make_forward_solution(raw.info,
5                             trans=trans,
6                             src=src,
7                             bem=bem,
8                             meg=True,
9                             eeg=False,
10                            mindist=mindist,
11                            n_jobs=4)
```

ここまでやった方にとって、上記のパラメータはだいたい分かるでしょう。

mindistは頭蓋骨から脳までの距離です。単位はmm。

ここで使うのはraw.infoです。epochs.infoでもいいかも。

## 手順 5、コヴァリアンスマトリックス関連

MNEによる推定にはcovariance matrixというものを使って割り算を綺麗にやります。

数学分かる人向けに書きますと、

「MNEはベイズのMAP推定の一種で、事後分布に脳波の分散共分散行列を使う」  
ってことです。

これにはMEGを空撮りした空データや、刺激提示されていないときのデータなどを使います。

下記で計算します。

```
1 from mne import compute_covariance
2 cov = compute_raw_covariance(raw_empty_room,
3                             tmin=0,
```

---

```
4                                     tmax=None)
```

ちなみに、刺激提示されていないときの計算は下記のとおりです。

```
1 from mne import compute_covariance
2 cov = compute_covariance(epochs,
3                           tmax=0.,
4                           method='auto')
```

ちなみに、この method=auto というのは MNE に実装された新しいやり方だそうです。

tmax=0 にしているので、刺激が入る前までの波を取り除きます。

つまりベースラインコレクションみたいな感じになるのです。

ちなみに、epochs で covariance...特に auto ですと結構重いです。

auto の場合、クロスバリデーションまでやってくれて凄いみたいですね。

その代わり、計算の重さが酷いです。

## 手順 6、逆問題

最終段階、割り算です。順問題と covariance matrix を組み合わせて割り算の形にしましょう。

下記のとおりです。

```
1 from mne.inverse import make_inverse_operator
2 inverse_operator = make_inverse_operator(epochs.info,
3                                         fwd,
4                                         cov,
5                                         loose=0.2,
6                                         depth=0.8)
```

inverse\_operator と言うのは何かというと、逆問題を算出するための式です。

この inverse\_operator を作るために頑張ってきたと言っても過言なしです。

ここで、第一引数に epochs.info を入っていますが、info なら

raw でも evoked でも良いはずです。

さて、ここで loose と depth という耳慣れぬ物が出てきました。一寸大事なパラメータです。

脳内の電流源推定と言っても、電流の向きを考慮しなくてはならないわけです。

loose はその向きがどのくらいゆるゆるかの指標です。

脳磁図はコイルで磁場を測る関係上、脳の表面と水平な方向の成分を捉えやすいように出来ています。

でも、脳波複雑だから完全な水平ってないよね？ どのくらいのを想定する？

という風なパラメータです。loose は 0~1 の値をとりますが、loose が 1 というのは超ユルユル、どの方向でも良いですよということです。

---

ちなみに、`loose` が 0 の時は一緒に `fixed` を `True` にする必要があります。  
`fixed` が `True` の時は、`MNEpython` が脳の形に沿って自動調整してくれます。

`depth` は何かというと、どのくらい深い部分を見たいか、です。  
`MNE` という計算手法は脳の表面の情報を拾いやすい偏った計算方法です。  
故に、深い部分に対して有利になるようにする計算方法があります。  
`depth` を設定すると、脳の深い所を探れるわけです。  
`depth` を `None` に設定すると、ほぼ脳の表面だけ見ることになります。

他に `limit_depth_chs` というパラメータもあります。これを `True` にすると、完全に脳の表面だけ見ます。

即ち、マグネトメータをやめて、グラディオメータだけで見るのです。

ここまで長かったので保存しておきましょう！

```
1 from mne.inverse import write_inverse_operator
2 write_inverse_operator('/home/hoge/fuga', inverse_operator)
```

この `inverse_operator` が作れたら、あとは色々出来ます。

## 手順 7 ソース推定

まずは、ソース推定をやってみましょう。

```
1 from mne.minimum_norm import apply_inverse
2 source = apply_inverse(evoked, inverse_operator, 1 / 9)
```

ちなみに、ここでは `evoked` を使っていますが、  
`epochs` なら `apply_inverse_epochs`、`raw` なら `apply_inverse_raw` です。

```
1 from mne.minimum_norm import apply_inverse_epochs
2 source = apply_inverse_epochs(evoked, inverse_operator, 1 / 9)
```

```
1 from mne.minimum_norm import apply_inverse_raw
2 source = apply_inverse_raw(evoked, inverse_operator, 1 / 9)
```

ちなみに、`epochs` の場合は `list` を返します。  
`list` の内容は `SourceEstimate` , `VectorSourceEstimate` , `VolSourceEstimate` です。  
`SourceEstimate` が `Bem` ベースの結果ですね。

一旦、これを図示してみましょう。

```
1 source[0].plot(time_viewer=True)
```

---

やりました！ これぞ、MNE-python の真髄、割り算であります！

この `time_viewer=True` は時間を追って見ていきたい時につけると良いオプションです。

さて、これで出てきた `source` の中に `data` という変数があります。

まさに膨大な数です。脳内の膨大な場所について電流源推定したのです。

これは、一つ一つが脳内で起こった電流と考えて良さそうです。細かい所は公式サイト見てください。

こんな膨大な数列があっても困りますよね？ 脳のどこの部位なのかわかりませんし。

そこで、`freesurfer` のラベルデータを使います。

それによって、脳のどの部分なのか印をつけてやるのです。

## 手順 8、前半ラベル付け

`freesurfer` にはいくつかのアトラスがあります。アトラスとは、地図みたいなものですね。

詳しくはここをみて下さい。

<https://surfer.nmr.mgh.harvard.edu/fswiki/CorticalParcellation>

`desikan atlas` とか `Destrieux Atlas` とか色々ありますよね。

こういうのを読み込まねばなりません。ターミナルでこのように打ってみて下さい。

```
1 ls $SUBJECT_DIR
```

もし `freesurfer` を既に動かしているならば、解析済みの MRI が沢山あるはずです。

サブジェクトの中身には `label` というディレクトリがあります。

この中にいっぱいそういう `freesurfer` のアトラスが入っています。

ファイルの形式には二種類あり、`annot` 形式と `label` 形式があります。

`annot` 形式は新しく開発されたアトラスが入っていて、`label` 形式はブロードマンと思います。

`annot` 形式の内容はこのように読みます。

```
1 from mne import read_labels_from_annot
2 labels = read_labels_from_annot(subject, parc='aparc')
```

詳しくは公式サイト (ry

この関数はラベルのリストを読んできます。

`parc` は一応 `aparc` と `aparc.a2009s` というに種類のものがあって、

このうち後者のほうが新しい分け方らしいです。どっちがいいとかはない。

こうして読んだら、`label` のリストが出てきます。単体の `label` は下記で。

```
1 from mne import read_label
2 label = read_label(filename, subject = None)
```

---

これで label を読み込めたら、次はそれを当てはめることになります。

Label オブジェクトには name というメンバー変数があるので、自分がどれを見ているのかは確認できません。

が、name だけでは実際はイマイチわかりません。

freesurfer の元論文を見てからどこかを見つけていきましょう。

ここ、どのラベルがそれなのかサクッと調べたいですね？

filter 関数やリスト内包表記でも使うのがいいかと思います。

```
1 # filter 関数
2 print(list(filter(lambda label: 'transv' in label.name, labels)))
3 # List 内包表記
4 print([label.name for label in labels if 'transv' in label.name])
```

手順 8 後半、label 当てはめ

では、label をベースにデータを抜き出しましょう。

```
1 from mne import extract_label_time_course
2 source_label = extract_label_time_course(source,
3                                         labels,
4                                         src,
5                                         mode='mean_flip')
```

ここでは source がソースのデータ、src が左右半球のソーススペースのリストです。

mode はいくつかあります。

mean: それぞれのラベルの平均です。これを使うのが普通でしょうか...

mean\_flip: 特異値分解を使ってベクトルが違うやつも取り出すのです。

pca\_flip: PCA を使って取り出してくるのです。

max: ラベルの中で最大の信号が出てきます

これで脳内の波形が取り出せたわけです。

これで、色々出来ます。なにしろ、今まで wavelet 等していたわけですから。

### 三次元 plot

でも、これだけじゃダメですね。きちんと見てみないと実感が無い。

ここで、mayavi と pysurfer が登場します。mayavi は三次元を表示するパッケージ、

pysurfer は freesurfer を mayavi を使って表示するパッケージですね。

下記は hoge 番目の Label を見えています。



---

```

1 from mne.minimum_norm import apply_inverse_epochs
2
3 hoge = 4
4 source = apply_inverse_epochs(evoked, inverse_operator, 1 / 9)
5 brain = source[0].plot(subjects_dir=subjects_dir, time_viewer=True)
6 labels = read_labels_from_annot('fsaverage', subjects_dir=subjects_dir)
7 brain.add_label(labels[hoge])

```

これにより、きちんと脳の部位にラベルをつけたまま脳活動を表示できます。

でも、実はこの方法だけじゃラベルつけたまま時系列表示できません。

時系列表示します。

```

1 from mayavi import mlab
2 import surfer
3
4 hoge = 4
5 scene = mlab.figure()
6 source = apply_inverse(evoked, inverse_operator, 1 / 9)
7 labels = read_labels_from_annot('fsaverage',
8                                 subjects_dir=subjects_dir)
9 brain = surfer.Brain('fsaverage',
10                     'lh',
11                     'inflated',
12                     subjects_dir=subjects_dir,
13                     figure=scene)
14 brain.add_label(labels[hoge])
15 source.plot(subjects_dir=subjects_dir,
16             time_viewer=True,
17             figure=scene)

```

何やってるかと言うと、mayavi で一旦 canvas 的なものを作って、

そこに一寸ずつ書き加えているイメージです。

まず、適当にソース推定します。そして、その推定を plot...の前に、

先に surfer.Brain で脳の画像を作って、add\_labels で脳の特定期位に色を塗ります。

手順前後はいけません！最後にソース推定を plot して終わり。

### 三次元 movie

三次元で表示したものを動画として保存してドヤ顔しましょう。

先程のスクリプトに scene というのがありましたね？ それを使います。

それを該当の箇所に入力して save\_movie だけです。

```

1 surf = source_estimate.plot(subject['mri'],
2                             surface='inflated',

```

```
3             hemi='both',
4             figure=scene)
5 surf.save_movie('hoge.gif')
```

ね、簡単でしょ？ 今回は gif にしてみました。

すげーヌルヌル動くんですが、同人誌の前の君に見せられなくて残念です。

## その後の楽しみ 1、ソースベース **wavelet**

ソースベースで wavelet やりたいなら、特別に楽ちんな関数が実装されています。

induced power と inter trial coherence を算出する関数は下記です。

※ label を選ばなければ激重注意！<sup>61</sup>

```
1 from mne.minimum_norm import source_induced_power
2 induced_power, itc=source_induced_power(epochs,
3                                         inverse_operator,
4                                         frequencies,
5                                         label,
6                                         baseline=(-0.1, 0),
7                                         baseline_mode='zscore',
8                                         n_cycles=n_cycles,
9                                         n_jobs=4)
```

基本は以前 wavelet 変換で行った事に、いくつか追記するだけです。

やり方は色々あります。label は freesurfer のラベルデータです。

これで算出された wavelet 変換の結果の取扱は、前に書いた wavelet 変換の結果と同じです。

この関数は、割り算のあと wavelet 変換してるのではなくて、

wavelet 変換したあと割り算してるっぽいんです。<sup>62</sup>

これ、コネクティビティを計算する時と手順が逆なんですよ。

なんだかよく分かりませんね…

あと、InducedPower と言いながら、多分 TotalPower を見ていたりするので、

これを計算する前に epochs に関して以下をやっておく必要があったりします。

```
1 epochs.subtract_evoked()
```

これやると、多分 itc はおかしくなっちゃうので、上記は実は 2 つ同時に計算は出来てない。

<sup>61</sup> label を選ばない場合これは激重です。何故なら 306 チャンネルの MEG からソースに落とし込むと計算方法によっては 10000 チャンネルくらいになります。ROI を絞ったとしても「人数 × タスク × ROI の数 × EPOCH の数」回 wavelet 変換して power と itc に落とし込むのですから…途方もない計算量です。label を選びましょう。

<sup>62</sup> 僕がソースコードを読んで知ったので、読み損ないはあるかも？

---

## その後の楽しみ 2、ソースベース **connectivity**

ソースベースでコネクティビティ出来ます。

mne\_connectivity というパッケージを使います。

これ、元々 mne と同じパッケージだったのが別になったやつです。

```
1 from mne_connectivity import spectral_connectivity_epochs
2 con = spectral_connectivity(
3     source_label, method='coh', mode='multitaper',
4     sfreq=500, fmin=30,
5     fmax=50, faverage=True, mt_adaptive=True)
```

使い方はセンサーベースコネクティビティと同じです。

この場合、さっき計算して出したラベルごとのデータと、

ラベルリストを放り込めば、先述の 5 つの変数が出てくるので楽ちんです。

## コラム 3-markdown で同人誌を書こう！

```
1 皆さんもこのような科学系同人誌書きたいですよね？
2 書いてコミケにサークル参加したいですよね？
3 **難しいLaTeXなんて覚えなくても大丈夫。そう、markdownならね！**
4 LaTeXは添えるだけ。手順は下記。
5
6 macならmactexとpandocをインストールします。
7 ubuntuやwindowsならTeXliveをインストールします。
8 mactexはググれば出てきます。pandocは
9 brew install pandoc
10
11 ubuntuなら
12 sudo apt install texlive-lang-japanese
13 sudo apt install texlive-xetex
14 sudo apt install pandoc
15
16 これでpandocでmarkdownからpdfに変換できるようになります。
17 例えばDoujinshi.mdというmarkdownファイルを作って
18
19 pandoc Doujinshi.md -o out.pdf \
20 -V documentclass = ltjarticle --toc --latex-engine = lualatex\
```

---

```
21 -V geometry:margin=1in -f markdown+hard_line_breaks —listings
22
23 四角で囲われているところはコードの引用の書式に従って書いた後、
24 コードの上の``の末尾に{frame=single}と書き加えてください。
25
26 これで同人誌に出来るPDFになります。詳しくはググってください。
27
28 何故僕が同人誌を書くかって？
29 君達が同人誌を書いてくれなかったからだ！
30 今度はぜひ君の素晴らしい同人誌を僕に見せてくれよな！
```

---

## 理論編

ここまで実践を行ってきましたが、ブラックボックスでした。  
でも、理解する努力はすべきでしょう。ここから理論編に入ります。

内容は高校数学でギリギリやれますし工学部二年生は大抵理解しています<sup>63</sup>が、  
初心者には結構難しいです。また、本書はあんま頭のいい人が書いてるわけじゃないです。  
だから、ここでは物凄くざっくり (やや不正確な) 解説をします。

### 初心者のための波形解析

脳波には波形解析が必須でしょう。波だからね。  
波形解析としては下記を行います。

- フーリエ変換
- ヒルベルト変換
- 非直交 Wavelet 変換
- 直交 Wavelet による多重解像度解析

実は他にもあるのですが、今の所はこれだけ。必要な知識は高校数学。  
高校を卒業していない人と忘れた人は下記辺りを参考にすればよいかと。  
<https://manabitimes.jp/math>

### 波形解析で得たいものと、必要な変換

脳波や脳磁図を解析して貴方は何をgetたいでしょうか？  
脳波や脳磁図の特定の波長...例えば $\alpha$ 波、 $\beta$ 波、 $\gamma$ 波等の強さを求めたいですね！ それもミリ秒単位で！  
また、どのくらい位相が揃っているのかも見たいです。波と波の関係性とかもみたいです。  
なので、必要なのは下記を兼ね備えたデータです。

- 波の強さ
- 波の位相
- 波の位置 (ミリ秒単位)
- 注目した周波数以外は無視できる
- 波と波の関係性

---

<sup>63</sup>工学部生は目をそらさないでいてもらおうか

---

波というものは位相があります。出っ張りもあれば凹みもあって、しかもいろいろな波が重なっていたりして定量化しにくいです。特定の周波数の波の出っ張りや凹みを両方同じように評価するにはどうすればいいでしょう？

これらを実現するのが周波数解析です。本当は色々な種類の周波数解析があるのですが、ここでは以下の3つの解析を解説しようと思います。

- フーリエ変換
- Wavelet 変換
- ヒルベルト変換

フーリエ変換とは

調べたい波を、全て  $\sin$  と  $\cos$  だけで表してしまうやりかたです。  
周波数ごとに長い  $\sin, \cos$  波を大きくしたり小さくしたりしながら当てはめます。  
要するに、周波数ごとに

$$A\cos x + B\sin x$$

という形に直していきますが、実はもう一寸スマートなやり方があって

$$A\cos x + B\sin x$$

という複素数の形式(!?) に落とし込んでいくのを目指します。

理由は波の強さ、位相を観察するためには複素数のほうが都合がいいからです。

### ShortTime フーリエ変換

$\sin$  や  $\cos$  は未来永劫続く波です。そして、実際の波は未来永劫続く波ではありません。

無限に続く波と有限の波を掛け算するのは厳しいですね？

そこで、有限の波を「この波は実は繰り返し起こってて、実は無限なんだ！」

と嘘こくのが ShortTime フーリエ変換です。

有限の波の最初と最後がブツンと切れると凄く都合が悪いので、

そこは緩やかに減衰させておいて (Taper) その波が続くものと仮定して変換していきます。

要するに、嘘っぱちですが、大まかな振幅とかは解ると思います。

### Wavelet 変換とは

$\sin$  も  $\cos$  も未来永劫絶対に減衰しない波であるため、不規則な波のフーリエ変換は元来きついです。

「一部を無理やり切り取ってきて、切り取った波が永遠に続く波と

---

想定した上で変換すること」は一応可能でも、実は解析結果が凄く荒くなります。<sup>64</sup>

そこで、フーリエ変換にほんの少しの細工を施して時間軸を加味したのが

Wavelet 変換です。

実は、これにも色々種類があって、主だった所は

非直交連続 Wavelet 変換と、直交多重解像度解析です。

まず、非直交の話をしてします。

理解するためにはフーリエ級数を理解する必要があります。

## ヒルベルト変換

上記とはちょっと違った風な解析ですが、実用上の性質や使い方は wavelet 変換によくにています。

wavelet 変換は一つの波から「実数部分と虚数部分を抽出してくる変換」

と言えますが、ヒルベルト変換は「元の波から架空の虚数軸部分を作っちゃう変換」みたいなイメージです。

架空の虚軸を出してくればパワーも位相も出し放題ですね。

内容的には逆フーリエ変換を足し算とすると、ヒルベルト変換は引き算みたいなものです。

計算の方法は簡単で、中学校の頃に習う  $1/x$  という風な双曲線を元の波に掛け算してあげるだけです。

この理屈に関しては後で述べます。

## フーリエ級数

早速ですが、フーリエ変換の時、元の波は下記のように表します。

$$f(x) = a_0 + \sum_{k=1}^{\infty} \left( a_n \cos \frac{2\pi n t}{T} + b_n \sin \frac{2\pi n t}{T} \right)$$

これは何かというと、波を変換している式です。解説します。それぞれ...

- $f(x)$ : 元の波を表す式
- $a_0$ : 波のベースの高さ
- $t$ : 時間
- $n$ : 解析したい周波数に対応した変数
- $T$ : 周期 (任意の定数)
- 右辺: 特定周波数の波を表す式

---

<sup>64</sup>短い時間で解析すると時間的には細かく解析できるけど、周波数の詳しい所が見れなくなります。これをフーリエ変換の不確定性原理と言います。量子力学にも出てくる言葉ですが、実は同じことなのです。

---

かなり複雑な式っぽいですが、 $\sum$ の中を御覧ください。  
解析したい周波数が変数として与えられてます。これは、特定の周波数だけにしか対応しないのです。  
大抵の波はこの単純な波の足し算で説明することが出来るのです。  
実はこの方程式を積分したりして解けば、各  $n$  に対する  $a$  も  $b$  も算出することが出来ます。  
右辺全体をフーリエ級数、 $a$  と  $b$  をフーリエ係数と言います。  
この式によって規則的なほとんど全ての波を別形式に書き換えられます。  
凄いですね！ ですが、それだけでは面白くありません。  
波をこのように別の式に書き換えた所で僕達がほしい  
「波の強さ」「波の位相同期性」「波の位置」の情報がないからです。  
なので、上の式を複素数を使って変換して公式を求めます。

### 複素フーリエ級数

高校数学の複素数平面を覚えているでしょうか？ 全ての二次元の座標は複素数平面上で下記の式によって表現できます。

$$r(\cos\theta + i\sin\theta)$$

高校時代に習った極座標みたいな何かですね！ おぼろげに覚えているかも知れません。  
最終的にはこの様な形に成れば、波の強さも位相も分かるはずです。

さて、貴方はオイラーの公式をご存知でしょうか？  
この世で最も美しく偉大な公式の一つです。(実際人気が高い)  
高校数学でギリギリ出てきます。下記です。

$$e^{i\theta} = \cos\theta + i\sin\theta$$

何度見ても美しい公式ですね。<sup>65</sup> 崇めて下さい。証明はググってください。  
これは美しいだけでなく役に立ちます。まずはこのオイラーの公式による変換を考えます。  
端折りますが、オイラーの公式を変形すると  $\sin$  と  $\cos$  を  $e$  と  $i$  で表現することが可能です。

$$\cos\theta = \frac{e^{i\theta} + e^{-i\theta}}{2}$$

---

<sup>65</sup> ちなみに、 $\theta$  が  $1$  の場合はオイラーの等式と言います。美しいです。



---

三角関数なんかいらなかったんや。

というわけで、フーリエ級数の式から  $\sin$  と  $\cos$  を排除できます。式を弄くり倒すと、

$$f(x) = \sum_{n=-\infty}^{\infty} C_n e^{i\pi n t/T}$$

と変形できます。 $C_n$  は変換  $a, b$  を複素数でまとめたものです。

この式の右辺を複素フーリエ級数といいます。

フーリエ変換とはフーリエ係数を求める計算のことです。

$f(x)$  は元の波、 $e^{i\pi n t/T}$  はオイラーの公式を見ると…半径 1 の円ですね。

ここで知りたいのは  $C_n$  です。 $C_n$  を求めるにはやはり高校数学によって下記のように書き換えられます。

$$C_n = \frac{1}{T} \int_{-T/2}^{T/2} x(t) e^{-2\pi n t/T} dt$$

$f(x)$  は脳波や脳磁図の結果ですので、フーリエ変換は

脳波や脳磁図に絶対値 1 の複素数を掛け算して積分する操作といえそうです。

plot すると勉強になるかも知れません。i 乗とか実際に計算するのはアレなので、実際の計算をする時もオイラーの公式で展開…と言いたいところなのですが、

python3 なんかは複素数の演算ができるので、 $e^i$  を計算できたりします。凄いですね！！

というわけで、上の式をそのまま python3 流に書けばフーリエ変換は

自分でコーディングすることが出来るわけです。でも、僕は自分ではしないです。

既にそれをする為の最適化されたパッケージは開発されているからです。<sup>66</sup>

ところで、フーリエ変換には FFT という超速いアルゴリズムがあります。

これは畳み込みの定理を組み合わせることで wavelet 変換にも応用できます。

結果は変わらないので実際の計算をする時は使うといいでしょう。

---

<sup>66</sup>既に作られているものをもう一回作る無駄のことを、業界では「車輪の再発明」と言い、性能が落ちる車輪の再発明のことを「四角い車輪の再発明」といいます。

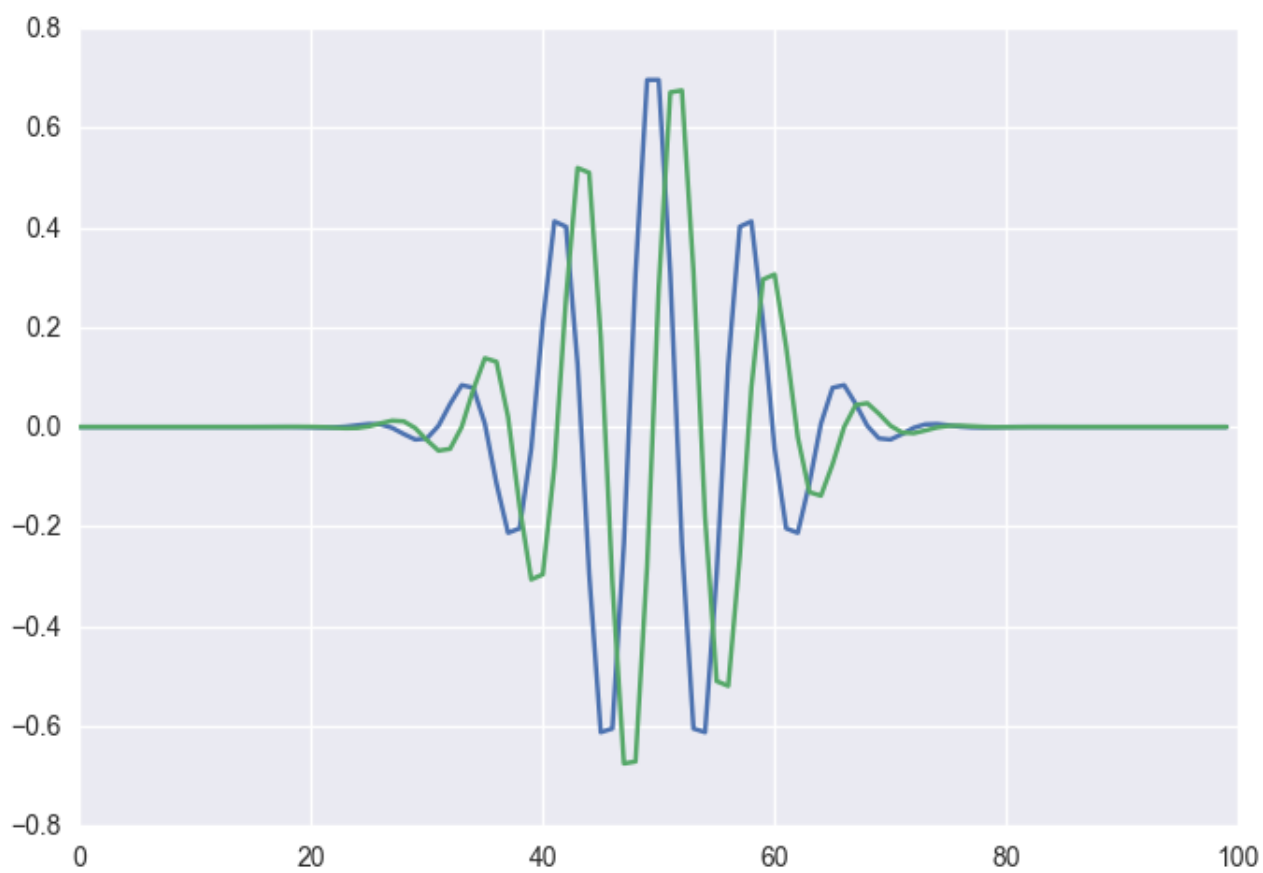


Figure 21: 再掲。Morlet Wavelet の 2D プロット

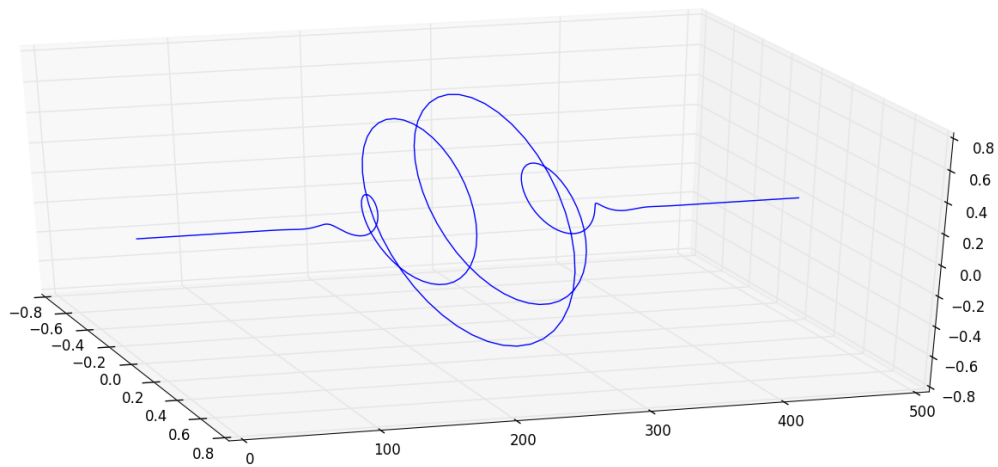


Figure 22: 再掲。3D プロット

結局何をしているのか

要するに

波 = 特定の周波数の波1 + 特定の周波数の波2 + 特定の周波数の波3...

特定の周波数の波 = 複素数の定数 \* 基準の波

という感じに変えていくだけの計算です。各周波数の数列さえわかれば波が表せます。

で、この定数を複素数にしたものが複素数版フーリエ級数。

複素数だから「定数」の絶対値や位相が計算しやすい、ということです。

さて、今あなたは周波数ごとに

$A + Bi$  または  $(r, \theta)$  という形の複素数を得ることが出来ています。

これをどう料理していくかが次の問題です。

ここまでくれば高校の複素数です。

## スペクトル解析

得たかったものは何だったか思い出しましょう。複素数から「絶対値、位相」が分かるというのは高校時代に数学をしたことのある人は明らかでしょう。

では、脳波の強さとは何でしょうか？ それはエネルギーみたいなものです。

即ち、絶対値の二乗に他なりません！

---

(物理の人に怒られる表現。実際はそう簡単でもない。)

脳波の位相はもちろん、上記の複素数の位相成分ですね。

フーリエ変換で出て来る数値 (フーリエなら係数に相当するやつ) は

スペクトルと業界 (脳波以外でも波を使う業界なら全て) で呼ばれています。

特に、自分自身に自分自身を掛け算したものの積分は「パワースペクトル」といいます。

$f(\omega)$  の絶対値の二乗が力 (パワー)、そして位相がそのまま位相です。

この、パワースペクトルの時間単位の平均値が

パワースペクトル密度 (PSD) と言われ、脳波解析の結果の一つです。

では、位相についてはどうするでしょうか？

一つの脳波の位相を取り出したところで意味はありません。

位相の良いところは他の波とリズムが揃っているか調べられる所にあります。

「毎回同じ位相を取り続けるかどうか」であったり、

「他の場所の位相とどのくらい差があるか」であったり、色々わかります。

毎回同じかどうかは PhaseLockingFactor とか InterTrialCoherence と呼ばれ、

それぞれ PLF、ITC と略されます。(意味は同じです)

他の場所の位相とどのくらい差があるかは PhaseLockingValue と呼ばれ、

PLV と略されます。(後で書きますが、PLV は頑健ではないので別の指標を使います)

さて、実際の計算は大したことはありません。複素数  $A + iB$  について考えてみましょう。

この絶対値は  $\$(A + iB)(A - iB)\$$  であることは自明です。

すなわち、複素共役同士を掛け算したものです。

これ、自分自身をかけ合わせると位相が  $\theta$  になりますが、

他の波の複素共役と掛け合わせると位相の引き算になるのです。

コネクティビティはこういうのを使っています。

さて、これまで「複素共役を掛け算して積分したもの」がいっぱい出てきました。

長いので、よく下記のように略されます。

$S_{xx}$  ( $x$  と  $x$ 、つまり自分と自分の関係、パワースペクトル)

$S_{xy}$  ( $x$  と  $y$ 、つまり自分と他人の関係、クロススペクトル)

以下、このように書いていきます。

そして **wavelet** 変換へ

一旦話を戻し、まずは Power を考えます。

フーリエ変換の時点では周波数ごとの複素数が一つずつありました。

これでは時間の次元が失われていますね。いつその Power だったんだよと思います。

---

この理由は、フーリエ変換に使う  $\sin$  と  $\cos$  が永遠に続く波であるからです。  
永遠に続く波を使えば、そりゃ時間軸は表現できるわけ無いです。

それを解決するのが wavelet 変換です。

時間軸を表現するためにものさしに使うものは減衰する波の必要があります。  
減衰させるには色々あるのですが、Gabor Wavelet では以下のようにします。

$$c\sigma\pi^{\frac{-1}{4}}e^{\frac{-1}{2}t^2}e^{i\sigma t}$$

この、左側がフーリエ変換のためのやつ、右に付け加えたのが減衰するやつ。

こんな感じの式に色々定数を付けたのが Gabor Wavelet です。

何故定数をつけるかというと、掛け算するときに掛け合わせるものの  
絶対値が 1 じゃないとエネルギーを計算するのが超面倒だからです。  
くるくる回りながら減衰する波です。

実際は、これは Wavelet としては結構お粗末なんだそうです。  
こいつに引き算を付け加えてスケールしても歪まないようにしたのが  
かの有名な MorletWavelet です。

$$c\sigma\pi^{\frac{-1}{4}}e^{\frac{-1}{2}t^2}(e^{i\sigma t} - \kappa\sigma)$$

$\kappa$  は  $\sigma$  から導かれる定数です。詳しくはググれ。

ちなみに、これは他にも色々あるやり方の一つです。

Wavelet 変換も計算結果が複素数平面として出てきますから、  
さっきと同じように Power と位相が分かるはずです。しかも、時間別に！

初めてフーリエ級数とか聞いた人はここまで読んで意味がわからなかったと思います。  
そんな人には高校数学の美しい物語の複素フーリエ級数関連の記事をお勧めします。

## wavelet 逆変換と bandpass filter

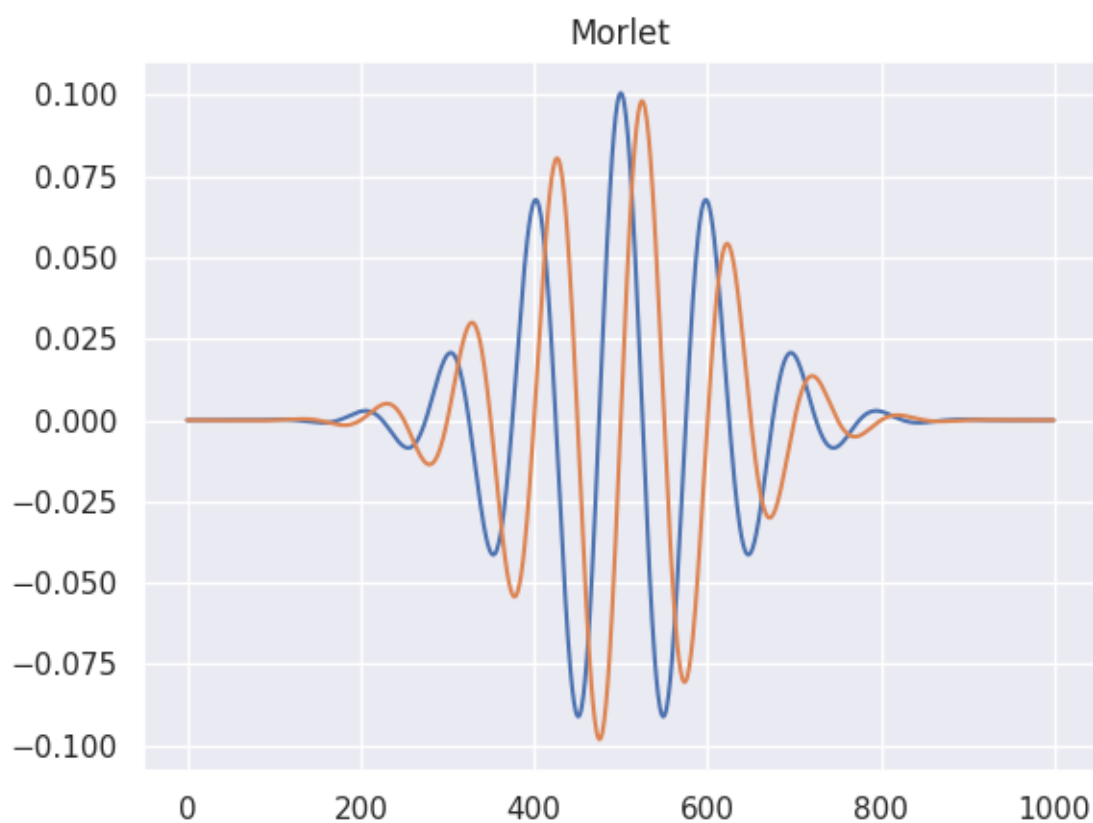
上記のように wavelet 変換は数値を出すのに応用できるのですが、  
これは実は bandpass filter モドキにも応用することが出来ます。  
何故なら、周波数を分けちゃう事ができるのと、Wavelet 変換は逆変換が出来るからです。  
手順としては、Wavelet 変換したものの中から欲しい周波数帯域を選んで、  
Wavelet 逆変換を行えば bandpass filter の一種みたいなものになります。

---

だから、意外とバンドパスフィルタを先につけないと駄目ということもないわけです。  
ちなみに、これは世間でよく用いられている bandpass filter とはちょっと違います。

**MorletWavelet** とは

$$\Psi(t) = c\sigma\pi^{-\frac{1}{4}}e^{\frac{-1}{2}t^2}(e^{i\sigma t} - \kappa\sigma)$$



青が実軸、橙が虚軸です。

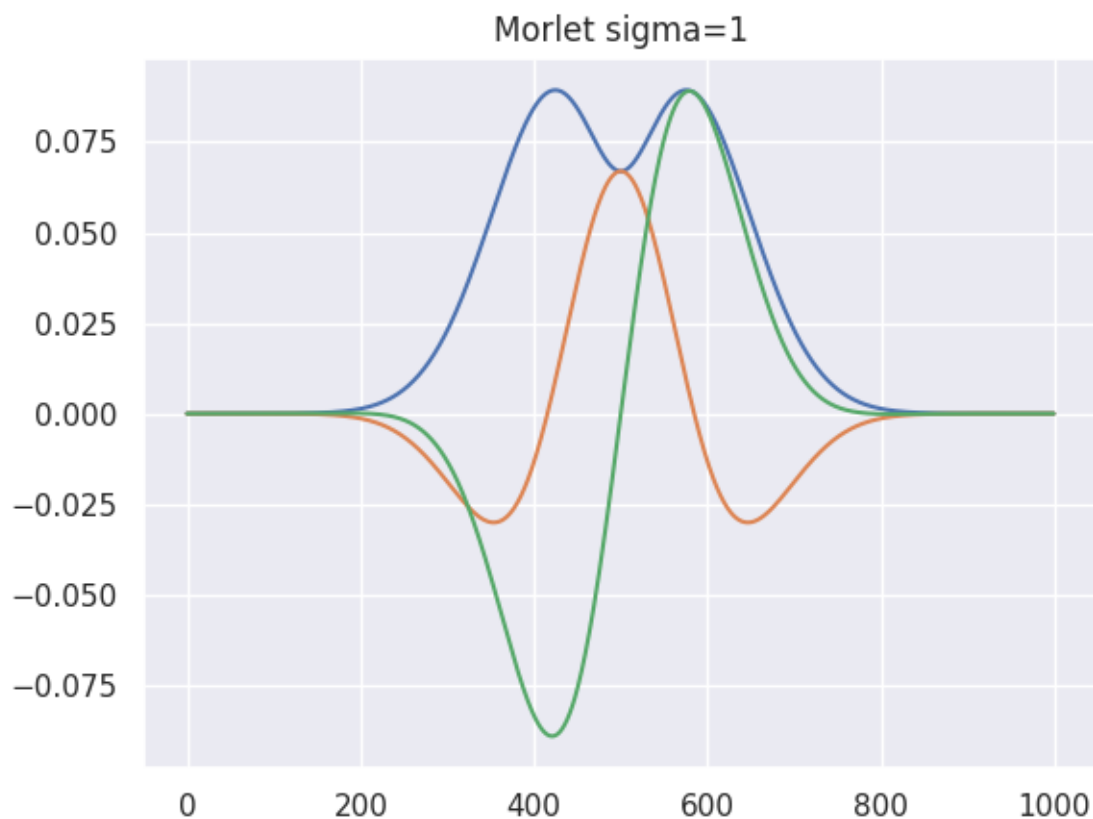
括弧の中に引き算が入りました。Morlet Wavelet です。[^morlet]

でも、これ、図は綺麗でも式の見た目的に凄く気持ち悪いですよね？

実際に  $\sigma$  を小さい値にしてみるとガウシアンな曲線がグチャッと潰れて

実軸小さすぎで虚軸が大きすぎな実に気持ち悪い状態になります。

その結果、パラメータによっては波の真ん中よりも周辺の方が過大評価されてしまいます。



青が絶対値、橙が実軸、緑が虚軸です。

フーリエ変換に時間軸にガウシアンな関数を畳み込んで時間軸を加え、  
歪みを取り除いたのに こうなってしまうては何のために  
ガウシアンにしたのかさっぱりわからないですね □

#### 気持ち悪さ対策

$\sigma$  を大きく取れば MorletWavelet と GaborWavelet は近似され、ほぼ重なります。

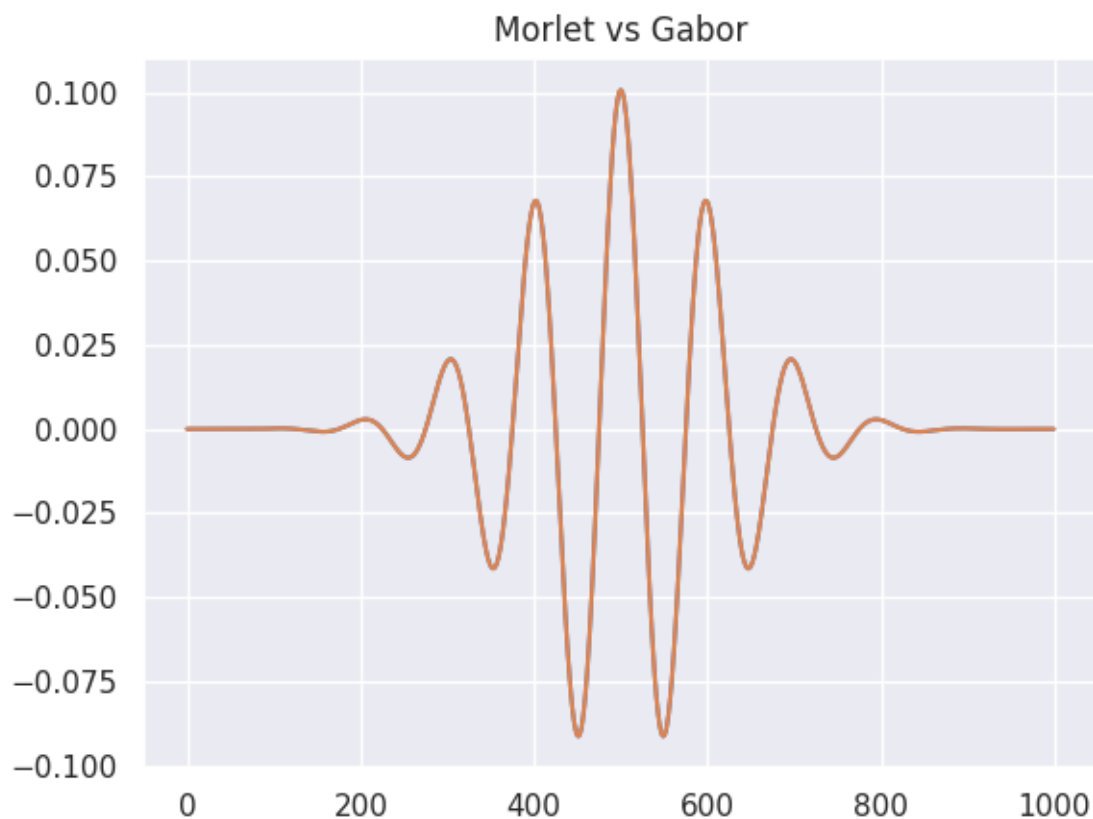
下のプロットを見ても明らかでしょう。

時間解像度を犠牲にすれば全然問題ないね！

実際、僕も  $\sigma = 7$  の MorletWavelet を愛用しており、実用上全く問題ないです。

良かったね！ 解散！

↓  $\sigma$  が 7 の Gabor と Morlet の比較。



重なって線が一本しか見えない！

…でも、根本的な解決になっていませんね。

そもそも  $\sigma$  が小さい場合はどうすれば良いのでしょうか…。

この場合は我慢して MorletWavelet を使うか、そうでなければ別のガウシアンに依存しない Wavelet を用いるところだと思います。

この世には他にも色々な Wavelet があります。

### フーリエ逆変換を使って算出する **Wavelet**

実は、フーリエ逆変換すると算出される Wavelet というのが数種類発見されています。

なんて変態的な導出の仕方なんでしょう！ と思います。

これらの利点は挙げろと言われると難しいのですが、

Morlet と比べて真ん中が凹まないやつが発見されていますから、そこが利点かも？



---

例えば 2000 年に入ってから知られるようになった一般モールス Wavelet[<sup>^</sup>morse] という Wavelet があります。

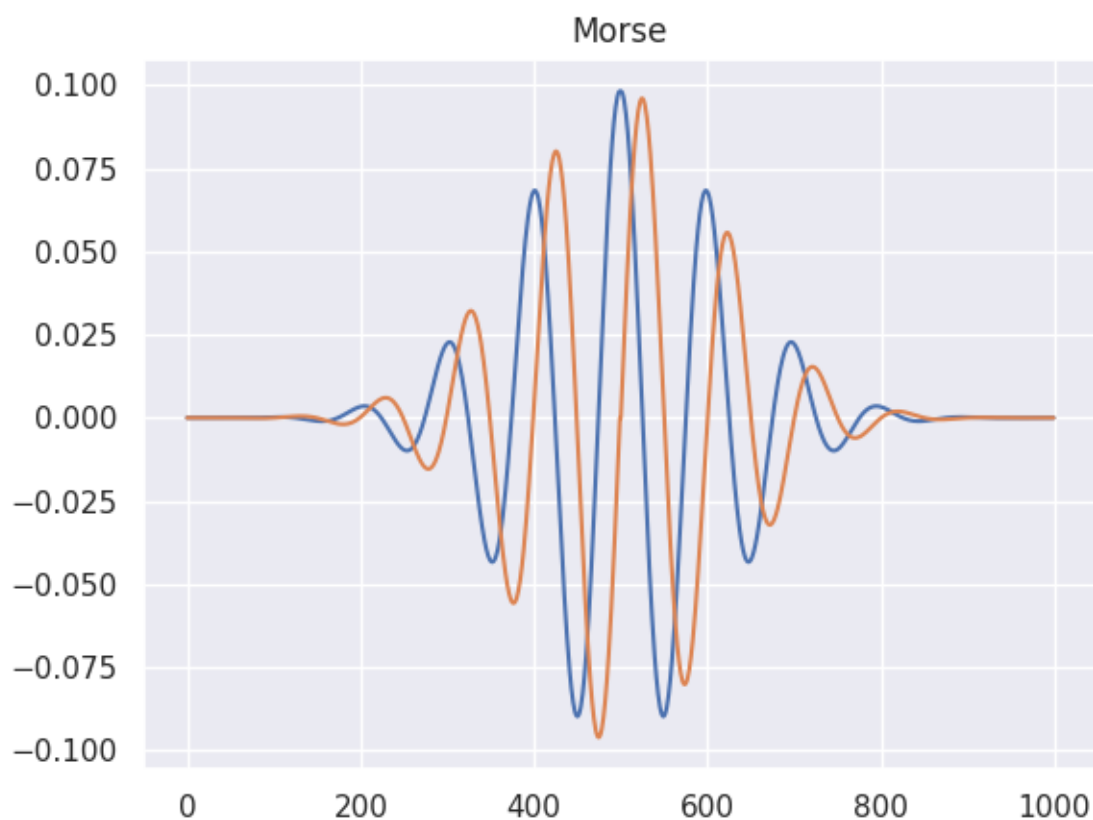
(Morse はモールスっていう読み方で正しいのかどうかは分からないので詳しい人教えて下さい)

既にこれを脳波解析に応用している論文もあります。定義は

$$\hat{\Psi}(w) = \text{sign}(w) \alpha_{\beta\gamma} w^{\beta} e^{-w^{\gamma}}$$

という式...ではなく、これのフーリエ逆変換が一般モールス Wavelet です。

計算上は FFT やるならフーリエ変換の手間が省けてお得ですね！

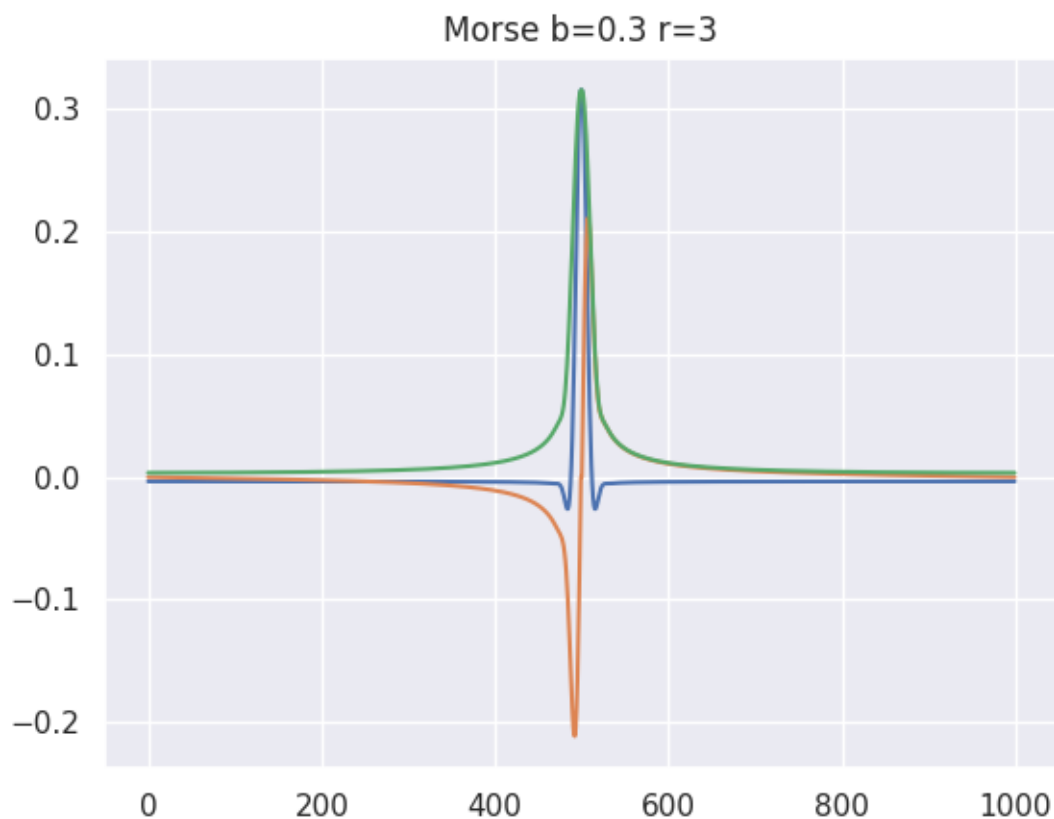


MorletWavelet とそっくりです。(パラメータいじって似せました)

ちなみに、この wavelet は  $\beta$  と  $\gamma$  で調整するんですが、

$\beta$  が低かったらちびまる子ちゃんの永沢君みたいに

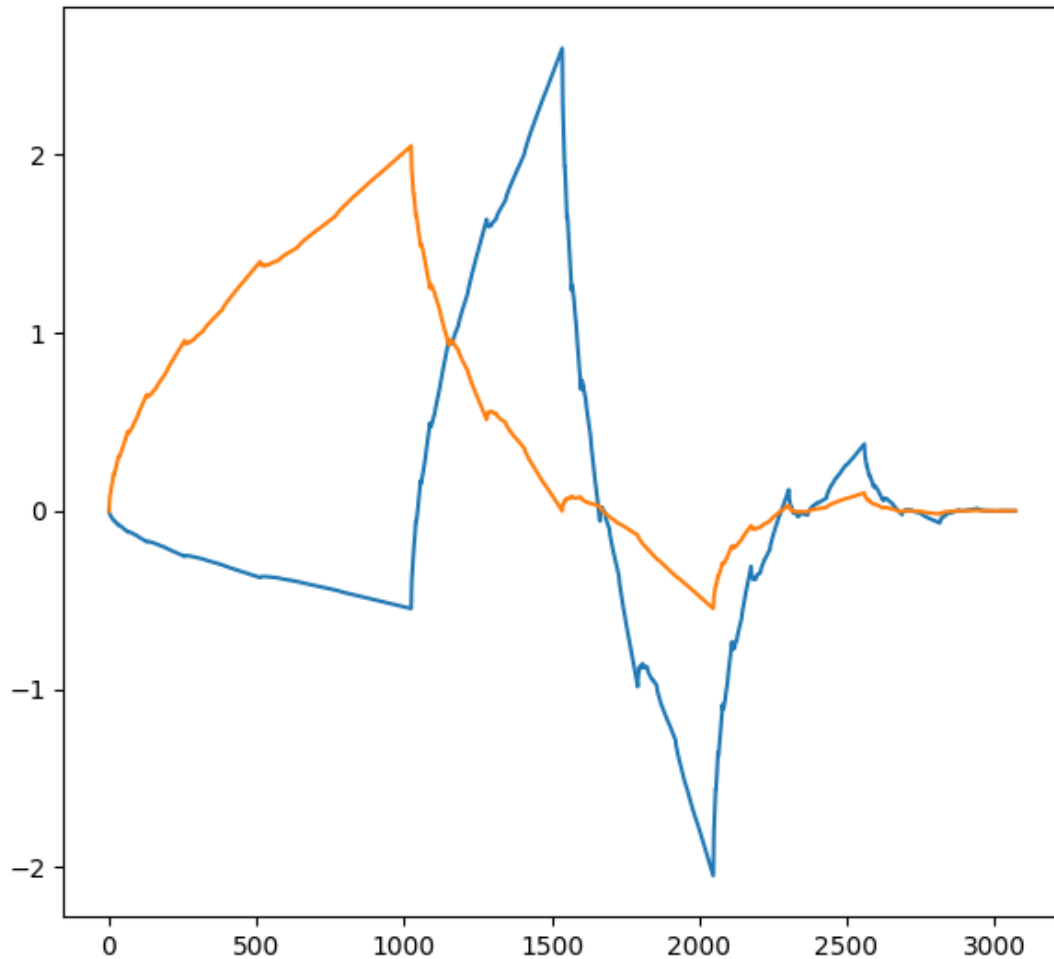
真ん中が尖った波になります。



永沢君みたいな wavelet が欲しかった人には朗報なんですかね？  
(周波数解像度は超絶悪くなります)

## 直交 Wavelet

んで、直交 Wavelet の話がまだだったんですが…マジレス、脳波解析に向きません。  
向きませんが多用されています。  
主に、リアルタイム解析に於いて。  
何故かって？ 多分、計算コストが小さいのと端の問題が深刻じゃないからだと思います…多分。  
一応、ウェーブレット 10 講というガチな本に書いてありますので、  
知りたい人は読んで下さい。下手な解説書よりもこれ読んだ方がいいです。  
名著です…役に立つかは分らんけど。  
直交 Wavelet って、すごいギザギザしていて、普通じゃないです。  
図をご覧ください。



こいつがどういうものかという、上記の Wavelet 変換が  
時間軸に於いて無限であることを辞められなかったのに対して、  
なんと、この Wavelet は時間軸に於いて有限です！

これって凄いことなんです。

---

## ヒルベルト変換

wavelet 変換は 1 つの実数の波から実軸と虚軸を分離しました。

でも、そもそも元々の波は実軸に存在します。おかしいですね？

ということは、もしかして波って本来実軸と虚軸があるのか？ ってなります。

そう、実はそうなのです！

男と女、ウサギとカメ、 $\sin$  と  $\cos$ 、実軸と虚軸の様に、

一つの波があればその相方が存在するのです！

そう、今君は相方の波を探しに行きたくなりました。

元々の実軸をそのまま実軸にして、新たな虚軸を生み出す変換が  
ヒルベルト変換です。導出の仕方は、フーリエ変換と似ています。

ヒルベルト変換の結果は、元の波を  $\cos$  とすると  $\sin$ 。

つまり、まるで Wavelet 変換のように波の振幅を計算できるのです。

凄いですね！

Wavelet 変換は掛け算をして積分をして算出していましたがヒルベルト変換では双曲線を掛け算します。

ここについて数学を理解するためにはコーシー主値だとか

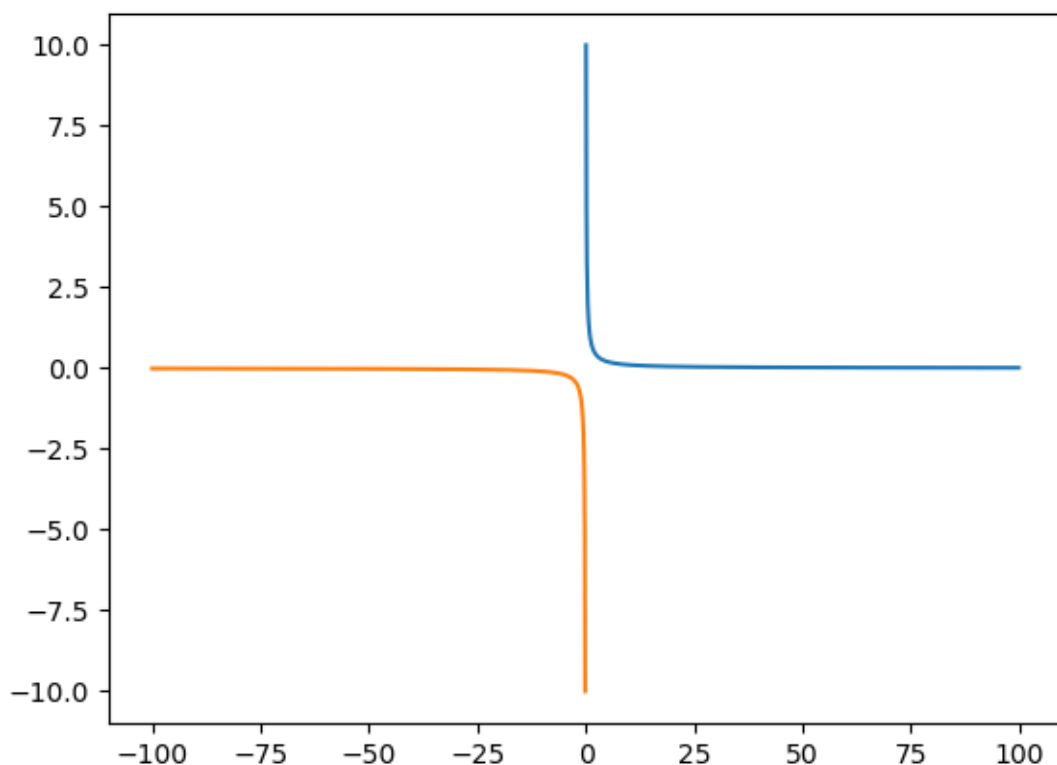
ディラックのデルタ関数を使う必要があります。

デルタ関数とかコーシー主値の解説は出来るけど超絶めんどいのでしません！

というか、この本で読むより別にいい本がいくらでもあります。

それらの出ないところだけ解説します！

一部無限大が出てきて気持ち悪さが残りますが、仕様です。



これについては、既に wavelet 変換と同等の性能を持っていることが分かっている...らしいですが、実際どうなのでしょう？  
なお、このセクションは数学者がブチ切れると思うので、恐恐書いています。

#### 複素フーリエ級数とフーリエ逆変換の復習

波とは、時間軸と振幅の軸がある二次元のデータです。

とりあえず、フーリエ変換の式を書いてみましょう。

$f(t)$  が元の波、 $\hat{f}(w)$  がフーリエ変換結果、 $t$  が時間、 $w$  が周波数として、細かい所を省いてこんな感じです。

$$\hat{f}(w) = \int_{-\infty}^{\infty} f(t)e^{-iwt}dt$$

波というのは時間と振幅の二次元データですね。

---

それを  $-\infty \sim \infty$  の周波数に変換しちゃうのがフーリエ変換でした。  
こんな感じです。

1 振幅 \* 時間 → 周波数 \* 周波数ごとの複素数

そこで、フーリエ逆変換はこんな感じでした。

$$f(t) = \int_{-\infty}^{\infty} \hat{f}(w) e^{iwt} dw$$

1 周波数 \* 周波数ごとの複素数 → 振幅 \* 時間

さて、このフーリエ逆変換は複素フーリエ級数の書き方で書くとこう書けます。

$$f(t) = \sum_{n=-\infty}^{\infty} C_n e^{iwn}$$

これは、単に  $\int$  を  $\Sigma$  に、 $\hat{f}(w)$  を  $C_n$  にそれぞれの書き換えただけです。  
つまり、各周波数  $n$  について、 $-\infty \sim \infty$  までを全部足し算するのがフーリエ逆変換です。  
ちなみに、 $C_n$  は複素数です。

フーリエ逆変換はこの無限の  $C_n$  で出来た複素数の級数から実際の波を求める計算といえます。

フーリエ変換の負の位相の性質

さて、 $\hat{f}(w)$  の  $w$  は  $-\infty \sim \infty$  の周波数 (スカラー値) なのですが、  
ここで複素数  $\hat{f}(w)$  と  $\hat{f}(-w)$  について

$$\hat{f}(w) = \overline{\hat{f}(-w)}$$

と、複素共役になるという事が知られています。  
ぱっと言われて「ん？」となった人はフーリエ変換の導出を復習してみてください。

相方の波を求める

さて、 $\hat{f}(w)$  というのは複素数です。 $a(\cos \theta + i \sin \theta)$  という形をしています。  
つまり、

$$\hat{f}(w) + \overline{\hat{f}(-w)} = a(\cos \theta + i \sin \theta) + a(\cos \theta - i \sin \theta)$$

---

となりますね。だから

$$\hat{f}(w) + \overline{\hat{f}(-w)} = 2a \cos \theta$$

これは実軸の  $\cos$  波です。フーリエ逆変換はこのように共役な複素数を全部を足し合わせていく変換で、正と負の周波数は複素共役の関係であったわけですから、実数が出てくるのは感覚的には普通でしょう。

では、虚軸だとか  $\sin$  波はどうでしょう？ こうすればいいだけです！

$$\hat{f}(w) - \overline{\hat{f}(-w)} = 2ai \sin \theta$$

相方が出てきました！ フーリエ逆変換を足し算とすれば、ヒルベルト変換は引き算なのかもしれない！

式の形にする

全ての正の周波数のフーリエ級数から、負の周波数のフーリエ級数を引き算することで  $\sin$  波が出るのですが、これをきちんと式で表してみましょう。sgn 関数を使います。sgn 関数とは  $\text{sgn}(x)$  の  $x$  が正のときは 1、負のときは -1 を吐き出す関数で、 $\sin$  に似てますが関係ないです。

$$\text{sgn}(w)\hat{f}(w)$$

という感じにすると、正の成分から負の成分を引いていったこととなりますね。

では、こいつを元に、フーリエ逆変換の  $\hat{f}(w)$  を全部引き算にしてやりましょう。

ただ、これだけでは虚数になっちゃうので  $-i$  を掛け算してあげます。

ヒルベルト変換の結果...つまり、欲しかった虚軸の波を  $H(t)$  とすると

$$H(t) = \int_{-\infty}^{\infty} -i \text{sgn}(w) \hat{f}(w) e^{iwt} dw$$

となります。やりました！ ヒルベルト変換です！

---

さらに形を整える

上の式からフーリエ変換の成分を駆逐してやります。

sgn 関数をフーリエ変換すると  $-i\pi/t$  になることが知られています。

(これを示すためにはデルタ関数のフーリエ変換について書くことになり、超絶めんどいので書きません)

では、加工するまえにフーリエ変換の畳み込みの復習をしていきましょう。

### 畳み込み定理

フーリエ変換やラプラス変換において以下のような公式があります。

F をフーリエ変換やラプラス変換とすると以下が成立します。

$$F(a * b) = F(a)F(b)$$

$a*b$  というのは畳み込みを表します。掛け算したうえで積分するというやつですね。

これを使ってヒルベルト変換っぽく作ってみます。ここで、畳み込みの公式を示しておきます。

$$F(a * b) = \int_{-\infty}^{\infty} a(x)b(t-x)dx$$

### よく見るヒルベルト変換の式の導出

ヒルベルト変換をフーリエ変換してみます。

$$F(H(t)) = \text{sgn}(w)\hat{f}(w) = -i\text{sgn}(w)F(f(t))$$

これは畳み込みにすごく近くないですか？ ここで、仮に  $\text{sgn}$  のフーリエ逆変換の結果を  $G$  とします。

$$= F(G)F(f(t)) = F(G * f())$$

で、両辺のフーリエ変換を解除します。

$$H(t) = G * f(t)$$



---

理由を話し出すとちょっと終わらないので今回は言いませんが、実はGは  $1/t$  であることが知られています。

$$H(t) = 1/t * f(t)$$

凄いですね！ これやこの、ヒルベルト変換です！

うん、数学者は見ていないね？…窓に！ 窓に！

---

## コネクティビティ各論

コネクティビティについては色んな method があります。なので、スペクトルとは章を分けてみました。ここではコネクティビティそれぞれの method について僕の考えを述べます。間違っていたらごめんね。ちなみに、以下の方法は基本、低周波じゃないと無理です。  
γ波なんかを処理しようとするのは無謀なのであります。

### PLV と Coherence の問題点

一つの個体の中でコネクティビティを語る場合、  
PLV と Coherence という指標は最早形骸である。  
敢えて言おうカスであると！

さて、フーリエ変換のところでコネクティビティについてちらりと書きました。  
位相の差をとっていけば PLV という「どのくらい波が関連しているか」という  
指標になると書きました。式で表すと

$$PLV = \left| \frac{\overline{S_{xy}}}{|S_{xy}|} \right|$$

という感じです。この指標は正しいです。が、大きな欠点があります。  
脳波にしる、脳磁図にしる、脳内に電極をブチ込むやり方ではなく、  
漏れて拡散してきた物を捉えることになります。  
ということは、何らかの大きな震源が近くにある場合、  
影響を受けて似たような波が出た全てのチャンネルは  
コネクティビティが「ある」と間違った結果が出てきてしまうのです。  
図に沿って言うと、2つの青い点(センサー)で、一つの赤い波を同時に測定すると、  
繋がっていると勘違いするのです。常識的におかしい。

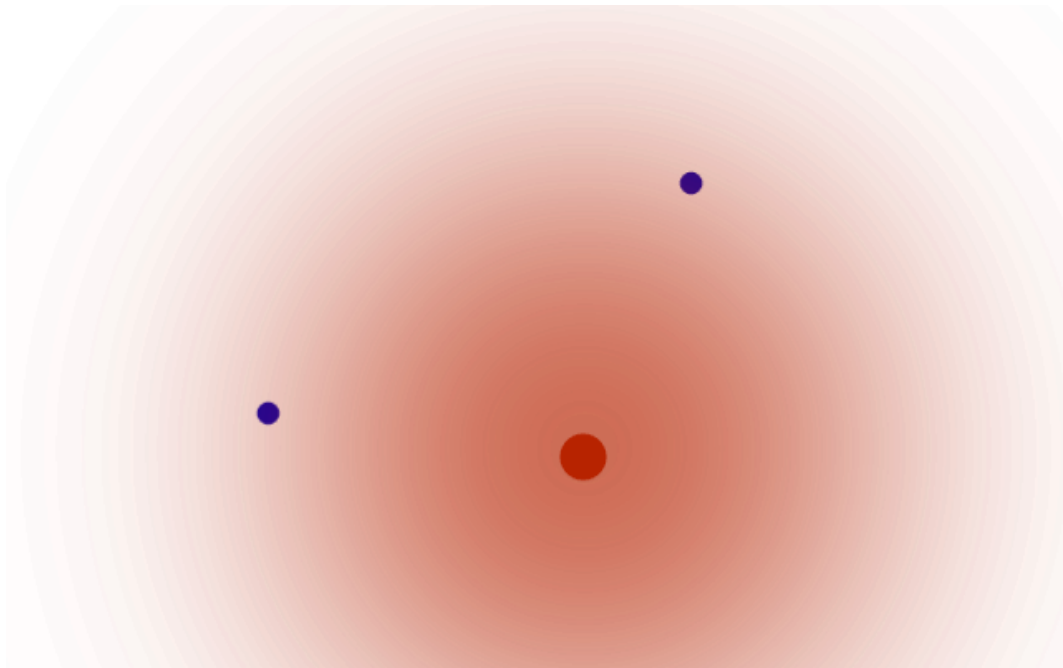


Figure 23: PLV でコネクティビティを計算する場合、この図の青い点が繋がっているということになる。

さて、PLV について語ってきましたが、Coherence という計算の方法もあります。以下のように計算します。これもまた、違和感のない計算方法ですが、PLV と同様に拡散やノイズの影響が大きいです。

$$Coherence = \frac{\overline{|S_{xy}|}}{\sqrt{\overline{|S_{xx}|} * \overline{|S_{yy}|}}}$$

ちなみに、Coherence に位相情報を残したものが Coherency です。

$$Coherence = \frac{\overline{S_{xy}}}{\sqrt{\overline{|S_{xx}|} * \overline{|S_{yy}|}}}$$

こいつの使いみちは僕はあまり知らないです...

## PLV や Coherence の欠点の克服

いくつか方法があります。大きく分けて 2 つ、またはその組み合わせです。

- 
- 拡散する前の電流を推定する
  - 拡散しても大丈夫な計算方法を採用

### 拡散する前の電流

CurrentSourceDensity という方法があるらしいです。

こっちはソースベース解析ほどバリバリに推定するわけではない感じが、PLV の弱点を補う方法ですね。残念ながら MNEpython には未実装です。ていうか、僕はこの方法を知らないのです。

### PLV の発展系

mne-connectivity に実装されている有名所として、PLI と WPLI を紹介します。

**PLI** PhaseLagIndex という指標があります。こいつは上記の拡散をキャンセルしてしまう方法で、MNE では spectral\_connectivity 関数に実装されています。式は下記

$$PLI = \overline{|sign(Im(Sxy))|}$$

ここで、sign は内容が正なら 1、0 なら 0、負なら -1 を返す関数。

Im は虚軸だけを返す関数です。

位相が常に x より進んでいたり、遅れている奴だけ加算していき、位相のズレが 0 付近をウロウロしてるやつやバラバラなやつは消すメソッドですね。確かにこれならばさっき挙げた偽のコネクティビティは出ないでしょう。

とはいえ、結構荒っぽいやり方なので、僕は初見「マジカヨ…」って思いました。これで Connectivity を Plot したら、隣同士ばかり繋がるような図じゃなくなります。つまり、成功しているということですかね。

**WPLI** 上記で、正なら 1、負なら -1 というのについてマジカヨ感が漂うのですが、そこをもう少しスムーズにしたのが WeightedPhaseLagIndex です。これは位相のズレが  $\pi/2$  に近ければ近いほど大きな値を、 $-\pi/2$  に近ければ近いほど小さな値を入れ込みます。そうすることによって、ノイズに強くなった…らしいです。そりゃそうですね。PLI は一寸ノイズがあったら 1 か -1 に振り切れますもの。式は以下。

---


$$WPLI = \frac{|\overline{S_{xy}}|}{|\overline{S_{xy}}|}$$

見にくい記法ですみません…

まあ、それでも「マジカヨ…」感はありますけどね。

## Coherence の発展系

さて、PLI 系は同一の電流源からのラグをとっていたわけですが、同様のやり方が ImaginaryCoherence です。

これは Coherence 虚軸のみを加算平均したものです。

虚軸を加算平均するということは、位相が  $\pi/2$  ずれたら最大になりますね。

式は下記のような感じです。

$$Coherence = \frac{\overline{Im(S_{xy})}}{\sqrt{|\overline{S_{xx}}| * |\overline{S_{yy}}|}}$$

で、こういうのってロバストなの？

分からない…僕には何もわからないんです…実際にコネクティビティを計算してみればいいです。各方法で計算した所、全く違う結果になることはよくあるんです。悲しくなりますね。

## グラフ理論

さらなる解析として、グラフ理論があります。

これは「互いにどんな風に繋がっているかな？」というのを考えていく理論です。色々なやり方があります。

例えば、一筆書きでどんな風に繋ぐか、ループを作らずにどんな風に繋ぐかとか、そういうノリのやつです。後述します。

---

## ソースベース解析の理屈

かなり面倒いので丁寧にはかけません。ここからは線形代数の範囲になります。  
分かる人に言うと、疑似逆行列を正規化した方法が MNE です。

以下、二行目で「は？」ってなったそこの君、PRML を読んで下さい。

PRML を読めないそこの君。君のために解説を書きます。

誤解を恐れずに脳内の活動を推定する方法を超簡単に言いますと、  
ソース推定とは鶴亀算であり、割り算です。

「鶴亀算と割り算は別のもんじゃないwww」と思った人は頑張らねばなりません。  
なので、そういう人のために、割り算については後述します。

さて、MNE はそんなふうに割り算をしてあげた上で、答えをもうちょっと  
スムーズにしてあげる為に正規化という処理を施して上げるものです。

つまり、MNE とはスムージング入りの割り算である！

と初心者には伝えたいです (怒られそう...)

数学得意な人のために一言で言うと下記の表のとおりです。

Method	内容
MNE	脳全体のノルムに関する重み付き最小ノルム解
dSPM/sLORETA	分散で補正した MNE
eLORETA	事前確率分布を事後確率分布で補正する MNE
beamformer 法	波のノルムに関する重み付き最小ノルム解

## もう一寸ちゃんと

まず、脳のソースから出た磁力...電力でも良いですが、そのようなものがセンサーに届く時、  
その強さはソースで発生した電力、磁力に比例するはずです。

詳細はアレですね...マクスウェルの方程式ですね。多分、距離の二乗には反比例すると思います。

つまり...距離云々を除くと簡単な一次連立方程式になるはずなんです。

センサーで捉えた情報を  $y$  とし、ソースで発生したのを  $x$  としましょう。

あるソースで発生した電力とセンサーで捉えた電力の関係を下記の式で表せるとします。

$$y = ax$$

---

これは単純な掛け算なわけですが、1 センサー 1 ソースではなく、多センサー多ソースです。  
ここで、沢山になった  $y, a, x$  について、下記のように表すします。

$$Y = y_1, y_2, y_3 \dots$$

$$X = x_1, x_2, x_3 \dots$$

ここで  $A$  を下記を満たす行列とします。

$$Y = AX$$

この連立方程式を解きます。

横と縦を掛け算するので、行列の掛け算は連立方程式になるのは自明です。

これがどうしても連立方程式に見えない人は、

行列代数の入門書でも読んで下さい。ネットで検索するよりこれは本の方がいいです。

では解きま...じつは解けません。一次連立方程式はあんまり数が多くなると解けなくなるのです。  
鶴亀カブトムシ算は無限の解があるのです。

で、最も真実に近いっぽいのを推定していきます。

今回、脳全体のある一瞬の波が最も小さくなるような波を推定しましょう。

何故最小にしたいかって？ 乱れた波というものはグニョングニョンしてるので

直感的に言ってグニョングニョンしてる波よりも静かな波のほうが

本物っぽいでしょう？ と位にしか言えません！

その本物っぽい割り算を  $A^\dagger$  として

$$X \Rightarrow A^\dagger Y$$

という感じにするのを目指します。

この方法が `MinimumNormEstimation(MNE)` です。

(他に一つ一つの波が一番小さくなる beamformer 法とか色々あります)

---

条件  $Y = AX$  のもとで  $X$  を小さくしたい<sup>67</sup>のです。  
 $X$  が小さいってことは下記の式が小さいってことです。

$$\|X\|^2$$

ここで、 $\|X\|$  とは  $X$  の対角成分の事を指します。  
例えば  $X$  の縦をチャンネル、横を時間軸とします。  
 $\|X\|^2$  がどうなるかと言うと、 $\|X^T X\|$  となります。  
何故これの対角行列かと言うと、  
 $X$  の 1 行目...例えばチャンネル 1 としましょう。これと  $X$  の 2 列目...これはチャンネル 2 です。  
これらの掛け算はチャンネル間の相関を表しているのであって、振幅を表しません。  
つまり、対角行列のみが大きさを表すのです。  
まず、ラグランジュの未定乗数法という公式を使います。知らないカスはググれ。  
僕はこんな同人誌を書く程度には親切で友達がいっぱい出来そうな人<sup>68</sup>なので教えてあげます。  
最大値や最小値を手軽に導くテクニックです。  
最大値や最小値は極値になるので、微分して 0 になればいいのです。  
 $f(x, y)$  が極値になる  $x, y$  は

$$L(x, y, \lambda) = f(x, y) - \lambda g(x, y)$$

の時に

$$\frac{\partial L}{\partial \lambda} = \frac{\partial L}{\partial y} = \frac{\partial L}{\partial x} = 0$$

の解、または

$$\frac{\partial g}{\partial x} = \frac{\partial g}{\partial y}$$

の解。...という感じの公式です。  
行列の微分をしないとイケないので、行列の微分の仕方を確認しておきます。

---

<sup>67</sup> ノルムが小さいということ。中学生風に言うと絶対値。

<sup>68</sup> リアル友達欲しいです (´; ω; `)



---

$$\frac{\partial a^t x}{\partial x} = a$$

$$\frac{\partial x^t a}{\partial x} = a$$

今回はこれを微分します。

$$L = \frac{X^2}{2} - \lambda^T (AX - Y)$$

$$\frac{\partial L}{\partial X} = X - A^T \lambda$$

これが0になるので

$$X = A^T \lambda$$

$$Y = AX = A A^T \lambda$$

$$X = A^T (A A^T)^{-1} Y$$

これで無事  $X$  を  $A$  と  $Y$  で表せました。  
 まあ、ここまでは普通の最小二乗の最適解なのだけど。

もっと良くする

もっと良くします。

$$Y = AX$$

という感じの割り算をときます。  
 が、今回はL2 ノルムを使います。L2 ノルムについてはググって下さい。

---

L2 ノルムは要するに二乗をちっさくしようということです。

$$\operatorname{argmin} L_x = \|Y - AX\|^2 + \lambda \|WX\|^2$$

ここで、 $WX$  とは何かという疑問が生じます。

$X$  はまあ  $X$  なのですが、 $WX$  は「これがちっさくなって欲しいな？」というものです。

$W$  が  $I$  でも計算は出来るのですが、 $W$  が  $I$  の場合どうなるかと言うと

「なるべく全てが、平等に正則化されるように」です。

$W$  は重みとでも言えるもので、 $X$  の振幅の逆数とでも言えるようなものです。

つまり、 $X$  が小さくなってほしいと言うより、

$X$  の大きい部分は大きく、小さい部分は小さく予測したいのです。

$W^T W$  の逆数は分散になるような感じです。

詳しくは最小二乗法でググれ。誰かが「重み付き最小二乗法」とかで書いてる。

ここで、 $\|X\|$  は  $X$  のトレースなのですが、行列の定義をみて計算してみてください。

これはきちんと「 $X$  の最小値」を求める式になっています。

$$\hat{L} = \operatorname{tr}((Y^T - X^T A^T)(Y - AX)) + \lambda \operatorname{tr}(X^T W^T W X)$$

$$= \operatorname{tr}(Y^T Y - X^T A^T Y - Y^T A X + X^T A^T A X) + \lambda \operatorname{tr}(X^T W^T W X)$$

ここで、微分します。

$$\frac{\partial \hat{L}}{\partial X} = -2A^T Y + 2A^T A X + 2\lambda W^T W X = 0$$

$$-A^T Y + A^T A X + \lambda W^T W X = 0$$

$$X = (A^T A + \lambda W^T W)^{-1} A^T Y$$

さて、 $W^T W$  は分散の逆数なのですが、

$(W^T W)^{-1}$  が長いので  $\Sigma$  って書き換えさせて下さい。

---

$$X = (A^T A + \lambda \Sigma^{-1})^{-1} A^T Y$$

出ました！ 万歳！ MNE の式です！ だけど、なんだか書いてあるのと違う？  
お前は教科書に載っている通りじゃないと満足しない心の狭い人間なのだね？  
OK、書き直すわ…

$$(A^T A + \lambda \Sigma) A^T = \Sigma A^T (A \Sigma A^T + \lambda I)^{-1}$$

上の式をじっと見て下さい。成立することが解ると思います。  
故に、こうなります。

$$X = \Sigma A^T (A \Sigma A^T + \lambda I)^{-1} Y$$

これやこの、MNE の式です。  
だけど、まだ  $\lambda I$  が怪しいですね？ ここで、はじめの式を見て下さい。

$$\operatorname{argmin} L_x = \|Y - AX\|^2 + \lambda \|WX\|^2$$

これ、あからさまに第一項が Y にまつわるもの、第二項が X にまつわるものですね？  
ということは、X に W があるなら Y にもそれに相当するものがあるって良いはずです。  
だから、脳内のノイズの分散共分散行列を C としてあげます。  
…ノイズと言いましたが「背景」と言ったほうがわかりやすいかも知れません。  
数学得意な人は「重み付き最小ノルム解」と言います。  
数学得意な人は一々強そうな名前言いますよね…。式はこんな感じ。

$$\operatorname{argmin} L_x = C \|Y - AX\|^2 + \lambda \|WX\|^2$$

実際計算するとこうなるようです。

$$X = \Sigma A^T (A \Sigma A^T + C)^{-1} Y$$

で、この値は正則化されていないわけですから、

---

グニョングニョンの値になってしまいます。  
なので、結局正則化のパラメータ入れるのです…。  
(ここは流石にググって)

$$X = \Sigma A^T (A \Sigma A^T + \lambda^2 C)^{-1} Y$$

読者の皆さんはつかれましたか？ 僕も疲れました。  
まだ続きます (´・ω・`)

白色化

脳内の各皮質って、それぞれ独立でいて欲しいですよね？  
じゃないと、脳内に沢山点々を打った意味がなくなります。  
こういう風に、それぞれの相関を打ち消す操作を白色化と言います。  
まず、想定されるノイズに関して、分散共分散行列を作ります。  
ぐぐ r…これは、分散を集めた的な行列です。  
この対角成分は波そのものの分散なはずです。  
毎回観測される波は同じようなものであるという想定のもとで推定をしたいので、  
この分散共分散行列を式に入れ込みます。

$$Y = AC^{1/2}C^{-1/2}X$$

$$\hat{A}^\dagger Y = C^{-1/2}X$$

$$\hat{A} = A^\dagger C^{1/2}$$

の元で計算できる  $A^\dagger$  を  $\hat{A}^\dagger$  とすると

$$\hat{A}^\dagger C^{1/2}Y = X$$

---

こんなかんじ。

$$\begin{aligned}\hat{A}^\dagger C^{1/2} \\&= \Sigma A^T C^{1/2} (A C^{1/2} \Sigma C^{1/2} A^T + C)^{-1} C^{1/2} \\&= \Sigma A^T (A \Sigma A^T + \lambda^2 I)^{-1} Y\end{aligned}$$

せっかくここまでcを入れてたのに脱力感半端ないっすね。  
まあ、実際の式ではcを掛け算したりはしますが、逆行列からは外れます。

## MAP 推定

今回はラグランジュの未定乗数法で二乗したやり方を書きましたが、  
ここまででも書きましたが、ベイズ統計学の視点からの見方もあります。  
しかし、ここまで書いて疲れました。これはここに書くのが超絶面倒いので書きません。  
この解説だけで本が一冊書けます。ぶるむるでも読めば良いんじゃないかな？

## dSPM の理屈

さて…MNE の式をよく眺めてみましょう。これは

$$Y = AX$$

の変形であり、シンプルな掛け算なのは言うまでもありません。  
そこで、MNE の式を次のように書き直してみます。

$$X = A^\dagger Y$$

ここで、ふとした疑問が出てきます。  
 $A^\dagger$  の分散が1 じゃない場合に奇妙なことが起こります。  
 $A^\dagger$  が1 じゃない場合で、空室を撮ったと仮定してみてください。

---

センサーが捉えるノイズと空室の分散は同じ大きさのはずですが、 $A^\dagger$  が1じゃなかったら空室の中のノイズが大きくなったり小さくなったりしておかしいですね???  
脳が存在しない場合でも、脳があるはずの所に何かが生じてしまう。

ということで、計算結果を補正してあげる必要があります。  
通常、補正の方法は分散の  $1/2$  乗で割り算してあげる事によって為します。  
分散って言うと、振幅の二乗ですから、つまり振幅で割ってる的なことです。  
それぞれのソースを、そのソースの振幅で割り算してあげるのです。

振幅で割るってどうするのかって?  
対角成分で割り算してあげるのです。

答えを分散で割るとき、その方法が有名所が2つあります。  
一つの方法は下記です。

$$Y = AX$$

$$A^\dagger Y = X$$

この  $A^\dagger$  を補正してあげるのですが、ソースの振幅を出さねばなりません。  
センサーの振幅は分散の  $1/2$  乗なので  $C^{1/2}$  とします。  
そこで、ソースの対角成分を考えます。Yの誤差を  $y$  としておきましょう。  
 $C$  はYの分散、即ち  $y$  の二乗に比例します。  
そして、縦横ともにYのチャンネル数に等しいです。  
 $\|A^\dagger y\|$  の大きさ的な物で割り算したいんですよね?  
なら、チャンネル数じゃなくて  $y$  のソース数分の大きさの行列にするべきです。  
 $A^\dagger$  は縦がチャンネル数、横がソース数の行列になるはずですが。  
その、縦横を考慮して、ソースを割り算できる行列を作るなら、  
 $\|A^\dagger y (y A^\dagger)^T\|$  という感じの形になるはずですが。  
つまり

$$\sqrt{\|A^\dagger C (A^\dagger)^T\|^2}$$

となります。これで割るわけですが、だから、こうなります。

---


$$X = \|A^\dagger C(A^\dagger)^T\|^{-1/2} A^\dagger Y$$

これぞ、dSPM です！

### sLORETA の理屈

ソースの分散を別の方法で計算してみます。

$$\sqrt{\|X^T X\|} \Rightarrow \sqrt{\|Y^T A^\dagger X\|} = \sqrt{\|Y^T A^\dagger A Y\|}$$

つまり、推定された世界では  $X$  の分散は  $\|A^\dagger A\|$  倍になっているということです。  
じゃ、それで割ればいいということになりますね。

$$X = \|A^\dagger A\|^{-1/2} A^\dagger Y$$

はい、sLORETA です。

### eLORETA の理屈

もうほんとに疲れてきましたが、ここが最後かな…。

sLORETA は  $A^\dagger A$  で割り算していました。

だけど、本当に  $A^\dagger A$  でいいん？ という問題があります。

実はこれ、もっとよく出来るよ！ というのが eLORETA です。

やることは、上記を再帰的にしたものです。

発想としては、この式の  $\Sigma$  ってどうよ？ ってことです。

sLORETA によると  $\|A^\dagger A\|$  がソースの分散でした。まずは

$$X = \Sigma A^T (A \Sigma A^T + \lambda^2 I)^{-1} Y$$

この式の内容から考えていきます。

sLORETA で想定される分散はこうなります。

---


$$\Sigma A^T(A\Sigma A^T + \lambda^2 I)^{-1} A$$

$\Sigma$  はソースの分散ですね。

で、 $\|A^\dagger A\|$  もソースの分散みたいなものです。

つまり、下記の等式が成立して欲しい。

$$A^T(A\Sigma A^T + \lambda^2 C)^{-1} A = \Sigma^2$$

$$\sqrt{A^T(A\Sigma A^T + \lambda^2 C)^{-1} A} = \Sigma$$

この  $\Sigma$  を収束させるために、繰り返し上記の式を再帰的に計算してきます。

1. まず、 $\Sigma$  を計算します。一寸ズレてます。
2. 次に、 $\sqrt{A^T(A\Sigma A^T + \lambda^2 C)^{-1} A}$  を計算します。
3. 2 を新しい  $\Sigma$  として、以下無限ループ。

はい、eLORETA です。一応、eLORETA って探索的に場所を特定していくのに向いているらしいです。<sup>69</sup>

## eLORETA の実行

さて、MNEpython で行う時は、いくらかパラメータを入れる必要があります。

一つは  $C$  を決める時のパラメータ、 $\epsilon$  です。

MNE は  $I$  に分散を掛け算することで  $C$  を求めるらしいのですが、

$\epsilon$  も掛け算します。これにより、どの程度正則化するかが決まるようです。

正則化項のコントロール目的のパラメータですね！

それから、`max_iter` というのがあります。

これは、上記の再帰を何回再帰するかです。

そして、`force_equal` というのがあります。

これは、脳内の電流は三次元ベクトルですから、それをどう扱うかのパラメータです。

これは default で `None` ですが、`None` にしていると MNE がよしなにしてくれます。

`loose` というパラメータがソース推定であります。

これは脳の皮質の電流の方向のズレをどの程度許容するかのパラメータです。

---

<sup>69</sup>Tamesh Halder, Siddharth Talwar, Amit Kumar Jaiswal and Arpan Banerjee eNeuro 16 July 2019, 6 (4) ENEURO.0170-19.2019



---

という感じの理解で多分合ってると思うけど、間違ってたらごめん (´・ω・`)

---

## 割り算を極める☒

こんにちは。このセクションは一つ前のセクションで、  
言っていることが何も分からなかった友達に特別支援をするための所です。  
君たちが分からなかったのは何かな？  
君たちが分からなかったのは割り算だ！ というわけで…

今回は 多変量解析 一般線形モデル 古典的機械学習  
割り算の話をしてします。

早速ですが、割り算とは、例えば下記の式です。

$$\frac{10}{2} = 5$$

逆元 ( $1/X$ ) を掛けることと等価です。

$$10 \times \frac{1}{2} = 5$$

何の変哲もない割り算ですが、これを極めると、  
様々な応用が効くようになります。(多変量解析・逆問題)  
例えば、鶴亀算は紛れもなく割り算なのであります。

鶴亀算とは連立一次方程式である

鶴亀算とはこの様な問題です 🐦🐢

鶴と亀の脚の本数の合計は 10、個体数を 4 とします。  
鶴と亀はそれぞれ何匹ずつ居るのでしょうか？

この問題は、小学校中学年がよく悩む問題です。<sup>70</sup>

そして、実は簡単な連立方程式で表わせます。鶴を  $x$  匹、亀を  $y$  匹とすると以下のとおりです。<sup>71</sup>

$$2x + 4y = 10$$

$$x + y = 4$$

上が足の数、下が髀の数です。

---

<sup>70</sup>ただし、世代による

<sup>71</sup>ここまでが中学数学の範囲

---

連立一次方程式とは行列の割り算である☒

上記の2つの式ですが、この様に変形できます。

(ここで、仮に足の本数を本、体の数を体の単位で表してみます)

$$\begin{pmatrix} 2\text{本} & 4\text{本} \\ 1\text{体} & 1\text{体} \end{pmatrix} \begin{pmatrix} x\text{匹} \\ y\text{匹} \end{pmatrix} = \begin{pmatrix} 10\text{本} \\ 4\text{体} \end{pmatrix}$$

横から縦に掛けるからそうなるのは実際に計算すれば分かると思います。

鶴亀算の高級な解き方

小学生には小学生なりの解き方がありますが、応用を利かすために大学生らしい解き方をします。

鶴亀算の式が上記のように行列で表現できるならば、逆行列を使うことで解を得られます。

ここで、上記の式を以下のように表してみます。

$$A = \begin{pmatrix} 2\text{本} & 4\text{本} \\ 1\text{体} & 1\text{体} \end{pmatrix}$$

$$X = \begin{pmatrix} x\text{匹} \\ y\text{匹} \end{pmatrix}$$

$$Y = \begin{pmatrix} 10\text{本} \\ 4\text{体} \end{pmatrix}$$

$$AX = Y$$

逆行列を用いて

$$X = A^{-1}Y$$

これでxを計算できます。なんと、鶴亀算とは行列の割り算のことでした 🐢

ちなみに、これは鶴てんとう虫算とか亀てんとう虫算とかも成立するのですが、

---

蛇鶴算とか亀猫算は成立しません 🐍 0本足や足の数が同じお友達が入ると難しいですね 🐢  
このやり方なら、方程式をコンピュータに自動的に解かせる事が出来ます。

## 実装 (python )

では逆行列を作りましょう!今回はプログラミング言語 python🐍 の numpy パッケージを使います。  
まずは、python で割り算のパッケージに、上の行列を入れ込みます。(足と体の数は適当)

```
1 from numpy import linalg, array
2 A = [[2, 4], [1, 1]] # それぞれ足と首の数
3 At = linalg.inv(A) # 逆行列
4 Y = array([10, 3]) # 足が10本、首が3本
```

逆行列 ( $A^{-1}$ ) が出来たので、あとは「逆元を掛ける」と解が出ます。

```
1 At @ Y # 逆行列の掛け算
2 >>> array([1., 2.])
```

答えは鶴1匹と亀2匹ですね。(ここまでが高校数学の範囲)

## 鶴亀てんとう虫算

鶴亀算が出来たので、次は鶴亀てんとう虫算をやってみましょう。  
項目を増やしていけば「鶴亀てんとう虫蜘蛛ムカデ」算が出来るようになり、  
非常に応用の幅が広がりそうです。(大学数学の範囲) 🐞🐜🕷️🐍

さて、鶴亀てんとう虫算は下記のような形をしているはずです。  
(もういいかなと思うので単位は省略します)

$$\begin{pmatrix} 2 & 4 & 6 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix}$$

## 無限個の解とその解決策 $\infty$

実は、鶴亀てんとう虫算は小学校中学校の正攻法では解くことが出来ません。  
亀が増えても減っても鶴とてんとう虫で調整できてしまうから、答えが一つに決まらないのです。<sup>72</sup>

---

<sup>72</sup>無限個の解が存在します。

---

では、どういう条件なら解があるのでしょうか？

鶴、亀、てんとう虫と、三種類の動物の数を調べるなら、  
足と首の数だけじゃ決まりません。3つの解には3つの式が必要なのです。

鶴亀てんとう虫算にはもう一つの値、羽の数が必要だったのですね ♪  
では、それでやってみましょう！<sup>73</sup>

$$\begin{pmatrix} 2 & 4 & 6 \\ 1 & 1 & 1 \\ 2 & 0 & 4 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

```
1 from numpy import linalg, array
2 A = array([[2, 4, 6], [1, 1, 1], [2, 0, 4]])
3 At = linalg.inv(A)
4 Y = array([10, 4, 6])
5 At @ Y
6
7 >>> array([ 3.,  1.,  0.] )
```

計算できました。

逆行列は作れないこともある

適当にさっきの行列を弄ってみましょう。

```
1 from numpy import linalg, array
2 A = array([[2, 4, 6], [1, 1, 1], [0, 2, 4]])
3 At = linalg.inv(A)
4 >>> LinAlgError: Singular matrix
```

エラーが出ました！ 🐛 この行列では逆行列は作れないのです！ こういうこともあります。  
うまくいくかどうかの条件はありますが、今回は難しい数学はしません。<sup>74</sup>

---

<sup>73</sup>ちなみに、羽が二枚の昆虫はハエとアブと蚊だけです。可愛い虫には羽がたいてい4枚あります。

<sup>74</sup>実は、この条件を求めるのは難しいだけでなく僕にとって退屈な作業です。ちなみに、高校数学の範囲内です。

---

## 疑似逆行列

足の数と首の数しか無いときは、答えがないので推定するしかありません。

よく使われる方法は  $X$  の L2 ノルム<sup>75</sup> が最も小さくなる解を仮の解とする方法です。

無理やり上記の  $AX = Y$  を  $X = A^\dagger Y$  の形にしてしまいます。

これは疑似逆行列と言います。<sup>76</sup> 便利ですね！ では実装してみます。

```
1 from numpy import linalg, array
2 A = [[2, 4, 6], [1, 1, 1]] # 足と首の数
3 At = linalg.pinv(A) # 疑似逆行列
4 Y = array([10, 3]) # 足10本と首3本
5 At @ Y # 疑似逆行列の掛け算
6
7 >>> array([1.5, 1. , 0.5])
```

適当に数を決めたら少数が出てきました ☹

この割り算は、実は色々なパターンがあるのですが、  
代表的なパターンとしては、例えば下記があります。

$$X = (A^T A)^{-1} A^T Y$$

## ここまでで学んだ割り算の応用☒

- 花の色や大きさから花の種類を推定する (アヤメ分類問題)
- fMRI や脳波・脳磁計による脳内活動の推定 (SPM/MNE)
- 多変量解析を用いた研究
- 以下、無限の応用...

カーネル法 (極めて強烈な掛け算。本が一冊かける。) を応用すれば、  
なんと無限次連立方程式すら解くことが可能！<sup>77</sup>

---

<sup>75</sup> 数学で言う、大きさの種類の一つ。全ての要素の二乗の合計のルートを取ったもの。

<sup>76</sup> 別名、ムーアペンローズの逆行列

<sup>77</sup> カーネル法を用いたこの方法はサポートベクトルマシンやカーネル PCA と言われる分類器に応用された。これは一世代前の最も成功した人工知能である。

---

## 具体的な応用多変量解析入門

では、多変量解析に入門してみましょう！しかし、最早入門も終わったも同然です。

ある集団の平均寿命を  $Y(89, 67, 54, 94, 75\cdots)$  とします。

一人ひとりの飲酒量  $A1(1, 2, 4, 0, 0\cdots)$

一人ひとりの喫煙量  $A2(0, 10, 20, 0, 10\cdots)$

一人ひとりの1日の歩数  $A3(4000, 7000, 1500, 2000, 3000\cdots)$

として集め、 $(A1, A2, A3)$  とし、一寸変形します。(転置)

そして、それぞれの生命予後への寄与率を  $X(a, b, c)$  とし、式にします。<sup>78</sup>

最後に、それぞれの計算でどうしても出てくる誤差を  $E$  とします。

$$\begin{pmatrix} 1 & 0 & 4000 \\ 2 & 10 & 7000 \\ 4 & 20 & 1500 \\ 0 & 0 & 2000 \\ 0 & 10 & 3000 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 89 \\ 67 \\ 54 \\ 94 \\ 75 \end{pmatrix} + E$$

$$AX = Y + E$$

ここで、計算の誤差を最小にする  $X$  を考えます。

具体的には  $E$  の絶対値が一番小さい時が都合いいです。

$$|AX - Y| = |E|$$

つまり  $E$  が  $0$  に近い時<sup>79</sup>こそが  $X$  の最適解なので、下記に近似させればいいです。

$$AX - Y = 0$$

なんと！ あからさまに割り算です。なので、コンピュータに解かせられます。

実際は微妙に違うのですが、大まかな理解の一步としては遠くないはずです。<sup>80</sup>

---

<sup>78</sup>ここでは数値はあくまで例なので適当です)

<sup>79</sup>完全に  $0$  にはならないけれど

<sup>80</sup>これには、もうちょっと考えるべきことが入りますが、解や式自体は全く同じになります。

---

## まとめ

- 鶴亀算 (連立方程式の一種) とは他ならぬ割り算であった
- 割り算はコンピュータで解くことが出来る
- 割り算を応用することで様々なことが分かる (医学への応用も多数)

## さいごに

君がしている割り算...それを圧倒的に高めたものが eLORETA。



---

## 補講

ここではちょっとした小技、オススメ書籍などを紹介します。

### python での高速化のあれこれ

MNEpython を使う場合、並列化以外の高速化はあまり考えないでいいです。  
理由は、numpy を使っているので十分速いと思われるからです。  
しかし、独自のメソッドを実装する時なんかには、処理速度が大事になる事もあります。  
その時のやり方をいくつか記しておきます。

#### for 文とリスト内包表記と map

python の for 文は絶望的に遅いため、for 文の入れ子はやめましょう...とされています。  
MNE 使うときには大差ないし良いんじゃないかと個人的には思いますが。  
しかし、この所は今後の高速化を学ぶための布石になります。  
for の代わりと言ってはアレですが、いくつか良い python 構文があります。

```
1 n = [i + 4 for i in range(5)]
```

この場合、[4, 5, 6, 7, 8] が帰ってきます。  
この書き方はリスト内包表記と言い、広く使われています。詳しくはググってください。  
他に map という関数があります。これも速いです。上と同じ内容を map で書いてみます。

```
1 def plus4(num: int) -> int:  
2     return num + 4  
3  
4 n = list(map(plus4, range(5)))
```

どっちが良いとかは特にありません...

一々 def で名前付きのを書きたくないなら lambda 式というので一行で出来ます。  
python の lambda 式、読みにくくて嫌われてるけどね。

```
1 n = list(map(lambda x: x + 4, range(5)))
```

---

## numpy(独自のメソッドを実装するときとか)

numpy は速いので、重い演算の時は使えるなら使いましょう。  
上記のリスト内包とか map なんかより numpy の方が圧倒的に速いです。

## 並列化 (これがやりたかった！)

これ、大事！ python での並列化はとても簡単です。  
やり方は色々ありますが、Pool というのがお手軽並列化ツールです。

```
1 from multiprocessing import Pool
```

使い方は、python の map 関数に近いものですので、並列化する時は必ず何か関数を定義して下さい。  
(ちなみに、pool の map は lambda 式を食べることが出来ません)

```
1 def test(i):  
2     return i * 8
```

これを with 文を使って Pool の中にぶちこみます。

```
1 with Pool(4) as pool:  
2     result = pool.map(test, [1, 2, 3, 4])  
3 print(result)
```

ね、簡単でしょ？

これ、test という関数が脳波を解析する関数だったら複数人の  
脳波解析を同時進行できて爆速です！  
(一人分をマルチプロセスするとメモリの取り合いが生じて遅くなる)

ここでは map\_async という関数を使う方法もあります。  
map\_async は map よりも頭の良い並列化関数です。  
map は全員一斉にやる感じ、map\_async は全員でやるけれど、  
終わった人は次の課題をし始める感じです。

```
1 with Pool(4) as p:  
2     result = p.map(test, [1, 2, 3, 4]).get()  
3 print(result)
```

map との違いは、p.map\_async(hoge).get() というふうに get してあげないと結果が得られないことです。

さて、map も map\_async も今の所引数が 1 つのものじゃないと無理です。  
複数ある場合は、starmap というのがあります。

---

とりあえず、starmap も starmap\_async もあるけど、async で書いてみます。

```
1 from multiprocessing import Pool
2
3 def test(x, y): return x + y
4
5 with Pool(4) as p:
6     result=p.map_async(test, [(1, 2), (4, 6)]).get()
7 print(result)
```

良いですね！ これで君の python はコア数に比例するスピードを得たのであります。<sup>81</sup>

## OPENMP のオーバーヘッド対策

並列化にも色々あります。numpy 内部でも並列化がされているのですが、並列化自体のオーバーヘッドもあります。numpy で並列化出来ない部分が多いです。上記のやり方で手動で並列化する場合、numpy 内部で並列化しないほうが速い状況もあるでしょう。だから、マルチスレッドを切っておく方法を書きます。

bashrc に以下のように書くか

```
1 export OMP_NUM_THREADS=1
```

python で以下のように書くかです。

```
1 from os import environ
2 environ['OMP_NUM_THREADS'] = str(1)
```

## MNEpython の並列化

MNEpython は並列計算に対応しています。凄いですね！

実は、MNEpython の内部の並列化はもっと別の joblib という仕組みで動いています。

ですが、MNEpython は全てを並列化することは出来ません。

例えば maxwellfilter は並列化されないのです。

それに、大きな行列をわざわざ分割して並列化しているので、

並列化自体にコストがかかります。

並列化したら遅くなることすらあります。駄目じゃん！

複数の人についてガン回しする時は多分、自分で並列化したほうが速いです。

つまり、僕としては python のネイティブな並列化の Pool().map がオススメです。

---

<sup>81</sup>実際はオーバーヘッドがあるので比例まではしない。

---

## graph

グラフ理論をご存知でしょうか？ グラフ理論はあれです、ネットワーク図から、何らかの指標を抜き出してくるアルゴリズム群みたいなものです。

ほら、全脳解析でコネクティビティが出るじゃないですか。

それを数値化してくれる便利なやつです。

(僕の中ではその程度の認識。ただし、多分あってる。)

python で graph 理論でなにかやりたい場合はこうです。

```
1 pip install bctpy
```

これで bctpy がインストールされました。

公式サイトはこれです。

<https://github.com/aestrivex/bctpy>

コネクティビティの結果である三角行列を突っ込みたいですね。

突っ込みます。

```
1 import bct
```

例えば conmat という numpy 三角行列があったとして、こいつを放り込むならまずは三角行列を普通の行列にしてやるべきでしょう。

(方向ありの行列なら三角行列にはならないのでそのままでもいいです)

```
1 dcon = conmat + conmat.T
```

global efficiency を重み付けありで計算したいならこうと思います。

```
1 bct. efficiency_wei(bct. distance_wei(dcon))
```

コネクティビティの行列から「距離」を求めてから計算を行います。

すると、Global efficiency の場合にはスカラー値が算出されます。

「このネットワークを表すスカラー値を出す」とか

「この点がどのくらい他と繋がっているか点ごとに出す」的な事をするのがグラフ理論です。(数学者に怒られる表現)

中には、閾値を設けて「これ以上の繋がりが具合なら繋がってる」とみなして計算するやつとかもあります。

この記事見て色々遊んでもいいかもですね。

<https://qiita.com/maskot1977/items/e1819b7a1053eb9f7d61>

---

## おすすめの参考書

ステルスマーケティングです。

- Analyzing Neural Time Series Data: Theory and Practice  
表題見て分かる通り、洋書ですが名著です。英語ですが平易に書かれています。  
amazonでも売ってます。どうすれば良いのかわからなくなった時の道標です。  
MNE-Pythonはどうすれば良いのか分からなくなる事が多いのです。  
買って下さい。
- 事象関連電位—事象関連電位と神経情報科学の発展  
脳波関連の和書の名著です。なのですが、絶版です。古本を見つけたらすかさず買いましょう。  
内容的には凄く難しい数学はなく、実践的です。  
通読向けではありますが、やはり手を動かしながらじゃないときついです。
- 意味がわかる線形代数  
これは異色の本です。多くの線形代数の本は文系や医学生には  
冥王星語でも読んでいるように感じるのですが、  
これは日本語で書いてあります。意味がわかります！  
反面、内容は「分かっている人」からすると薄いでしょう。
- 統計的信号処理 信号・ノイズ・推定を理解する  
MNEって行列代数なのです。算数の基礎がわかってないと分かりません。  
でも、基礎がわかっててもわからない本が普通にあるので悲しい。  
この本は基礎がわかったら何とか読めます。凄く親切に書いてある名著です。  
アホでも読める感じがしました。買いましょう。(理解できるかは別)
- 完全独習ベイズ統計学  
ベイズ統計学に関して直感的に分かる本です。  
この本は上記「意味がわかる線形代数」と同様、  
冥王星語を関西弁くらいまで下ろしてくれます。  
やはり入門書はざっくりしたものが一番です。
- Electromagnetic Brain Imaging: A Bayesian Perspective 2015  
割りと平易な英語で書いてある応用数学の本です。  
MNE, beamformer, dSPM, sLORETA等が載っていて重宝します。  
内容は結構噛み砕いてくれています、元が僕にとっては難しいものなので大変でした。  
算数を理解したいなら買って下さい。
- ゼロから作る DeepLearning  
コラム4参照。ニューラルネットワーク系人工知能本の名著です。

---

機械学習に興味がお有りなら買ってください。

MNE には役立ちません！

基礎を学ぶには本当に良いです。その後は pytorch だとか tensorflow だとか触れば良いんじゃないですかね？

- パターン認識と機械学習 (上下)

通称ぶるむる。

難しくてとてもつらいですが、機械学習方面では聖書の一つです。

この領域では機械学習を使うこともあるので、必要になればどうでしょう？

- Wavelet10 講

直交 Wavelet の本です。

Wavelet 変換のガチ勢になりたければ読むといいです。

リアルタイム解析の世界への展望が開けるかも知れません。

- MNE 同人誌

本書。MNE-python に関する、多分世界で唯一の和書。

## おすすめサイト

高校数学の美しい物語 <http://mathtrain.jp>

高校数学についてのサイトです。高校数学を復習するにあたって、このサイトは素晴らしい。

フーリエ変換とか行列計算とか複素数とか、そういったことを考える時に

辞書みたいにつかってみては如何でしょうか？

ウェーブレット変換の基礎と応用例:連続ウェーブレット変換を中心に <https://www.slideshare.net/ryosuke-tachibana12/ss-42388444>

このウェーブレット変換スライドは素晴らしいです。作者が学生の頃に勉強して作ったのだそうですが、多くの「分かっている人向けの数学的入門書」と違い、直感的に分かるように書いてあるのです。

## おすすめ SNS

qiita <http://qiita.com>

日本のプログラマ用の SNS...というかブログサービスです。

かなり分かりやすい記事が多く、大変重宝します。

反面、コピペプログラマになるのを避ける心がけは大事ですね。

カジュアルな雰囲気漂う気軽なサイトです。

---

twitter <http://twitter.com>

twitter かよ！と思われるかと思いますが、学者さんのアカウント、雑誌のアカウント  
開発者さんのアカウントは極めて有用かつ濃密です。

カジュアルな感じですが、酔っ払って書き込んで炎上させるのには気をつけましょう。

github <https://github.com>

プログラマ用 SNS の中でも最も有名なものでしょう。

qiita はただの記事集ですが、github は開発ツールです。ただし、学習コストが高いですね。

下記の git を中心に据えた web サービスです。

余談ですがマスコットキャラの octocat が可愛いです。

## おすすめソフト

これまで散々色々なソフトを紹介してきましたが、

それ以外のツールも紹介しておきましょう。

当然ながら全てフリーウェアです。

- edfbrowser

edf ファイルの変換に使えることがあります。

windows 用ですが、コンパイルすれば Unix 系でも使えます。

- git

バージョン管理ソフトです。プログラム書くときとか、長文書くときとかに

どこをどう修正したのか分からなくなったりしませんか？

また、修正したいけど壊しそうで怖いからコピーにとって修正したりしていませんか？

そんな貴方に必要なのはバージョン管理ソフトでしょう。

git は最も有名なバージョン管理ソフトの一つです。

- pandoc

markdown という形式で書いた文書をあらゆる形式に変換するソフトです。

詳しくはコラム参照。これの何が嬉しいかというと、markdown で書いたものを

word、LaTeX、PDF、HTML 等、あらゆる形式に変換してくれるのです。

つまり、markdown さえかければ、他はいらなかった！

(細かい所の調整は LaTeX 書く必要が有ることもありますが...)

何故これを勧めるかって？ Jupyter や Github は markdown を多用するからだよ。

---

## MNEpython 実装時の小技

一応、実装が苦手な人が読者と思っているので、ありふれた小技ですが紹介します。  
Object 指向とか関数型とかは他の本を読んで下さい。得られるものが多いでしょう。

### メソッド・チェーン

今回は超手軽に解析してみましょう！  
何度もフィルタ掛けるの面倒くさいから、一気にかけちゃう方法です。  
メソッド・チェーンを使います。  
メソッド・チェーンとはドットで数珠つなぎに処理をしていく技法です。  
MNEpython では raw オブジェクト辺りで割とできる感じです。実際見てみましょう。

```
1 from mne.io import Raw
2 Raw('hoge.fif').filter(1, 100).notch_filter(60).save('fuga.fif')
```

どんだけ略してんだよ！ というくらい略されてますね。  
このケースでは、読み込んでフィルタを2つ掛けて保存しています。  
まあ、使いすぎは色々大変になるので良くないです。

### 変数を減らしてみる

raw を弄る時 raw.filter 関数などを使うと raw 自体が書き換わってしまいます。  
これ自体は正しい動作なのですが、一寸わかりにくさを感じるかも知れません。  
raw は raw としてどっしり構えてもらって、加工品だけ作って行きたいかも知れません。  
そんなときは raw.copy 関数がいいです。

```
1 raw2 = raw.copy()
2 raw3 = raw2.filter(1, 100)
3 raw4 = raw3.notch_filter(60)
```

これで raw の copy が出来ましたね。しかし、どうも変数が多くなります。  
raw2, raw3, raw4 と作るうちに raw $\infty$  とかなって死にます。  
その対策にはメソッドチェーンがいい味を出すと思っています。

```
1 filtered = raw.copy().filter(1, 100).notch_filter(60)
```

raw2 など要らなかった。



---

## MNE の API 引数多すぎだろ死ね！

確かに MNE の method は引数が多すぎである。

引数が多すぎて毎回引数入れるのがダルいし、ミスも多くなりそうだ。

だが、落ち着いて聞いてほしい。python には良い道具があるのだ。

```
1 from functools import partial
```

こいつは関数を部分的に解いちゃう関数だ。

今君は、複数の epoch オブジェクトを作りたいとする。event\_id は 1、2、3、4、5、6 だとする。

その都度入力するのはダルいし、変数が増えすぎると管理も大変だ。そんなときはこのようにすればいい。

```
1 from mne.io import Raw
2 from mne.epochs import Epochs
3 from mne import find_events
4
5 raw = Raw('hoge.fif', preload=True)
6 events = find_events(raw)
7 make_my_epochs = partial(EPOCHS, raw, events)
```

これで make\_my\_epochs という割と決め打ち的な関数が出来た。以降は例えば

```
1 make_my_epochs(4)
```

とかで event\_id が 4 の epoch オブジェクトが返る。これで君の怒りが少しでもおさまってくれたら嬉しい。

ここまでのまとめ

というわけで、凄く省略すれば、epoching まで下記のように書けるのです。

```
1 from mne.io import Raw
2 from mne import Epochs
3 raw = Raw('hoge').interpolate_bads().filter(1, 100).notch_filter(60)
4 make_epochs = partial(EPOCHS,
5                       raw, mne.find_events(raw),
6                       tmin=-0.2, tmax=5.0)
7 epochs = [make_epochs(n) for n in range(1, 7)]
```

まあ、ICA とかが省いているから本当はもうちょっと長いです。

---

解析失敗したやつをスキップしたいんだが

気持はよく分かる。たまに失敗した実験が紛れてたりするんですよね。

そんな君には try 文と filter 関数をおすすめしましょう。これにより、だいぶ楽になります。

**try 文** やってみて、失敗したらやめるというやつ。

```
1 try:
2     raw = Raw('hoge.fif')
3 except FileNotFoundError:
4     return None
```

こんな感じで使います。これで万一どこかで失敗しても処理が止まらない！

エラーの書き方は、まあ、ググって調べてください。

**filter 関数** 例えば、何らかの理由で epochs を作れなかった raw があったとします。

(トリガーが入ってなかったとか、色々あると思う)

そんなのが紛れ込んでで for 文が動かんくなったら糞面倒くさいです。

ここで、os.path モジュールの exists 関数<sup>82</sup>を

filter 関数や lambda 式<sup>83</sup>と組み合わせて使うといいです。

filter 関数は list や tuple の中で、条件に合うやつだけを抜き出すものです。

これは高階関数といって、関数を引数に取る関数です。

こんなかんじ。filter(関数, list)

では今回は存在する raw だけを抜き出すという操作をやってみましょう。

```
1 from os.path import exists
2
3 file_list = ['hoge', 'fuga', 'piyo']
4 fnames = list(filter(exists, file_list))
```

これで存在するものだけを読み込めます！ 成功例のみ続けていけますね！

でも、「どれが読み込めたか分からない」って思いましたか？

大丈夫。epochs とかのオブジェクトにはたいてい filename 的なメンバー変数が入っているからそれを参照して下さい。

つまり、こんな感じですね。

```
1 from mne import read_epochs
```

---

<sup>82</sup> というか、bool 型を返してくれるやつなら何でも行ける。

<sup>83</sup> 無名関数。1 行の使い捨てのやつ。

---

```
2
3 file = 'hoge'
4 epochs = read_epochs('file')
5 ...
6 print(epochs.filename)
7 epochs.save(f'modified-{epochs.filename}')
```

Raw の場合は連結できたりする関係上、filenames のようです。

他に map 関数とか、reduce 関数も時に有用です。

MNE 使う時は割と関数型パラダイムは有効っぽそうです。ただ、気をつけて下さい。

map とか filter とかは一度値を取り出すと空っぽになります。

使いまわす場合は list とかに一々保存したほうが良いでしょうね。

また、上記の方法ではファイルが壊れていた場合に対応できないので、結局は try 文を使うことになるかと思います。

**file** 名じゃなくてフォルダ名が欲しいん

概ねこんな感じでゲットできます。

```
1 from pathlib import Path
2 path = Path(epochs.filename).parent
3 dirname = str(path)
```

こういう小技、大事ですよーいや、工学部の人が良いんだけどさー

---

## あとがきとお願い

### 令和 2 年 2 月時点のあとがきだ

こんにちは、ニンジャというしがない同人作家です。

この度はこのような分厚い薄い本を読破していただきありがとうございます。

(まあ、読破はしてないと思うし、その必要もない)

本作は僕の初めての技術系同人誌です。大学院でほぼ独学で頑張った時のメモをベースにしました。

自画自賛しますがマジで力作と思う。ちなみに博士課程 5 年目ですが博士論文は書いてません wwwwww

まだまだ気が向いたら加筆修正していくつもりで、github に随時 up ってるので、

「君は古いのを掴まされたんだよ、プギャー wwwwww」と言えなくもない。

この本自体は入門書であり、紹介した機能はごく一部です。もっと色々出来ることがあります。

虎の威を借りますが、MNE-python はマジで凄いです。凄いですが、不親切極まりないです。

MNE を使いこなすには修行が必要で、君の MNE ライフはこの本を

「は？ 当たり前のこと書いてんじゃねえよ糞ボケ死ね！

つか、ここ間違ってるし、作者の脳はチンパンジー以下だな！」

と思うレベルのところから始まるのであります。

出来れば、読者の中の誰かが高名な脳科学者 (このワード胡散臭くて好き) になって頂けると、

一人の落ちこぼれ大学院生が浮かべられます。どうか精進なさってください。

令和 2 年 2 月 ニンジャ

### 令和 3 年 5 月時点のあとがきだ

この度更新いたしました。致したんですよ？

令和 2 年の頃の僕は今の僕よりもっとクソザコで、間違いを色々書いていた！

上に書いた通り、チンパンジー以下だと思います。

恥ずかしいです。昔を持っている人は必ず更新して下さい。

(これを手にしている人は既になっているので無意味土下座)

それで、僕は今でもクソザコ留年生です。

せめて、読者である君！ 君が強い脳科学者になるんだよ！

じゃないと浮かべられません。ほんとお願ひします。(土下座)

令和 3 年 5 月 ニンジャ

---

令和 5 年 6 月時点のあとがき

8 年目大学院生のニンジャと名乗るチンパンジーです。  
マジ爆発寸前。

この度再改訂致したのは、チンパンジーの本を読んだ人が居たのと、  
チンパンジー的に人と情報交換するために必要だったから。

今回、この本を改訂するために見直しましたが、  
...やはりチンパンジーが書いた感じではある。

それでも MNE-python の為の唯一の和書として本邦の脳科学の  
発展に寄与できればうれしいね。

令和 5 年 6 月 チンパン・ニンジャ