Table of Contents

[はじめに 6](#_Toc510711105)

[本書の目的・限界・対象者・使い方 6](#_Toc510711106)

[センサーベース/ソースベース解析とはなんぞや 7](#_Toc510711107)

[スラングや用語の説明とunix系のお約束 8](#_Toc510711108)

[それぞれの研究に必要な環境・特色のまとめ 8](#_Toc510711109)

[MNE/pythonとは 9](#_Toc510711110)

[freesurferとは 10](#_Toc510711111)

[それぞれのソフトの関係性〜それofficeに例えるとどうなの？〜 11](#_Toc510711112)

[MRIと脳磁図計とELEKTA製ソフトの準備 11](#_Toc510711113)

[脳波計 11](#_Toc510711114)

[コンピュータの準備 12](#_Toc510711115)

[OSの準備 13](#_Toc510711116)

[開発環境の準備（脳波、脳磁図の場合。出来るだけリッチに。） 13](#_Toc510711117)

[jupyterの設定(やり得) 17](#_Toc510711118)

[jupyterでのplot 17](#_Toc510711119)

[python2,3二刀流(現時点では不要) 17](#_Toc510711120)

[Rをjupyterで動かすために(非常に便利) 18](#_Toc510711121)

[CUDA 19](#_Toc510711122)

[バージョン管理git 21](#_Toc510711123)

[gitサーバー 21](#_Toc510711124)

[gitbucket導入 21](#_Toc510711125)

[jupyterで作ったスクリプトのバージョン管理(小技) 22](#_Toc510711126)

[方法1 22](#_Toc510711127)

[方法2 23](#_Toc510711128)

[実験のディレイの測定 23](#_Toc510711129)

[必要物品 24](#_Toc510711130)

[maxfilterのインストール(MEG研究の場合) 28](#_Toc510711131)

[freesurferを使う(MRI) 29](#_Toc510711132)

[recon-all同時掛け(freesurfer) 29](#_Toc510711133)

[freesurferの解析結果の表示 30](#_Toc510711134)

[解析結果の修正の必要性 30](#_Toc510711135)

[画像解析の修正 31](#_Toc510711136)

[SkullStripのエラー 31](#_Toc510711137)

[脈絡叢の巻き込み 32](#_Toc510711138)

[眼球が白質と間違われた時 32](#_Toc510711139)

[頭蓋骨と間違って脳をえぐっているとき 32](#_Toc510711140)

[白質の内部に灰白質があると判定されるとき 33](#_Toc510711141)

[白質が厳しく判定されているとき 33](#_Toc510711142)

[MNE/pythonのインストール(脳波、脳磁図をする場合) 33](#_Toc510711143)

[MNE/Cのインストール(脳波、脳磁図をするばあい) 34](#_Toc510711144)

[*コラム1-SNSの活用* 34](#_Toc510711145)

[freesurferを使う(MRI) 35](#_Toc510711146)

[recon-all同時掛け(freesurfer) 35](#_Toc510711147)

[freesurferの解析結果の表示 36](#_Toc510711148)

[解析結果の修正の必要性 36](#_Toc510711149)

[画像解析の修正 36](#_Toc510711150)

[SkullStripのエラー 37](#_Toc510711151)

[脈絡叢の巻き込み 37](#_Toc510711152)

[眼球が白質と間違われた時 38](#_Toc510711153)

[頭蓋骨と間違って脳をえぐっているとき 38](#_Toc510711154)

[白質の内部に灰白質があると判定されるとき 38](#_Toc510711155)

[白質が厳しく判定されているとき 39](#_Toc510711156)

[mricronのインストールと変換(MRIを使う場合) 39](#_Toc510711157)

[mricronによるMRIのファイルの変換 39](#_Toc510711158)

[MNEを使う 42](#_Toc510711159)

[jupyterの起動 42](#_Toc510711160)

[MNEpythonを使う前に学んでおくべきモジュール 43](#_Toc510711161)

[numpyで遊ぼう 43](#_Toc510711162)

[データの読み込み…の前にmaxfilter(脳磁図の場合) 44](#_Toc510711163)

[解析を始める前のwarning! 45](#_Toc510711164)

[pythonでの高速化のあれこれ 45](#_Toc510711165)

[for文とリスト内包表記 45](#_Toc510711166)

[numpy 45](#_Toc510711167)

[並列化(白魔術、弱) 46](#_Toc510711168)

[クラスタレベルの並列化(白魔術、強) 46](#_Toc510711169)

[Cython(黒魔術、使いこなせば相当強い？) 48](#_Toc510711170)

[C言語、C++、FORTRAN(最終兵器) 49](#_Toc510711171)

[作図用おまじないセット 49](#_Toc510711172)

[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) 50](#_Toc510711173)

[データの読み込みとfilter,resample(僕の解説) 52](#_Toc510711174)

[脳波読み込みの問題 53](#_Toc510711175)

[脳波のセンサーの位置が変則的な場合 54](#_Toc510711176)

[脳波のセンサーの名前が変則的な場合 54](#_Toc510711177)

[基準電極 55](#_Toc510711178)

[トリガーチャンネル 56](#_Toc510711179)

[bad channelの設定 57](#_Toc510711180)

[ICAをかけよう 57](#_Toc510711181)

[EpochとEvoked 60](#_Toc510711182)

[データのplot、主にjupyter周り、そしてPySurfer 61](#_Toc510711183)

[多チャンネル抜き出し 65](#_Toc510711184)

[センサーレベルwavelet変換 66](#_Toc510711185)

[そもそもwavelet変換とは何なのか 66](#_Toc510711186)

[wavelet変換にまつわる臨床的な単語 67](#_Toc510711187)

[wavelet変換の実際 68](#_Toc510711188)

[データの集計について 69](#_Toc510711189)

[Rとpadasの連携、特にANOVAについて 72](#_Toc510711190)

[Connectivity 73](#_Toc510711191)

[5つの返り値 75](#_Toc510711192)

[fourier/multitaperモード 76](#_Toc510711193)

[waveletモード 76](#_Toc510711194)

[plot 77](#_Toc510711195)

[indicesモード 77](#_Toc510711196)

[ソースレベルMEG解析 78](#_Toc510711197)

[手順1、BEM作成 79](#_Toc510711198)

[手順2、ソーススペース作成 80](#_Toc510711199)

[手順3、trans 81](#_Toc510711200)

[手順4、順問題 82](#_Toc510711201)

[手順5、コヴァリアンスマトリックス関連 83](#_Toc510711202)

[手順6、逆問題 84](#_Toc510711203)

[手順7 ソース推定 85](#_Toc510711204)

[手順8、前半ラベル付け 86](#_Toc510711205)

[手順8後半、label当てはめ 86](#_Toc510711206)

[その後のお楽しみ1、ソースベースwavelet 87](#_Toc510711207)

[その後のお楽しみ2、ソースベースconnectivity 87](#_Toc510711208)

[*コラム3-markdownで同人誌を書こう！* 88](#_Toc510711209)

[初心者のための波形解析 88](#_Toc510711210)

[波形解析で得たいものと、必要な変換 89](#_Toc510711211)

[フーリエ級数 90](#_Toc510711212)

[複素フーリエ級数 90](#_Toc510711213)

[結局何をしているのか 93](#_Toc510711214)

[臨床的な数値を算出する 93](#_Toc510711215)

[そしてwavelet変換へ 94](#_Toc510711216)

[*コラム4-ニューラルネットワーク* 94](#_Toc510711217)

[Minimum-norm-estimationの理屈 95](#_Toc510711218)

[MNEの重み付け 98](#_Toc510711219)

[L2ノルムMNEとベイズ推定 99](#_Toc510711220)

[dSPMの理屈 100](#_Toc510711221)

[おすすめサイト 102](#_Toc510711222)

[おすすめSNS 103](#_Toc510711223)

[おすすめソフト 103](#_Toc510711224)

[参考文献 104](#_Toc510711225)

# 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。  
さぁ、MNE/python、freesurferの世界で良い生活を送りましょう！

…というか、周りにMNEpython使いほとんど居ない…一人じゃ辛い。

## 本書の目的・限界・対象者・使い方

MNE/pythonやfreesurfer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elekta社のMEGを使っているのでelekta前提で書きます。

本書の限界は僕のスキル不足（１年と少しかやっていない）と、これが同人誌であることです。  
不確実なものとして、疑って読んでいただければ幸いです。

本書の対象者は以下のとおりです

* 脳磁図計を使って脳の生理学的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
* 脳波研究で、応用的な解析をしたい初心者  
  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。  
  (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](#MNEpython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](#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[[1]](#footnote-1)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[[2]](#footnote-2)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[[3]](#footnote-3)。  
普段計測している脳波・脳磁図は所詮は「漏れでた信号」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**ない**[[4]](#footnote-4)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モジュール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も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拡散してしまう性質があります。  
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。[[5]](#footnote-5)

## 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。  
適宜読み替えていってください。それ以外にも色々スラングあるかもです…。

* 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。
* 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。

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  
文脈上特に何もない場合、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。

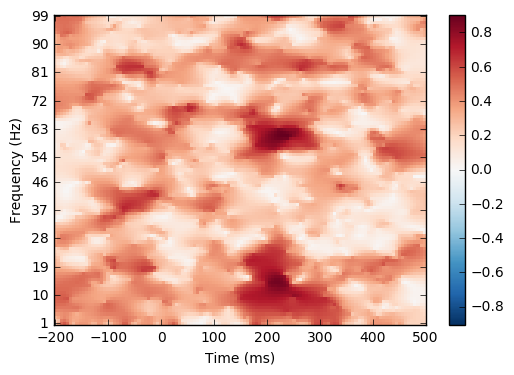
## それぞれの研究に必要な環境・特色のまとめ

本書ではまず環境を構築しますので、色々インストールが必要です。  
必要物品についてまとめると下記です。

* 脳波研究  
  python(本書ではanaconda使用)、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。  
  脳の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。
* MEGセンサーレベル研究  
  python(本書ではanaconda使用)、MNE/python、MNE/C、maxfilter  
  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。  
  それさえクリアできれば処理の重い脳波みたいなものです。  
  ※ノートでは厳しいです。
* MRI研究  
  freesurfer、mricron  
  莫大な資金が必要ですが、それなりに普及しています。  
  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。  
  ※ノートでは無理です。  
  ※グラフ理論で解析する場合はpython必要
* MEG+MRIソースレベル研究  
  MEGとMRIを組み合わせた応用編となるため、紹介したものの全てが必要です。  
  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†**　…のはずなんですが、どうなんでしょうね？  
  あまり頭の良くない僕にはこの理論はあまりに難しく、頭がクラクラします。  
  ※膨大な計算量が必要なため、でかいコンピュータが必要です。

## MNE/pythonとは

脳磁図を解析するためのpython[[6]](#footnote-6)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。)  
wavelet変換、コネクティビティ、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。  
C言語で実装されたMNE/Cというのもあり、脳磁図やるなら併用する必要があります。  
最近はMNEpythonに機能を移しているそうですが、移行はまだ完全ではないからです。  
両方共フリーウェアですが、MNE-Cは登録が必要です。  
開発は活発で、最近新バージョンは[MNE/python 0.15](http://martinos.org/mne/stable/index.html)です。  
freesurferは[6.0](https://surfer.nmr.mgh.harvard.edu/)が、pythonはpython3.6が最新です。  
導入と紹介を書いていこうと思います。ちなみに、諸事情ありpython2.7を使います。  
python3.6は使いません。(そのうち使えるようになります)



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。  
反面、激重な上にサイズが大きくbash使う必要があります。  
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？〜

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す。  
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 |
| 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 | 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 | 何でも一人でこなそうとする部下 |

## MRIと脳磁図計とELEKTA製ソフトの準備

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

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。  
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[[7]](#footnote-7)とか。

## 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。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。  
その場合はワークステーション…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。[[8]](#footnote-8)  
メモリいっぱい、CPUは多コアがいいです。ソースベース解析するならnvidiaのGPUとか  
載ってるやつも良いかもしれないです[[9]](#footnote-9)。どの程度のものが必要かは実験系によります。  
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。

## OSの準備

OSはlinuxかMACが普通と思います、windowsではどうなのでしょう？[[10]](#footnote-10)  
よく分かりませんが、MNEpythonは動きます。  
僕は新しめのdebian系linuxであるUBUNTU[[11]](#footnote-11)またはMAC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を勧める理由は  
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)を用いることになります。  
以下、UBUNTU16.04LTSかmacos10.12を想定して書いていきます。  
UBUNTU16.04LTSは下記サイトから無料でダウンロードできます。

https://www.ubuntulinux.jp/ubuntu

僕自身は少しでも速く処理して欲しいので、誤差範囲かもですが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。MACを使う場合はhomebrewというパッケージマネージャを  
インストールすると色々楽になることがあります。  
https://brew.sh/index\_ja.html

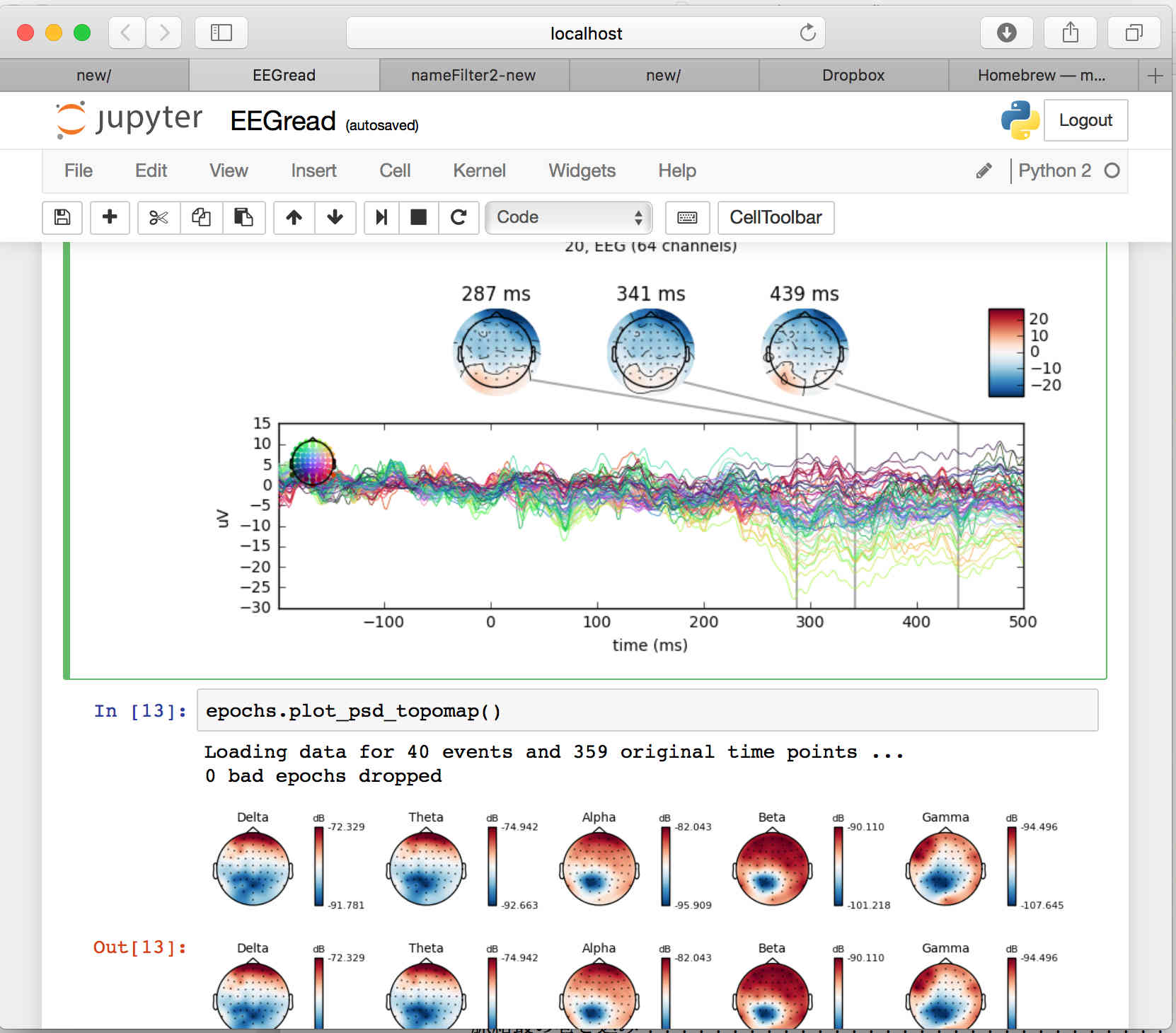
# 開発環境の準備（脳波、脳磁図の場合。出来るだけリッチに。）

* 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。

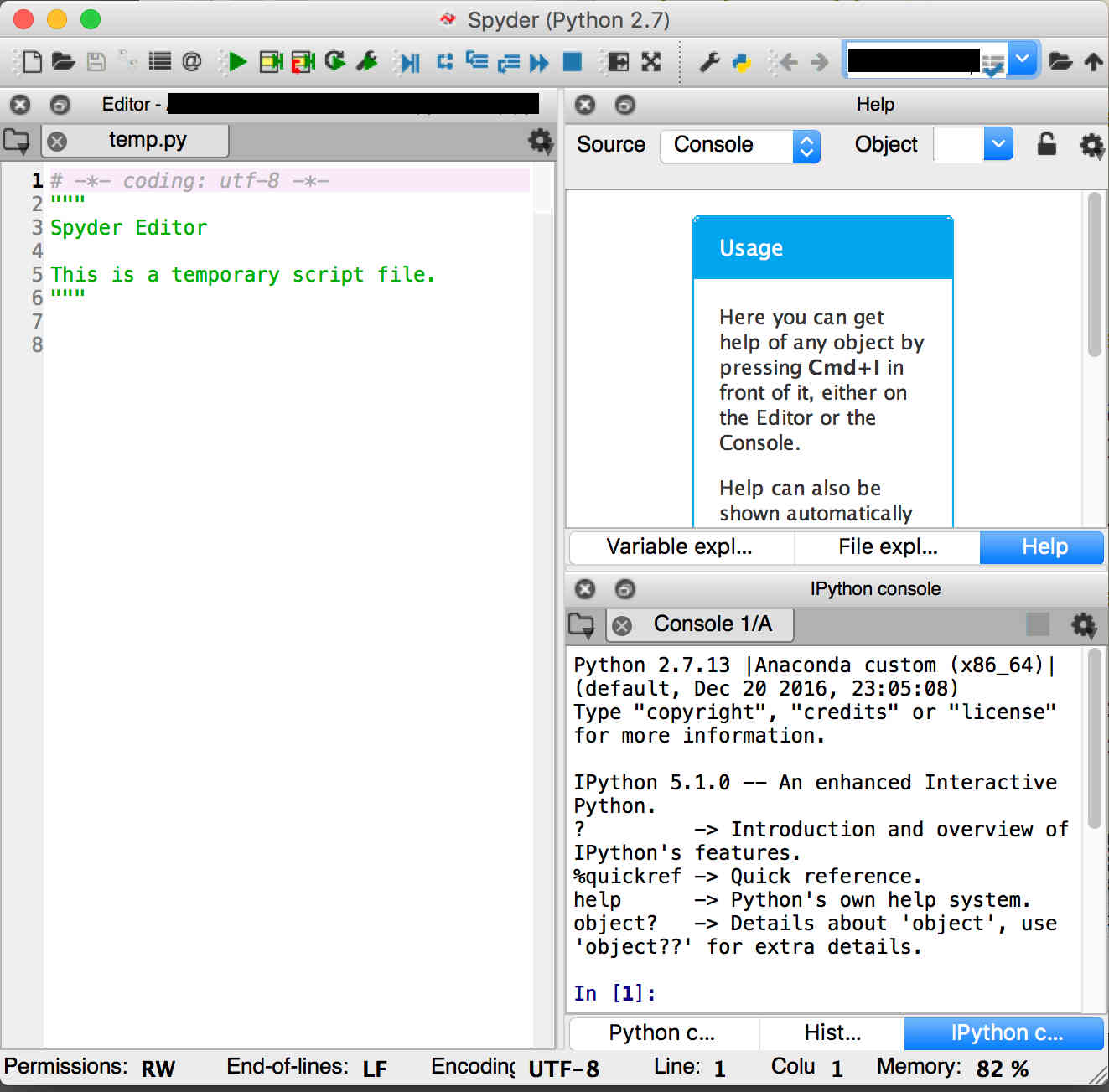
開発環境はMNE使うなら必要です。詳しい人からは「docker[[12]](#footnote-12)じゃダメなん？」という質問が  
来そうですが、ライセンスの問題がありますし、セットアップは自分でできなければ  
困ることもありましょう。僕はanaconda[[13]](#footnote-13)を使います。何故ならインストールが楽だからです。

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は2と3があり、それぞれpython2と3に対応しています。どちらを選ぶか悩ましいです。  
3は現時点ではソースベース解析出来ませんが、そのうちpython2自体のサポートが切れます…。  
anacondaにspyderというIDEが付いてくるのでそれを使うのもいいでしょう。  
spyderは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。  
しかし、僕のおすすめはjupyter+anaconda2で多言語切り替えながら使う方法です。  
jupyterはwebベースの開発環境で、他の人とデータやソースコードの共有が容易なので捗ります。[[14]](#footnote-14)



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。詳しい人に「githubじゃダメなん？」と言われましたが、ここでいう共有ってそういう意味じゃないんですよ。非プログラマの先輩が後輩に教えながらコーディングする小さなラボで…という意味ですよ。まぁ、触ってみればわかります。また、本格的なソースの共有はできないので、git併用するのが良いかと思います。



spyderの画面。ごく普通の素直な挙動のIDE。jupyterより動作が軽いのも魅力。一人で解析するならこれも良いものです。

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なります。  
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。

bash Anaconda3-4.3.0-Linux-x86\_64.sh

インストール先はホームフォルダでいいかとか、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。

## jupyterの設定(やり得)

素のjupyterでも強力ですが、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。

conda install -c conda-forge jupyter\_contrib\_nbextensions  
jupyter contrib nbextension install --user  
ipcluster nbextension enable --user

これでextensionが使えるようになります。jupyterは機能が拡張できるので便利です。

## 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上に書いておいてください。

jupyter上に直接出力したい時

%matplotlib inline

python2環境下で別ウィンドウで表示したい時  
これはスクロールが必要な時に便利です。

%matplotlib qt

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。

%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)

%gui qt

これについては後でまた詳しく記載します。

## python2,3二刀流(現時点では不要)

このセクションは必要ありませんが、将来的にきっと  
python2から3に移行する時がMNEにもやってくると思いますので記しておきます。

python2と3を二刀流すべくキーボードを叩きます。現時点ではmayaviと言う  
結果の図示に必要なモジュールが古い環境でしか動かず、通常はpython2系を使うべきと考えます。  
以下、anaconda2から入れた場合です。逆の場合python2.7と読み替えて下さい。

ipython kernel install --user  
conda create -n python3 python=3.6 anaconda  
source activate python3  
ipython kernel install --user  
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python2の環境をjupyterに載せておく
2. condaで別バージョンのpython環境を作る
3. 切り替える
4. jupyterに組み込む
5. 確認

source activateコマンドでpythonの環境を切り替えられます。これでjupyterでpython2と3を  
切り替えられると思います。ちなみに間違って環境を作った場合は以下のコマンドで消せます。

conda remove -n python3 --all

## Rをjupyterで動かすために(非常に便利)

anacondaを使っているなら下記でRがインストールできます。

conda install libiconv  
conda install -c r r-essentials  
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」[[15]](#footnote-15)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

%load\_ext rpy2.ipython

とした後

%%R -i input -o output  
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。

import pandas as pd  
data=pd.Dataframe([二次元配列])

%%R -i data  
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。

## 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。  
つまり、GPGPUです。  
これはMNEpythonでも使うことが出来るので、やってみましょう。  
これのインストールも詰まるとそれなりに面倒です。  
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。

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)

sudo dpkg -i cuda-repo-ubuntu1604\_9.1.85-1\_amd64.deb`  
sudo apt-key adv --fetch-keys http://hogehoge.pub  
sudo apt-get update`  
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。  
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。

export PATH=/usr/local/cuda-9.1/bin${PATH:+:${PATH}}  
export LD\_LIBRARY\_PATH=/usr/local/cuda-9.1/lib64\  
 ${LD\_LIBRARY\_PATH:+:${LD\_LIBRARY\_PATH}}

これでCUDAへのリンクが貼れたはずです。  
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

sudo apt-get install nvidia-cuda-dev nvidia-modprobe  
git clone http://git.tiker.net/trees/pycuda.git  
cd pycuda  
./configure.py --cuda-enable-gl  
git submodule update --init  
make -j 4  
python setup.py install  
cd ..  
git clone https://github.com/lebedov/scikit-cuda.git  
cd scikit-cuda  
python setup.py install

これでインストールできてたら成功です。  
pythonで

import mne  
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。

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！

# 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しする時、引き継ぎとかも大変です。  
だから、貴方はgitを使ってください。  
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。  
http://www.backlog.jp/git-guide/

## gitサーバー

git単体でもいけるのですが、折角だからgitのサーバーを導入してみましょう。  
一番いいのはgithubのプライベートリポジトリを使うことなんですが有料です。  
他にもbitbucketだとかgitlabとか色々あるのですが…僕自身は研究室のローカルなサーバーで  
実現したかったんです~~趣味もある~~  
なので、gitbucketを採用しました。gitbucketはgithubのクローンを目指して開発されたものです。

### gitbucket導入

gitbucketをググってgitbucket.warをダウンロードしてください。  
で、javaというか、jdkをインストールします。めんどいんで詳しくはググってください。  
http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

このまま

java -jar gitbucket.war

でも動くのですが、安定性に欠けるらしいので僕はPostgreSqlを導入します。

とりあえず、いくつかunixユーザーを作りましょう…。  
そんで、データベース上に自分とgitbucketを登録します。

useradd postgres  
passwd postgres  
su postgres  
createuser -d hoge  
createdb hoge  
createdb gitbucket  
exit

上記で一応なんとかなると思うのですが、念のため確認を…  
postgresqlにログインして下記を叩けばちゃんとデータベースが出来たかを確認できます。

\du

.gitbucket/database.confを下記のように書き直します。

db {  
 url = "jdbc:postgresql://localhost/gitbucket"  
 user = "test"  
 password = "test"  
}

で、

java -jar gitbucket.war

これでip+:8080にアクセスすればgitbucket動いてます。  
(もちろん、これだけではセキュリティ面等、不十分です。  
セキュリティ詰める自信がないならローカルだけで使いましょう。)

## jupyterで作ったスクリプトのバージョン管理(小技)

jupyterを僕は使いますが、jupyterのファイルはgitしにくいです。  
でも、何とかなります。

方法は2つありますが、僕は方法2が楽でいいと思っています。

### 方法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/hoge/.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[[16]](#footnote-16)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。  
どちらかというと素直にpyファイルにしてダウンロードしてgitを使うほうが良いかもしれません。

### 方法2

gitを使いますが、git側の設定だけでもどうにかなります。  
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。

\*.ipynb diff=ipynb

そして、下記を.git/configに

[diff "ipynb"]  
textconv=jq -r .cells[] |{source,cell\_type}  
prompt = false

下記を.gitignoreに

.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。

# 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。

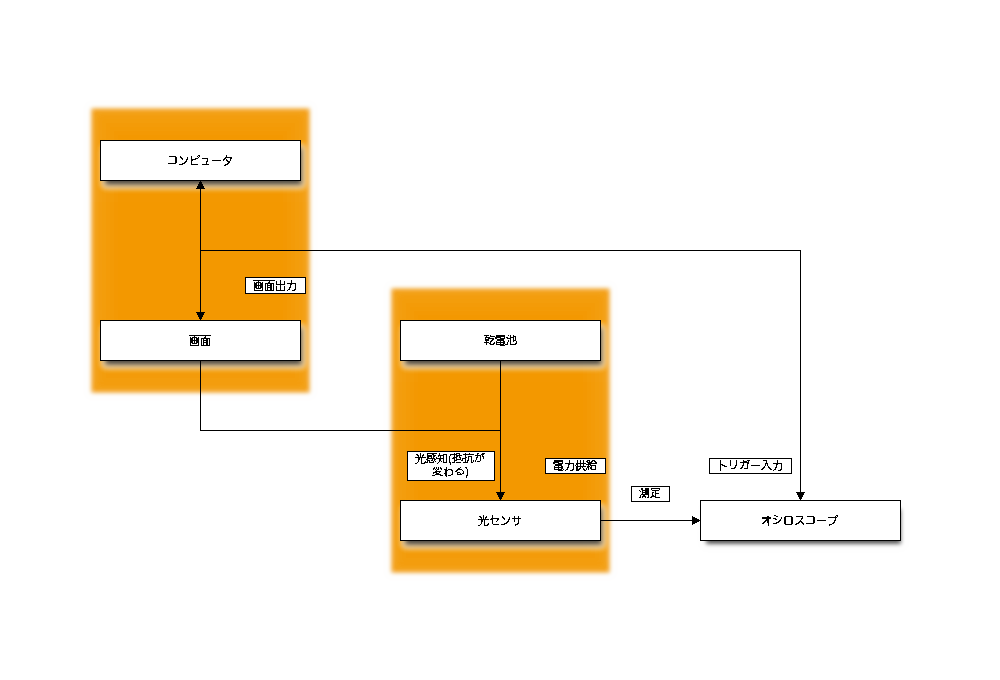
ここでは僕がどうやったかを書いておきます。  
僕がやったのは被験者の目の前の画面に図が表示されるまでと、  
「画面提示」の信号が脳磁図計に届くまでの差を調べることです。

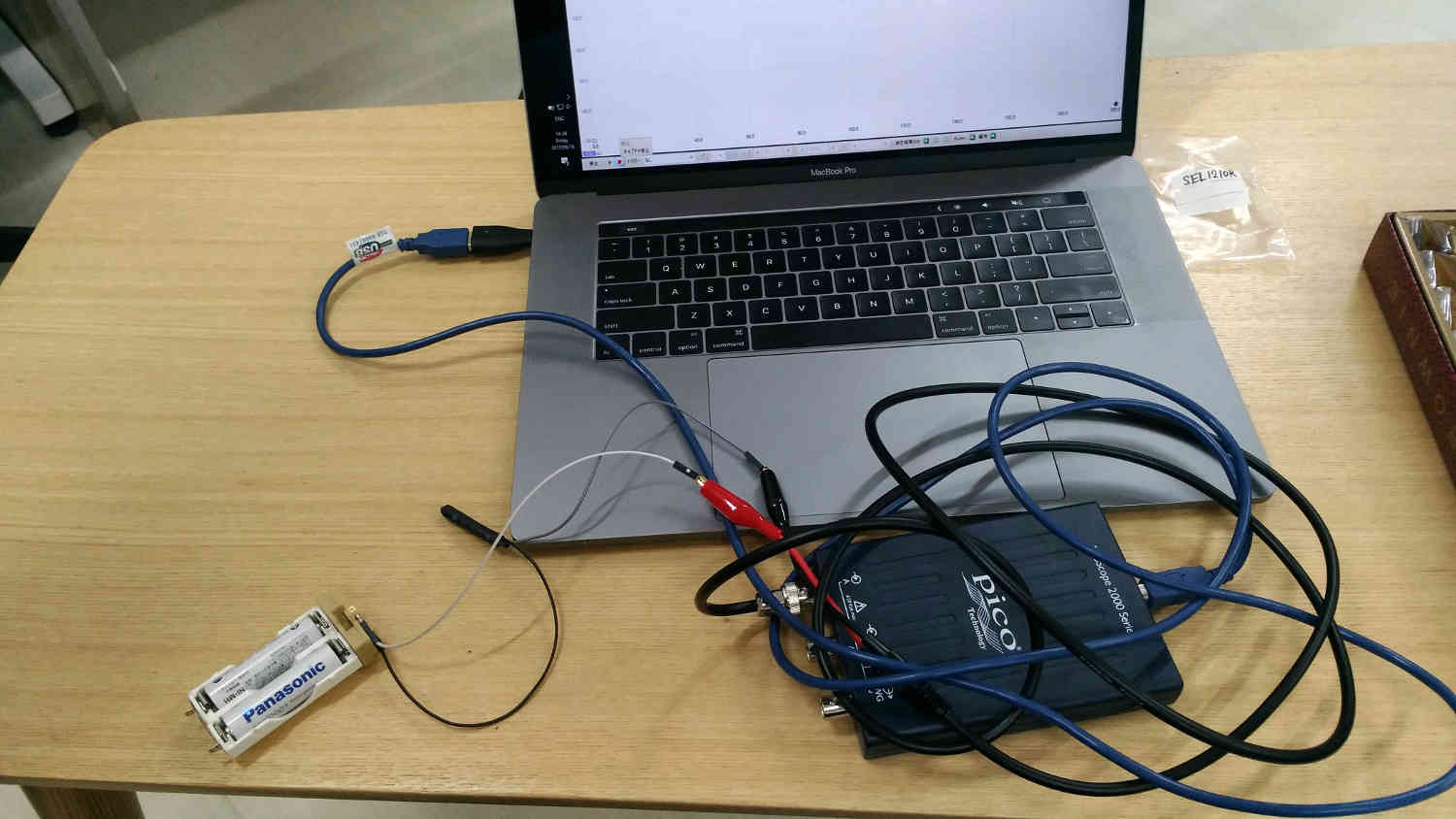
## 必要物品

* 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、μs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。  
  他の通販サイトにはあるようです。スペック上μs単位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。  
  あまり逆の電流は流さない方がいいのかもしれません。
* 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、鰐口クリップ付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ぎます。  
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ぎます。  
さらに、ディレイ測定用コンピュータにも繋ぎます。  
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…  




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いので、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。

## コンピュータクラスタ(計算量が膨大な場合)

これは無いならないでもいいですが、解析したい人数や観察したい場所が  
多くなるなら役に立つかもしれないので記しておきます。科学計算はときに膨大な演算が  
必要になります。ちなみに僕は今まで最長で12コアのワークステーションで3日かかっています。  
ただし、パソコン初心者とかにとっては敷居高いかもしれません。  
以下、UBUNTU16.04を想定して書きます。

やり方としてはまずnfsを使ってディスクを共有します  
僕はホームディレクトリをそのまま共有しました。共有元ではこうです。

apt install nfs-kernel-server

次に、/etc/exportsを書き換えます。

/home/hoge/ fuga(rw)

ディレクトリ、ip、オプションの順です。詳しくはググって下さい。

次に共有先のコンピュータをいじります。  
まず、他のコンピュータのディスクを読むためのソフトをインストールします。

apt install nfs-common

そして、起動時にそのホームディレクトリを読みに行くように設定します。  
具体的には/etc/fstabをいじります。homeディレクトリを他のところから読む設定です。  
fugaは他のコンピュータのip、hogeは

fuga:/home/hoge /home/piyo ext4 rw 0 0

あとは再起動すれば、毎回ホームディレクトリが共有されます。  
必ず共有元を先に起動するようにしてください。（実はクラスタやめる時に一寸面倒くさいです）  
通信方式はsshを使うのでssh-agentを使えるようにします。まず、秘密鍵と公開鍵を作ります。[[17]](#footnote-17)

ssh-keygen -t rsa

ssh-addコマンドで鍵を登録することが出来ます。

ssh-add hoge/fuga

下記のコマンドでちゃんと登録できたか確認できます。

ssh-add -l

下記のコマンドでSSH-agentを起動出来ます。

eval `ssh-agent`

また、ssh-agentは-Aオプションつけてやるのがお勧めですが、これはまた別の話。  
sshはネットワークの基礎的な技術なので、本書ではここまでにしておきます。  
詳しい事はインターネット上に分かりやすい記事がたくさんあると思いますので、検索してみて下さい。

あとはipyparallelの機能を用いて並列計算します。ipyparallelをインストールしましょう。

conda install ipyparallel

pipでも良いみたいです。ずっと先でこれの使い方を解説します。

下記の公式サイトの説明の通りにしていけば良いです。  
https://ipyparallel.readthedocs.io/en/latest/

とはいえ、英語の公式サイトを見ながらゴリゴリ設定する…というのもきついものがあります。  
ネット上ではこの記事などが参考になります。  
http://qiita.com/chokkan/items/750cc12fb19314636eb7

何故か僕がやると正攻法で動いてくれなかったので、下記のやり方をとりました。  
http://qiita.com/uesseu/items/def93d1a0e829aec8e86

# maxfilterのインストール(MEG研究の場合)

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なのですが、  
MNEでは特許の問題もあって信頼に足る実装が出来ていません。泣き所の一つです。  
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。  
僕はdocker[[18]](#footnote-18)でcentos5のコンテナをダウンロードしてインストールを試みました。

docker run -it --name centos5 -v ~:/home/hoge centos:5

これでcentos5がダウンロードされ、centos5の端末に入ります。  
ELEKTA社製のソフトは32bit,64bitのソフトが混在しています。  
依存しているものとしては32bitと64bitのfortran、whichコマンドです。  
また、neuromagというユーザーをneuroというグループに入れる必要があります。

yum install compat-libf2c-34.i386  
yum install compat-libf2c-34.x86\_64  
yum install which  
useradd neuromag  
groupadd neuro  
usermod -a neuromag neuro

その上で、DANAとmaxfilterのインストールスクリプトをそれぞれ動かします。

sh install

僕は難しいこと考えるのが嫌だったので、インストールファイルをHDDにコピーして  
スクリプトを動かしました。インストールできたら

/neuro/bin/admin/license\_info

として出力結果をELEKTAに送り、ライセンスを取得します。  
最後に脳磁図計のキャリブレーションファイルを入れる必要があります。  
つまり「人が入っていない時の状態」を入れることになります。  
/neuro/databases/sss/sss\_cal.dat  
/neuro/databases/ctc/ct\_sparse.fif  
この2つが必要です。ライセンスなどは日本法人の人に聞いたほうが良いです。  
細則があります。以上でmaxfilterのインストールは終わりです。

# freesurferを使う(MRI)

ここからターミナルを使っていくことになります。下記は必要最低限の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。

cd DATA

辿っていって、目的のファイルを見つけたならば、freesurfer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。

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。

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、色々な事をしてます。詳しくはfreesurfer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です。[[19]](#footnote-19)  
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スレッド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。  
MNE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…いや、将来性含むとpython一択です。

## freesurfer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)

freeview -v <subj>/mri/orig.mgz \  
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。

## 解析結果の修正の必要性

recon-allが終わった時点で、下記コマンドを入力しましょう。

asegstats2table --subjects hoge1 hoge2 hoge3 ...\  
 --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。  
freesurfer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。  
http://freesurfer.net/fswiki/Tutorials

このfreesurfer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

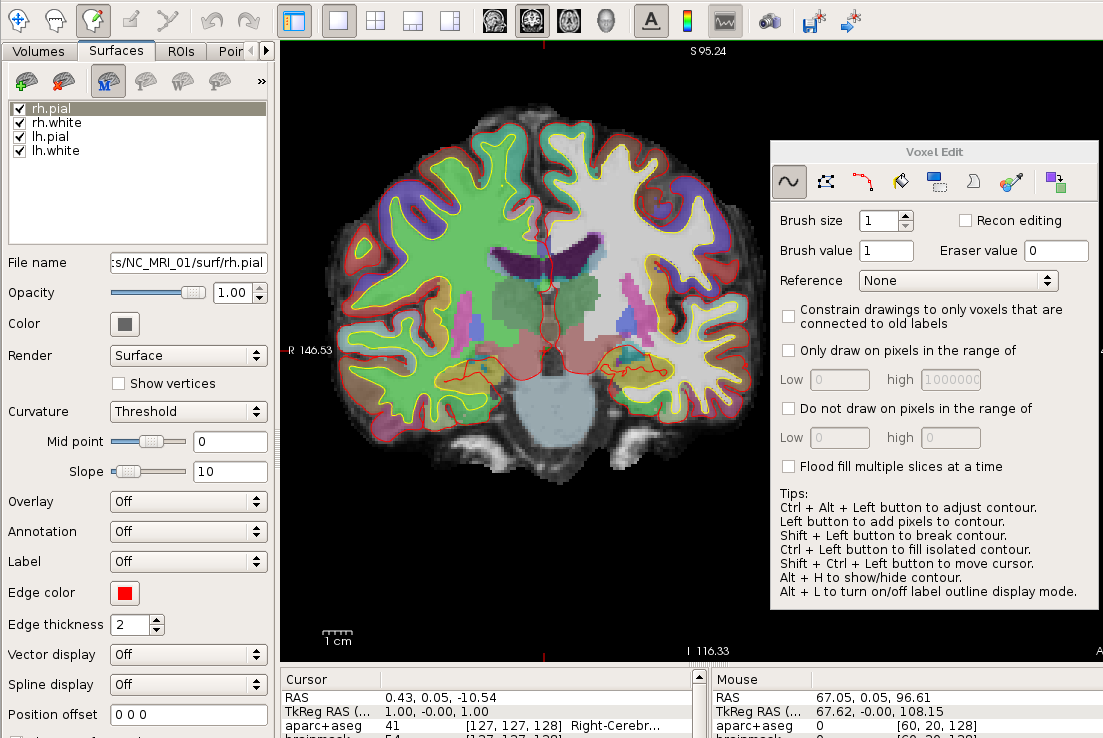
Freesurferは脳だけを解析するためにSkull Stripという作業をします。要するに、頭蓋骨を  
外してしまうわけです。この時にwatershedmethod[[20]](#footnote-20)という方法を使うのですが、頭蓋骨を  
切り取ろうとして脳まで取ったり逆に眼球や脈絡叢まで脳と間違うことがあるので修正が必要です。

### 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

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

とします。



freeviewによる編集

### 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

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

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。

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。

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。

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

# MNE/pythonのインストール(脳波、脳磁図をする場合)

こちらはanacondaの存在下ではかなり簡単です。  
公式サイトをみながら頑張りましょう。  
http://martinos.org/mne/stable/install\_mne\_python.html

要約すれば、下記に列挙するコマンドを順に叩いていけば良いです。  
- conda install scipy matplotlib scikit-learn  
- pip install mne --upgrade  
- conda install mayavi  
- pip install PySurfer  
- conda install pyqt=4

CUDA[[21]](#footnote-21)(GPGPU)についてもそのサイトに記載があります。

この中で特にインストールの鬼門となるのはmayaviです。mayaviは3Dの表示をするソフトで、  
freesurferのデータをMEGとすり合わせる時の必需品となっています。

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。  
また、mayaviは基本、python2系でないと動きません。

export LD\_PRELOAD='/usr/$LIB/libstdc++.so.6'  
export DISPLAY=:0

さらに、jupyter内で下記を実行しないといけません。

%gui qt

# MNE/Cのインストール(脳波、脳磁図をするばあい)

MNECも必要です。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しょう。  
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。  
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。

tar zxvf MNE-hogehoge  
mv MNE-hogehoge MNE-C  
cd MNE-C  
export MNE\_ROOT=/home/fuga/MNE-C  
. $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*

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。  
時に炎上する人だって居ます。廃人になる人も居ます。  
しかし、最先端の科学にとって、SNSは大変有用なのです。  
twitterでMEGやMRIの研究者をフォローしてみてください。  
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。  
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  
フォローしてアンテナはっています。  
ちなみに、若いエンジニアはよくするらしいです。

# freesurferを使う(MRI)

ここからターミナルを使っていくことになります。下記は必要最低限の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。

cd DATA

辿っていって、目的のファイルを見つけたならば、freesurfer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。

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。

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、色々な事をしてます。詳しくはfreesurfer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です。[[22]](#footnote-22)  
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スレッド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。

## freesurfer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)

freeview -v <subj>/mri/orig.mgz \  
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。

## 解析結果の修正の必要性

recon-allが終わった時点で、下記コマンドを入力しましょう。

asegstats2table --subjects hoge1 hoge2 hoge3 ...\  
 --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。  
freesurfer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。  
http://freesurfer.net/fswiki/Tutorials

このfreesurfer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

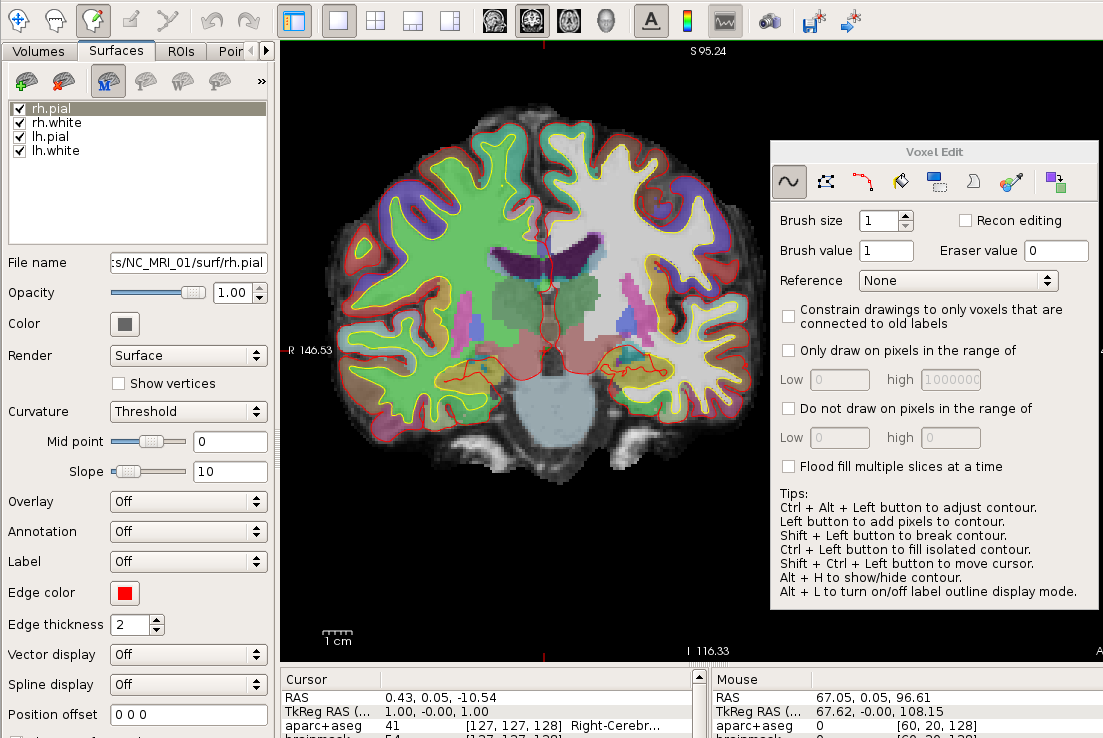
Freesurferは脳だけを解析するためにSkull Stripという作業をします。要するに、頭蓋骨を  
外してしまうわけです。この時にwatershedmethod[[23]](#footnote-23)という方法を使うのですが、頭蓋骨を  
切り取ろうとして脳まで取ったり逆に眼球や脈絡叢まで脳と間違うことがあるので修正が必要です。

### 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

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

とします。



freeviewによる編集

### 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

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

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。

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。

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。

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

# mricronのインストールと変換(MRIを使う場合)

mricronが必要になることもあるので、入れましょう。UBUNTUなら

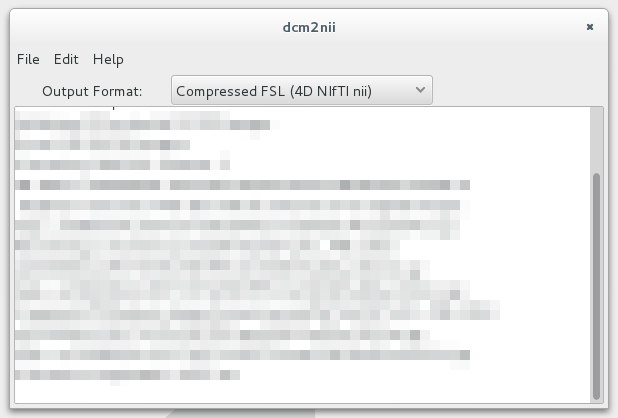
sudo apt install mricron

MACならhttp://www.mccauslandcenter.sc.edu/crnl/mricron/から  
インストーラーをダウンロードします。このmricron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。

以上でfreesurfer/MNE/pythonのインストールは終了しました。  
これでjupyter経由でゴリゴリ計算していくことができます。

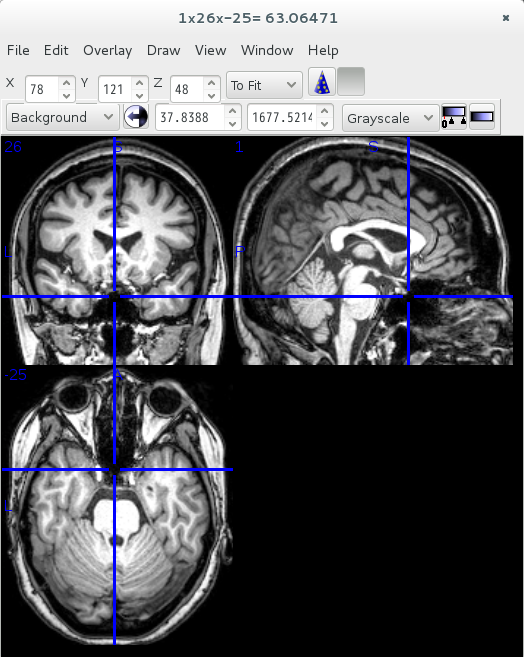
## mricronによるMRIのファイルの変換

mricronはmriの画像の閲覧が出来るソフトですが、  
この中にdcm2niigui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。



dcm2niiの画面

ちなみに、mricron自体はmri閲覧ソフトで、これもこれで有用です。



mricron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。

# MNEを使う

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のスクリプトなのですが…  
かなり流暢なpythonistaが書いていると思われます。  
そのため、初心者が見るには敷居が高目です。  
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。  
http://martinos.org/mne/stable/tutorials.html  
http://martinos.org/mne/stable/auto\_examples/index.html  
http://martinos.org/mne/stable/python\_reference.html

## jupyterの起動

ターミナルで下記を叩いてください

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。

jupyter notebook --ip hoge

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python2を起動しましょう。

## MNEpythonを使う前に学んでおくべきモジュール

とりあえず、pythonとnumpy[[24]](#footnote-24)の基礎を学ばねばなりません。  
これは最低限のことです。これが書けないのであればmne/pythonは無理です。  
PySurferはpythonでfreesurferのデータをプロットするために必須です。

他に学んでおくべきモジュールは  
- matplotlib(作図用。seabornで代用可能なこともある。)  
- pandas(python版excel。必須ではないが、劇的にやりやすくなる。)

pandasやmatplotlibもググってください。qiitaも結構良いです。  
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。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/](http://www.python-izm.com/)

pythonを初めて使う場合は、どうせならjupyterを初めから使って  
慣れておけば良いのではないかと思います。  
素のpythonも悪くはないですが、MNEの公式サイトにはjupyter notebook形式のスクリプトが  
用意されていますから、使わない手はないでしょう。

## 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。  
下記サイトが参考になります。

Pythonの数値計算ライブラリ NumPy入門 http://rest-term.com/archives/2999/](http://rest-term.com/archives/2999/)

import numpy as np  
a=np.array([5,6])  
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。  
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。

print(a+b)

結果

[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の配列は違うものであり、素のpythonならこうなります。

a=[5,6]  
b=[7,8]  
print(a+b)

結果

[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。

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

結果

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。

## データの読み込み…の前にmaxfilter(脳磁図の場合)

maxfilterをかけると数学的に脳の外のノイズを取ることができます。  
これはMNEpythonでも出来はするのですが、今のところまだ成熟していないらしいので、  
エレクタ社のソフトを使うのが適当です。特許の問題もあり、MNEpythonでは  
当面難しいのかもしれません。  
これがないとかなりノイズだらけのデータになりますので、必須です。

## 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、  
微分方程式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。  
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。  
僕はあまり強くない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。

## pythonでの高速化のあれこれ

今後、処理速度が大事になります。特にソースレベル解析ともなると膨大な計算量になります。  
その時のやり方をいくつか記しておきます。  
実用上必要になりますので、後ででも良いので見てみてください。

### for文とリスト内包表記

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…と言われています。  
軽い処理なら良いんじゃないかと個人的には思いますが。

代わりと言ってはアレですが、このようなpython構文があります。

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。

### numpy

numpyは速いので、重い演算の時は使えるなら使いましょう。  
pythonは四則演算とかfor文とかとっても遅いのです。

### 並列化(白魔術、弱)

いつもクラスタ対応な感じで並列化してて、この方法は覚えてないです。ごめんね。  
pyparallelでググってください。

### クラスタレベルの並列化(白魔術、強)

僕は普段からコンピュータが1台でも沢山でも、この方法で並列化しています。  
書き換えがめんどいからです。この方法は元々クラスタ作ってやる方法ですが、  
一人二役(コントローラとエンジン)することで一台でも実現できます。

今回は複数の引数付きでやってみたいと思います。クラスタの作り方については詳しくは述べませんが、  
準備段階については前述していますから参照してください。

その上で、おなじみのアレです。

pip install ipyparallel

クラスタの設定ファイルを作ります。

ipython profile create --parallel --profile=default

この設定ファイルのいじり方は公式サイトとか、qiitaの記事を見てください。

その後のやり方は色々ありますが、僕のやり方を書きます。  
まず、元締めのコンピュータでクラスタを起動します。ターミナルで以下を叩いてください。

ipcontroller --ip=hogehoge --profile default

これでdefaultという名前のクラスタのコントローラをip指定で起動しました。  
…もちろん、1台だけの場合はipは要りません。  
次に、下記のように各計算機(子機？)でエンジンを起動していきます。

ipcluster engines --n=4 --profile=default

--nは使うコア数です。元締めのコンピュータでも計算するなら同じようにしてください。  
これで準備が整いました。

例えば、貴方が下記のような関数を実装したとします。

calc\_source\_evokedpower(id,person,tmax,tmin):  
 d=id+person+tmax+tmin  
 return d

うん、立派な関数ですね。

この関数はグローバル変数を参照できないことに注意してください。  
変数やimport文は全て関数内で宣言するようにしてください。  
宗教的な理由で関数内でimport出来ない方はお引き取りください。

このうち、idとpersonは全ての組み合わせを、tmaxとtminは固定した値を入れたいとします。  
これをクラスタレベルでガン回しします。

import itertools  
from ipyparallel import Client  
client=Client(profile='default')  
print(client.ids)  
  
function=calc\_source\_evokedpower  
  
product=zip(\*list(itertools.product(id,person)))  
plus1=tuple(['20Hz']\*len(arglist[0]))  
plus2=tuple(['50Hz']\*len(arglist[0]))  
  
arglist=product+[plus1]+[plus2]  
  
view=client.load\_balanced\_view()  
async=view.map\_async(function,\*arglist)  
async.wait\_interactive()

まず、ipyparallelをインポートします。一応クライアントを確認しておきます。  
functionに実行したい関数名を入れます。

その後、itertoolsのproduct関数を使ってidとpersonの全ての組み合わせを作ります。  
さらに、固定した20Hzと50Hzを後に加えます。そして、配列を足し算していきます。

その後、3行の呪文を唱えれば出来上がりです。返り値はasync[:]で見れます。

### Cython(黒魔術、使いこなせば相当強い？)

Cpython[[25]](#footnote-25)ではありません。Cythonという別ものです。  
pythonをCに変換することで場合によってはpythonの100倍[[26]](#footnote-26)のスピードを  
実現することが可能です。ただし、型を指定するなど加工しないと超速にはならないため、  
一寸手がかかります。さらに、numpyとかは型関係が難しいです。  
純粋かつ簡単なCで実装できるコードを突っ込むべきでしょう。  
jupyterは大変優秀なので、下記のようにするだけでCythonを実行することが出来ます。

%load\_ext cython

これをjupyterで実行した後、関数を実装します。下記はnumpyの例です。

%%cython -a  
import numpy as np  
cimport numpy as np  
DINT=np.int  
ctypedef np.int\_t DINT\_t  
DDOUBLE=np.double  
ctypedef np.double\_t DDOUBLE\_t  
  
def u(np.ndarray[DDOUBLE\_t,ndim=1] ar):  
 cdef int n  
 cdef double m  
 for n in xrange(5):  
 m=np.mean(ar)  
 print(m)

上5行はCythonとnumpyを組み合わせた時の特有の黒魔術です。  
上では、numpyのためにint型とdouble型を用意してあげています。また、cdefは型指定です。  
関数を宣言するときも黒魔術的にnumpyの型を指定してあげねば  
なりません。じゃないと動くけど遅いままになります。ndimはnumpy配列の次元数です。

それ以外はC言語を書いた人からすると型指定が必要なただのpythonなので、  
苦労はあまりないはずです？ちなみに、元々C言語なのでCythonを普通に使おうとすると  
普通のC以上に面倒くさいコンパイルの手続きが必要になります。

詳しくはcythonのホームページをググってください。  
http://omake.accense.com/static/doc-ja/cython/index.html

### C言語、C++、FORTRAN(最終兵器)

まぁ…そういうやり方もあります。正統派なやり方なのですが、本書では触れません。  
車輪の再発明に気をつけましょう。

## 作図用おまじないセット

下記はjupyterのコマンド

%matplotlib inline  
%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[[27]](#footnote-27)を作りたいなら、inlineを変えてください。  
python3の場合

%matplotlib qt5

python2の場合

%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、こういうのもあります。

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。

## 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  
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)[[28]](#footnote-28)  
- 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cell(パラグラフの事)[[29]](#footnote-29)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の**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書いてありますから、読み替えてください。  
  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、jupyterやspyderやipythonの  
コンソール上でtabキーを押せば中の関数が見えるので、入力自体は楽です。  
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。[[30]](#footnote-30)  
  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。ある周波数以上の波を除去します。  
  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なようです。  
  0.1Hz未満でするのがいいのかもしれません。  
  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。

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

## 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めば良いのではないでしょうか？

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。

raw=mne.io.Raw('hoge',preload=True) #読み込み  
raw.filter(1,100) #0.1~100だけ残すバンドパスフィルタ  
raw.notch\_filter([60,100]) #この場合、60と100Hzを消してる  
raw.resample(sfreq=100) #100Hzにリサンプルする  
raw.save('fuga')

* 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。[[31]](#footnote-31)
* 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。[[32]](#footnote-32)
* 3行目で送電線のハムノイズを取っています。[[33]](#footnote-33)
* 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。
* 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。

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だいたい以下の情報が入っています。

* 波形データ
* チャンネル名と空間データ
* 測定条件

例えば、僕がbiosemi active2という脳波計でやったときは  
出力されたファイル形式はbdf形式でした。  
このbdf形式は広く世界で使われている形式で、日本でよく使われる  
日本光電の脳波計でも新しい機械なら出力できたりします。  
僕がやったときは実験時に64番から65番チャンネルに  
眼球運動が割り当てられており、それぞれの名前は'X1','X2'でした。  
このあたりは脳波計のユーザーが設定できる所なので、  
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)を指定せねばなりませんでした。  
そのため、下記のように書きました。

raw=mne.io.read\_raw\_edf(filename,preload=True,  
 montage='biosemi64',  
 eog=[64,65],exclude=[66,67,68,69])

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、何番目のチャンネルはいらないよ、というやつです。  
  あまりチャンネルが有ることは日常茶飯事です。

### 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。

raw.set\_montage(mont)

### 脳波のセンサーの名前が変則的な場合

で、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](https://mne-tools.github.io/stable/generated/mne.channels.read_montage.html#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法を見てみます。

mont=mne.channels.read\_montage('standard\_1020')  
print(mont.ch\_names)  
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。

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ーく見比べて変えていって下さい。

channel\_list={  
 "EEG Fp1-Ref": "Fp1", "EEG Fp2-Ref": "Fp2",  
 "EEG F3-Ref": "F3", "EEG F4-Ref": "F4",  
 "EEG C3-Ref": "C3", "EEG C4-Ref": "C4",  
 "EEG P3-Ref": "P3", "EEG P4-Ref": "P4",  
 "EEG O1-Ref": "O1", "EEG O2-Ref": "O2",  
 "EEG F7-Ref": "F7", "EEG F8-Ref": "F8",  
 "EEG T3-Ref": "T3", "EEG T4-Ref": "T4",  
 "EEG T5-Ref": "T5", "EEG T6-Ref": "T6",  
 "EEG Fz-Ref": "Fz", "EEG Cz-Ref": "Cz",  
 "EEG Pz-Ref": "Pz","EEG A1-Ref":"A1",  
 "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。  
僕が読み込んだ時は上記のように変えなきゃなりませんでした。

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。

### 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。

raw = mne.set\_eeg\_reference(  
 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。

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  
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。

mne.find\_events(raw)

rawの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

import pandas as pd  
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

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形式の数列です。  
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  
jupyterで%matplotlib qtとしたあとでraw.plot()でデータを見ながらひたすら下記のように  
badchannelを設定していってください。それだけです。

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、そういうやつを選んでください。  
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。

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。

## ICAをかけよう

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、  
何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が脳波、脳磁図に混じることがあるので、これを除去するのです。  
これはICAという方法(独立成分分析)で波を幾つかの波に分け、  
その上で眼球運動や心電図っぽい波を除去するフィルタを作ります。

順を追って内容を説明します。

from mne.preprocessing import ICA  
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。

picks\_meg = mne.pick\_types(raw.info, meg=True, eeg=False, eog=False,  
 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。

n\_components = 25   
method = 'fastica'   
decim = 4   
random\_state = 9

n\_componentsはICAが作る波の数です。  
ICAで作る波の数は何個が良いのか僕にはよく分かりません。  
多分現時点で決まりはないと思うので、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。

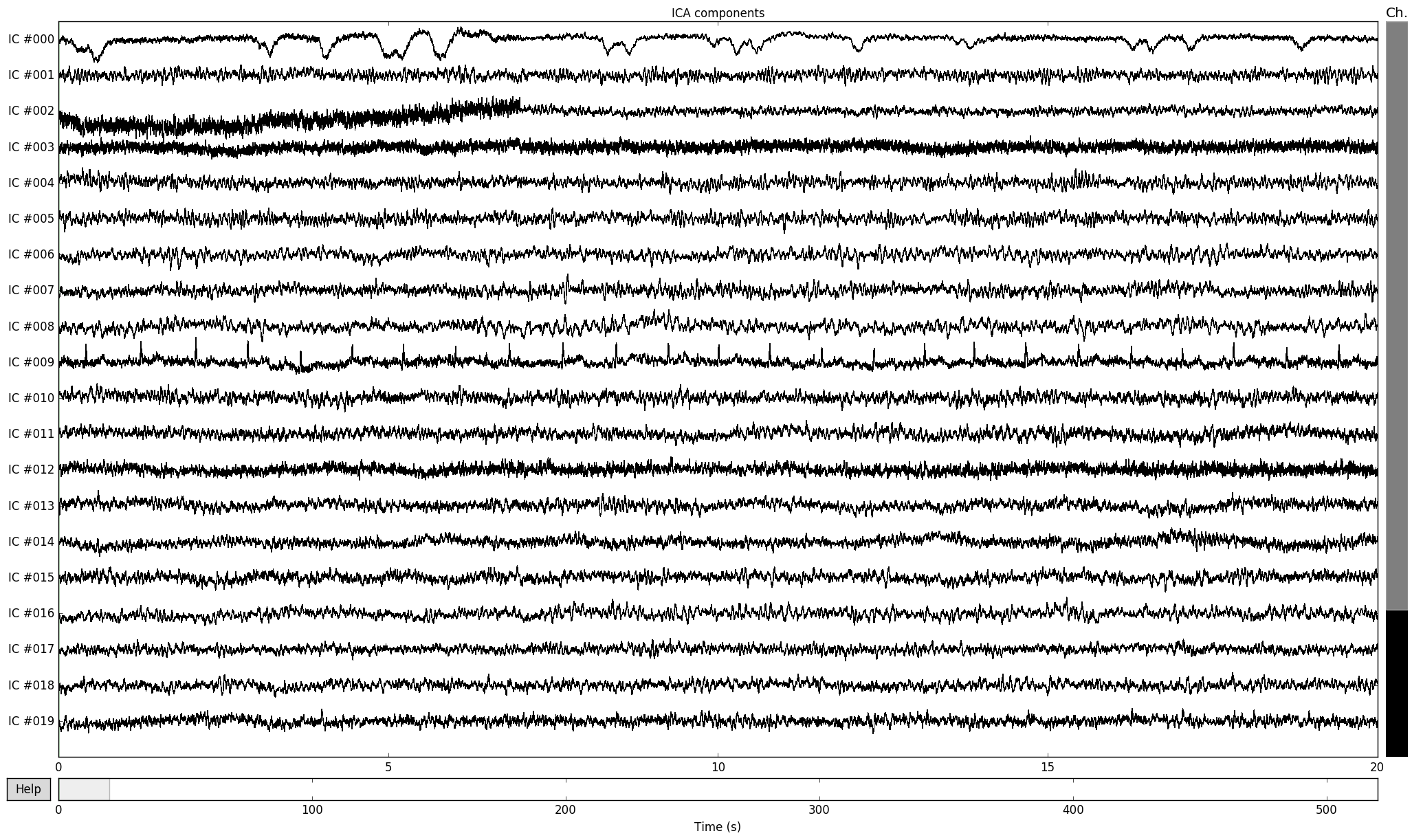
decimはどの程度詳しくICAをかけるかの値です。  
数字が大きくなるほど沢山かけますが、数字を入力しなければ最大限にかけます。

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。  
便利ですね！

ica = ICA(n\_components=n\_components,  
 method=method, random\_state=random\_state)  
ica.fit(raw, picks=picks\_meg, decim=decim,reject = dict( grad=4000e-13))

icaのセットを作り、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。  
先に%matplotlib qtと入力した上で下記を実行してください。

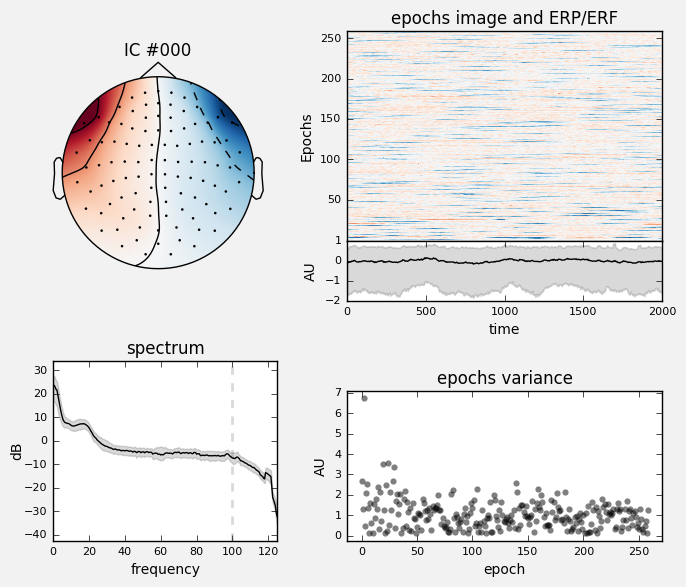
ica.plot\_sources(raw)



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。

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。

ica.plot\_properties(raw, picks=0)



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。

filtered\_raw=ica.apply(raw,exclude=[0,10])

これでicaはかけ終わりです。  
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。

## EpochとEvoked

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ものです。  
epochは元データをぶつ切りにしたものです。  
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きます。  
evokedは切り出したものを加算平均したものです。

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。

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。

221 events found  
Events id: [1 2 4 7 8]  
Out[205]:  
array([[ 15628, 0, 2],  
 [ 18053, 0, 2],  
 [ 20666, 0, 4],  
 [ 23131, 0, 1],  
 [ 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。

epochs=mne.Epochs(raw,event\_id=[1],events=events)

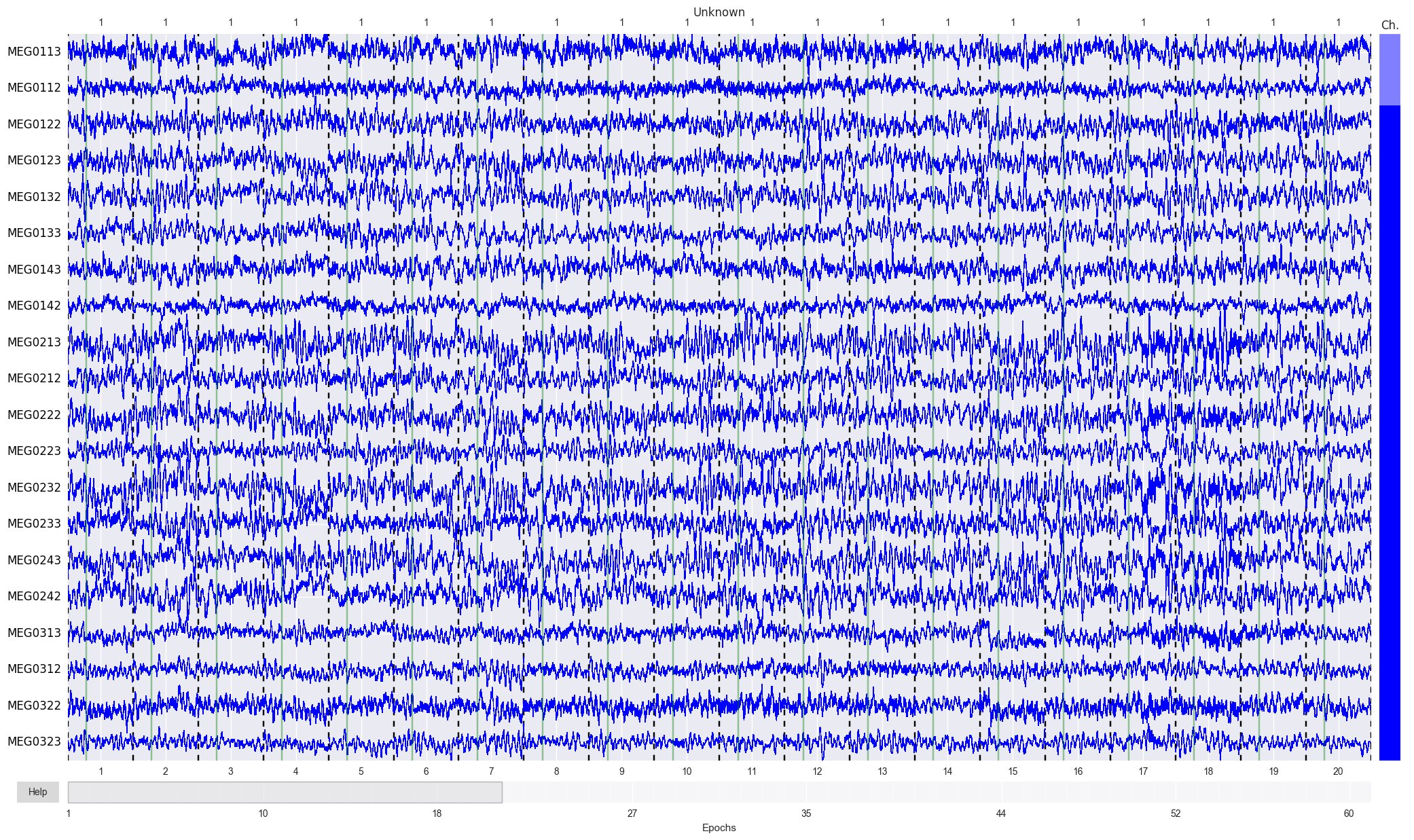
先程のeventsを使っています。  
event\_idは配列にしてください。ここは[1,2]とかも出来るのでしょう。  
evokedを作るのはとても簡単で、下記のとおりです。

evoked=epochs.average()

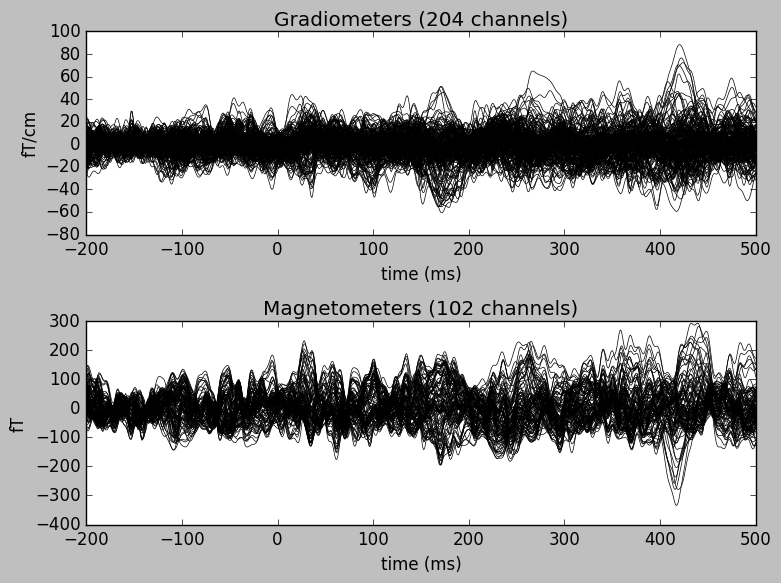
## データのplot、主にjupyter周り、そしてPySurfer

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。

epochs.plot()  
evoked.plot()



epochsの例



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ではどのように表示するかを選ぶことが出来ます。

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。

%matolotlib 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。

%matplotlib qt

また、3D画像を表示したい場合は

%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。

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。

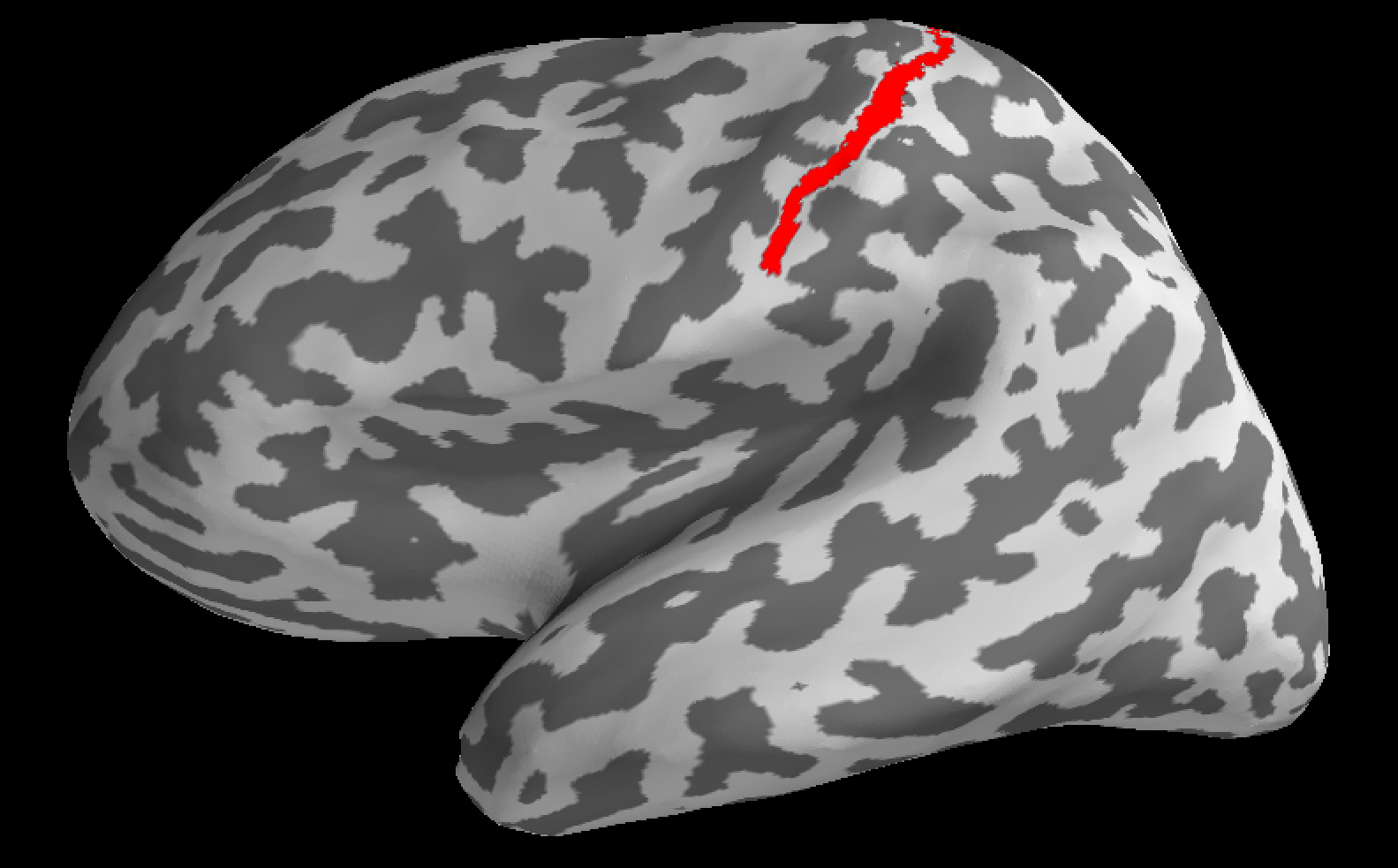
また、PySurferについては例えば下記のような感じです。  
これはmacの場合ですが、ubuntuも同じ感じです。  
subjectやsubjects\_dirはfreesurferの設定で読み替えてください。  
jupyterで下記の呪文を唱えましょう。

import surfer  
%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。

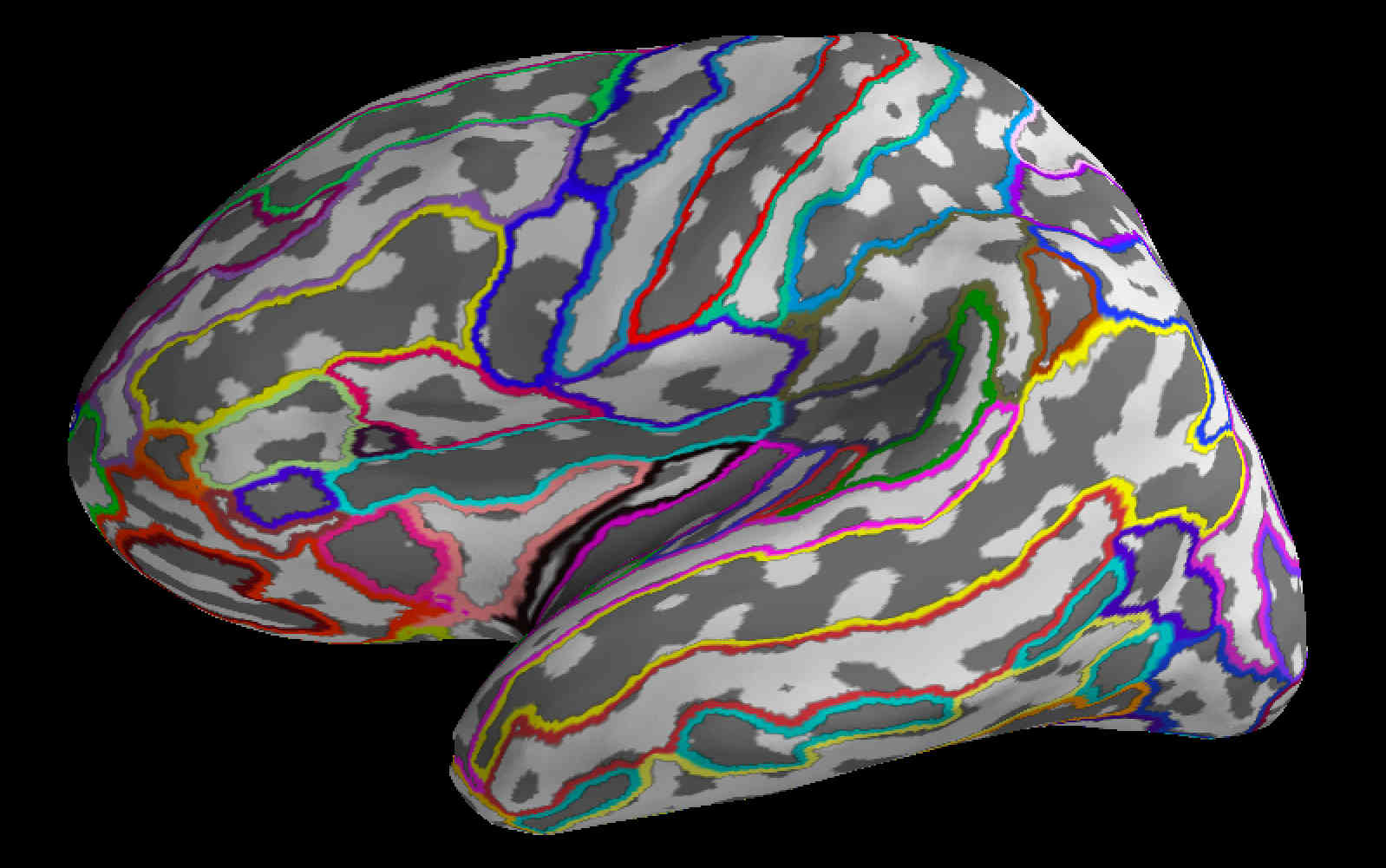
brain = surfer.Brain(subject, "lh", "inflated",  
subjects\_dir=subjects\_dir)  
brain.add\_label("BA1.thresh", color="red")

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。



pysurfer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。



pysurferで表示したfreesurferのannotationファイル

brain = surfer.Brain(subject, "lh", "inflated",  
subjects\_dir=subjects\_dir)  
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ぁと思いました。

## 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。

channel=['Fz','FCz','FC1','FC2','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。

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。

data=[]  
for ch in channel:  
 data.append(evoked.data[evoked.info['ch\_names'].index(ch)])

ちなみに、下記のように書くのがよりpythonicと思われます。  
これはリスト内包表記と言って、pythonistaが好んで使う方法です。

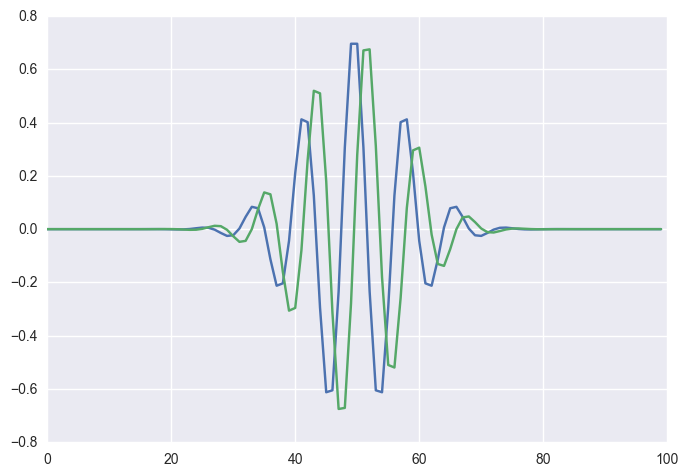
data=[evoked.data[evoked.info['ch\_names'].index(ch)] for ch in channel]

## センサーレベルwavelet変換

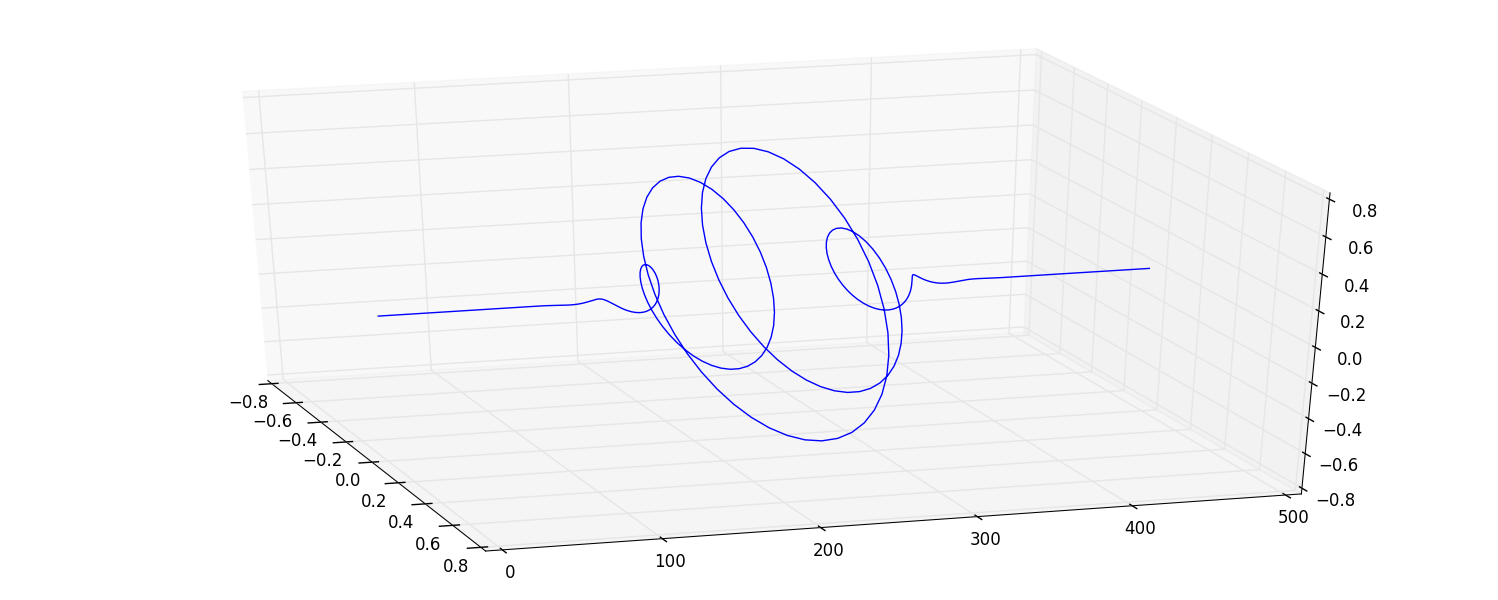
これは解析のゴールの一つと言えましょう。

### そもそもwavelet変換とは何なのか

特定の周波数の波の強さや位相を定量化するための計算方法です。  
僕は数学が苦手なので、適当な説明です。フーリエ変換という言葉をご存知でしょうか？  
これは波をsin波の複合体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。  
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)  
そこで、減衰するwaveletという波を使って波を表す方法を使います。  
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。  
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。



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。



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

wavelet変換に登場する単語としては以下のものが挙げられます。

|  |  |  |
| --- | --- | --- |
| 単語 | 内容 | 特徴 |
| evoked power | 波を加算平均した後にwavelet変換、波の強さ | ノイズにやや強い |
| induced power | waveletした後に結果加算平均、波の強さ | ノイズに弱いが後期成分に強い |
| phase locking factor | 同一部位での位相同期性 | ノイズにやや強い |
| phase locking value | 色々な部位同士の同期性 | 後期成分に強い |

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。  
それぞれ生理学的には違うものを見ているらしいです。  
これらの特徴の違いが何故生まれるかについても後半の  
「初心者のための波形解析」を御覧ください。

ではevoked power,induced power,phase locking factorについて  
解析を行いましょう。

### 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。  
下記のスクリプトで実行できます。

freqs=np.arange(30,100,1)  
n\_cycles = 6  
evoked\_power=mne.time\_frequency.tfr\_morlet(evoked,n\_jobs=4,  
 freqs=freqs,n\_cycles=n\_cycles, use\_fft=True,  
 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かでも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。

|  |  |  |  |
| --- | --- | --- | --- |
| return\_itc | 引数 | 返り値1つ目 | 返り値2つ目 |
| False | evoked | evoked\_power | なし |
| False | epochs | induced\_power | なし |
| True | epochs | induced\_power | phaselocking\_factor |

itcを計算したい時は返り値が2つになりますから、下記のです。

freqs=np.arange(30,100,1)  
n\_cycles = 6  
induced\_power,plf=mne.time\_frequency.tfr\_morlet(epochs,n\_jobs=4,  
 freqs=freqs,n\_cycles=n\_cycles, use\_fft=True,  
 return\_itc=False, decim=1)

## 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。  
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

evoked.data.shape

とすればデータの構造が確認できます。

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。)

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ってか一寸複雑です。

|  |  |  |
| --- | --- | --- |
| 形式 | 読み込み関数 | 備考 |
| raw | mne.io.Raw() | 脳磁図の場合。脳波とかは公式サイトAPI参照 |
| epochs | mne.read\_epochs() |  |
| evoked | mne.read\_evoked() | 条件によって配列で返されることあり |
| itc | mne.time\_frequency.read\_tfrs() | 条件によって配列で返されることあり |
| power | mne.time\_frequency.read\_tfrs() | 条件によって配列で返されることあり |

例えば

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  
条件によって配列で返されることがある関数だからです。この場合は行列として返されます。  
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の中の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、

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

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×サンプリング周波数/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

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。

3と4はjupyterならシームレスに扱うことが出来ます。これは超楽なので僕のオススメのやり方です。

## Rとpadasの連携、特にANOVAについて

「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

%load\_ext rpy2.ipython

とした後

%%R -i input -o output  
hogehoge

という風に記述すればhogehogeがRとして動きます。  
データの受け渡しにはpandasを使うのが良いです。

import pandas as pd  
data=pd.Dataframe([二次元配列])

%%R -i data  
print(summary(data))

さて…これを応用します。  
前述のnp.mean()関数で特定の時間、周波数など切り出した数値(配列ではない)があります。  
この数値を仮にnumという変数に入れるとします。これに背景情報を付けます。データの背景情報に  
「疾患群、健常者群」「右脳、左脳」「刺激提示、プラセボ」という分類を作ったとしましょう。  
…日本語は色々と面倒なので、下記のような分類に変えます。  
['disease','normal'],['right','left'],['stimuli','placebo']

そして、上記で出したデータが  
['disease','left','stimuli']という背景情報に合致するのであれば、  
次のような配列を作ります。

['disease','left','stimuli',num]

この配列をさらに大きな配列に入れていきます。

data=[['disease','left','stimuli',num]]

仮に、次のデータが  
['disease','left','placebo',num2]なら、

data.append(['disease','left','placebo',num2])

とすれば追加されます。[[34]](#footnote-34)  
さて…これで被験者の背景情報まで含まれた2次元配列が出来ました。  
これをpandasを使ってRのDataFrameとほぼ同等のものにします。

import pandas as pd  
df=pd.DataFrame(data,  
 columns=('group','hemisphere','test','value'))

これで、横軸にcolumsのラベルの付いたデータフレームが出来ます。  
jupyterのRではこれを読み込めます。具体的には下記のようにします。

%%R -i df  
print(summary(aov(df$value~df$group\*df$hemisphere\*df$test,data=df)))

ここでpythonからいきなりRを書き始めます。  
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？」  
と査読者からrejectを食らう可能性もありますから辞めましょう。  
今回は多重比較です。多重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、print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。

## 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。

MNEpython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

このうち、multitaperとfourierは離散の計算方法、  
morlet waveletは連続wavelet変換です。(今までやってたのと同じ)

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
* Imaginary
* Phase-Locking Value
* Phase Lag Index
* Weighted Phase Lag Index

…多すぎですね(´・ω・｀)  
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？  
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[[35]](#footnote-35)ので、  
pick\_channelやdrop\_channelで要らないのを外していきます。

epochs.pick\_channels(['hoge'])  
epochs.drop\_channels(['fuga'])

では、始めましょう。

from mne.connectivity import spectral\_connectivity as sc  
cons = sc(epochs, method=’coh’, indices=None,  
 sfreq=500, mode=’multitaper’, fmin=35, fmax=45, fskip=0,  
 faverage=False, tmin=0, tmax=0.5, mt\_bandwidth=None,  
 mt\_adaptive=False, mt\_low\_bias=True,  
 cwt\_frequencies=None, cwt\_n\_cycles=7,  
 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、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ですね。上記の通り。
* indices: どことどこのconnectivityを見たいかです。
* sfreq: サンプリング周波数です。
* 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。  
この関数には5つの返り値と、実質4つのモードがあると考えるとやりやすいと思います。  
それぞれについて解説します。

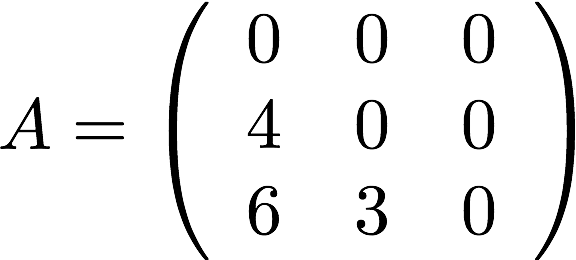
### 5つの返り値

さっきの関数には5つの返り値があります。順に

* con: connectivity numpy形式  
  幾つかパターンあるのであとで書きます
* freqs: 周波数  
  何故ここで周波数が出てきたか一瞬怪訝に思いそうですが、  
  これはfourierとmultitaperモードの時に力を発揮します。  
  というのは、離散的な演算をするので、**周波数は関数が勝手に設定する**からです。  
  morlet waveletの場合はcwt\_frewuenciesで設定した物が出てきます。
* times: 解析に使った時刻のリスト
* n\_epochs: 解析に使ったepochの数
* n\_tapers: multitaperの時だけnullとして出力  
  DPSSという値が格納されます。

上記のコードではconsという変数にタプルを入れているので、  
cons[0]がcon、cons[2]がtimesです。

この中で大事なのはconです。何故なら、これが結果だからです。  
conの中で一番大事なのは中に入っている三角行列です。  
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。



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。

conmat=np.mean(con,axis=(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。

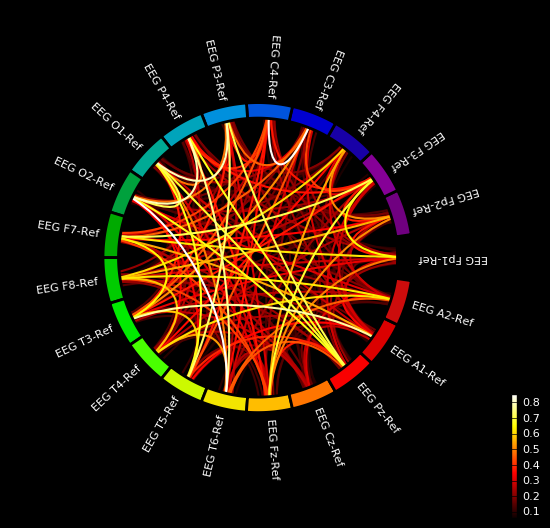
conmat=np.mean(con,axis=(2,3))

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。  
三角行列がある場合は綺麗なplotが出来ます。下記のとおりです。

mne.viz.plot\_connectivity\_circle(conmat, epochs.ch\_names)



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。

indices = (np.array([0, 0, 0]),  
 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 X 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 X 周波数 X 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。  
難しいのです。頑張りましょう。

ソースレベル解析については冒頭の記述を見ていただくとして、  
MRIとMEGをくっつけていきます。  
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。  
手順としては以下のとおりです。

1. MRIから脳の形を取ってきて計算できる形にする。  
   これをBEMという。
2. 脳の形から「推定するべき脳の位置」を特定する。  
   この脳内の位置情報をソーススペース(source space)という。
3. 「推定するべき脳の部位」とMEGで取れる頭の形情報を重ねる。  
   この作業は手動で行われる。(やればわかる)  
   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。
4. 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。
5. ノイズについて考慮する。この時、covariance matrixと言うものが必要になる。
6. 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逆問題(inverse solution)という。  
   逆問題を解くために数式を作る。その数式をinverse operatorという。  
   逆問題には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。
7. 脳全体で推定した波形のうち、欲しいものをとってくる。

その後は色々なストーリーがあるでしょう。

* 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1、BEM作成

上記の通り、MRIから抽出してくる形データとして、BEMと言うものを使います。  
BEMは脳の全体を包み込むサランラップみたいなデータです。  
頭蓋骨とか皮とか、そういうのいろいろ考慮するために、BEMは三枚一組で  
出力されます。実装上はr3枚あるということを意識しなくても大丈夫です。

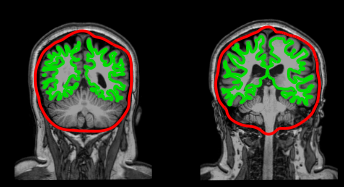
作るためにはfreesurferによる解析データが必要となります。  
ターミナルに戻って下記コマンドを入力しましょう。しばらくかかります。  
freesurferを既に使っているならSubject関連は既に馴染んだ言葉でしょうか？  
もちろんSUBJECTやSUBJECTS\_DIRは読み替えてください。

mne watershed\_bem -s subject -d subjects\_dir

これにより、BEM(脳の形情報を変換したやつ)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。

mne.viz.plot\_bem(subject = subject, subjects\_dir = subjects\_dir,  
 brain\_surfaces = 'white', orientation = 'coronal')

これでBEMが表示されるはずです。



BEMの図示。

## 手順2、ソーススペース作成

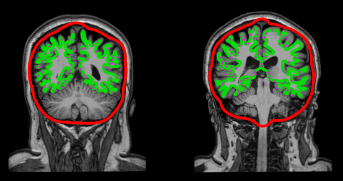
脳磁図で見れる空間のうち、どの部分の電源を推定するかを  
設定する必要があります。その設定がソーススペースです。

src = mne.setup\_source\_space(subject = subject, spacing = 'oct6',  
 subjects\_dir = subjects\_dir)

これで、srcという変数にソーススペースが入りました。

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。



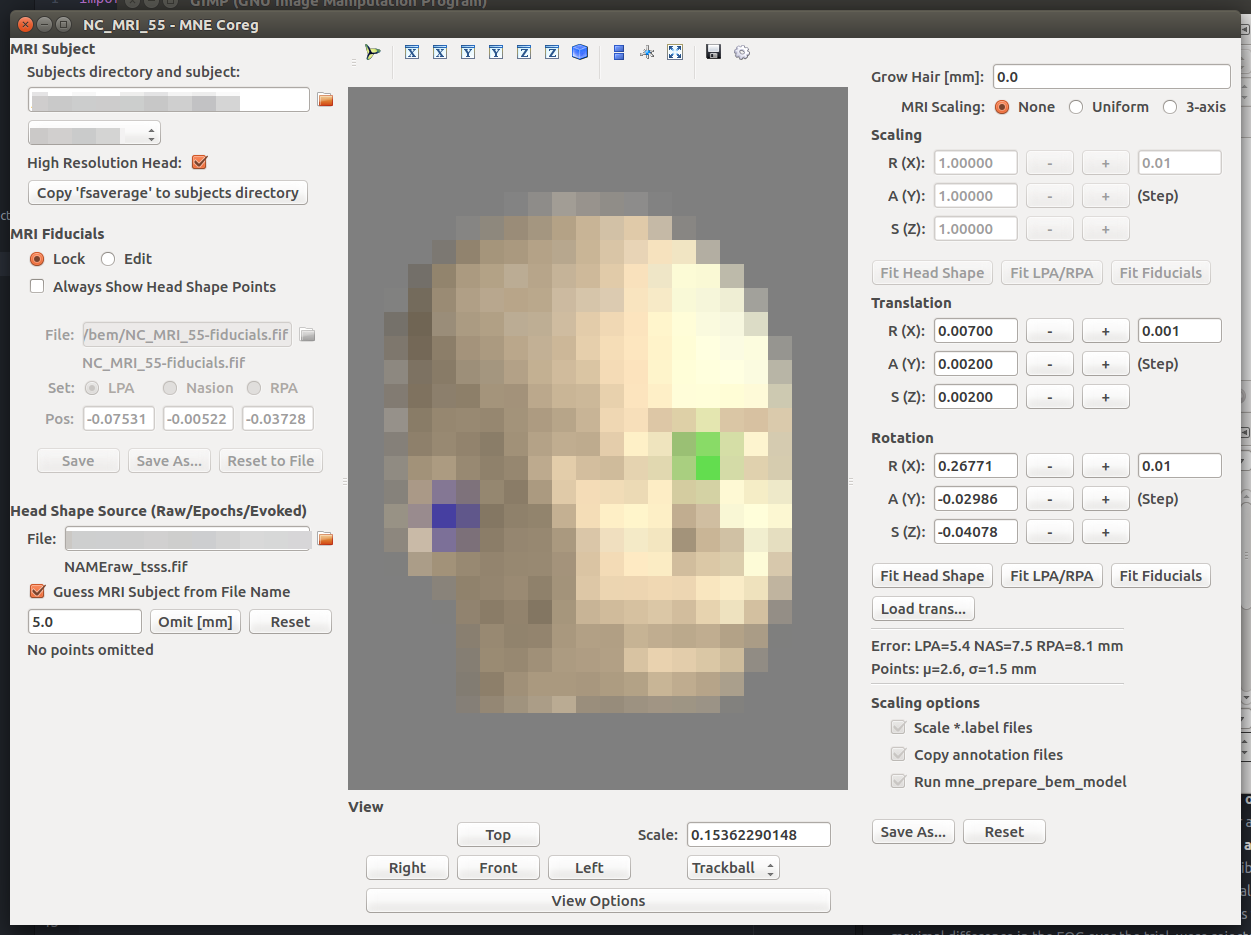
ソーススペースの図示。小さい点々がソーススペース。

## 手順3、trans

GUIでの操作となります。今回、freesurferのsubject,subjects\_dirが必要なのと、  
MEGのデータへのpathが必要です。フルパスです。下記のコードを実行すると画面が立ち上がります。

mne.gui.coregistration(subject = subject,subjects\_dir = subjects\_dir,  
 inst = file\_path)

instは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。  
コードを実行して立ち上げた場合、既にデータが読み込まれているので、楽です。  
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。スクリプト書きましょう。



mne coregistrationの画面。大して苦行ではない。

手順はこうです。

1. 左側、setのところで耳と眉間の位置を入力(MEGのスタイラスでポチるところです)
2. それの一寸上の所、lockをポチる。
3. 右側、Fit LPA/RPAボタンを押す。
4. 中の人の顔データ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。
5. 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

trans = mne.read\_trans('/Users/hoge/fuga/trans.fif')

みたいな感じで読み込んで使います。

## 手順4、順問題

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。  
そもそもEEGのソースベース自体どうかとも僕は思っています。

では、BEMで順問題を解く準備をしましょう。

conductivity = (0.3,)  
model = mne.make\_bem\_model(subject = 'sample', ico = 4,  
 conductivity = conductivity,  
 subjects\_dir = subjects\_dir)  
bem = mne.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)とかになったりします。

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。

trans = mne.read\_trans('/hoge/fuga')  
mindist = 5  
fwd = mne.make\_forward\_solution(raw.info,  
 trans = trans,  
 src = src, bem = bem,meg = True,  
 eeg = False,  
 mindist = mindist, 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。  
mindistは頭蓋骨からみて、一番浅い部分にあるソーススペースの距離です。

## 手順5、コヴァリアンスマトリックス関連

次に、MAP推定という方法を用いて脳活動を推定します。  
この推定にはcovariance matrixというものを使ってソースベースのデータの  
ノイズ周囲の事を計算していかねばなりません。  
これにはMEGを空撮りした空データを使います。下記でからの部屋データを読み込みます。

cov = mne.compute\_raw\_covariance(raw\_empty\_room,  
 tmin = 0, tmax = None)

これでコヴァリアンスを作ることになりますが…MNEには更に追加の方法があります。  
上記の空室の方法は広く行われている方法ですが、  
誘発電位を見たい場合はrestingstate(脳が何もしていない時の活動電位)  
がノイズ(本来ノイズではないが、ここではノイズ)として乗る可能性があります。  
それを含めるならば、下記のようにすることも出来ます。

cov = mne.compute\_covariance(epochs, tmax = 0., 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ですると結構重いです。

他に、rawデータからcovariance matrixを作る方法もあります。

compute\_raw\_covariance(raw, tmin = 0, tmax = 20)

これはresting stateとかに良さそうですね？  
…最早当然ですが、tmin,tmaxは時間です。単位は秒です。

## 手順6、逆問題

最終段階です。順問題とコヴァリアンスを組み合わせて逆問題を解きましょう。  
下記のとおりです。

inverse\_operator = make\_inverse\_operator(epochs.info, fwd,  
 cov,loose = 0.2, 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を入れていますが、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！

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。

from mne.minimum\_norm import apply\_inverse  
source = apply\_inverse(evoked, inverse\_operator)

これで出てきたものの中に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いと思います。  
…とは言え、推定ではあるので本物かどうかはわかりませんが。  
細かい所は公式サイト見てください。

さて…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。  
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。

ls $SUBJECT\_DIR

freesurferのサブジェクトが沢山出てくるはずです。  
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。

mne.read\_labels\_from\_annot(subject,annot\_fname = 'hoge')

詳しくは公式サイト(ry  
他にも読み方があります。  
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。

mne.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。

### 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。

source\_label = mne.extract\_label\_time\_course(stcs,  
 labels, src, mode = 'mean\_flip')

ここではstc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  
mean\_flip: すみません、わかりません＞＜  
pca\_flip: すみません、わかりません＞＜  
max: ラベルの中で最大の信号が出てきます

殆どわからなくてごめんなさい。  
ただ、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。

### その後のお楽しみ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phaselocking\_factorを算出する関数は下記です。  
※激重注意、要WS！[[36]](#footnote-36)

induced\_power, itc = source\_induced\_power(  
 epochs, inverse\_operator, frequencies, label,  
 baseline = (-0.1, 0),  
 baseline\_mode = 'zscore',  
 n\_cycles = n\_cycles, 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。  
まず、ベースラインコレクションはここではzscoreでしています。  
やり方は色々あります。labelはfreesurferのラベルデータです。  
baseline補正の時間についてはデータの端っこすぎると値がブレるので、  
そこのところはデータ開始時点〜刺激提示の瞬間の間で適切な値にしておいてください。  
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。

### その後のお楽しみ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。

con, freqs, times, n\_epochs, n\_tapers = spectral\_connectivity(  
 source\_label, method = 'coh', mode = 'multitaper',  
 sfreq = 500, fmin = 30,  
 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！*

皆さんもこのような科学系同人誌書きたいですよね？  
書いてコミケにサークル参加したいですよね？  
\*\*難しいLaTexなんて覚えなくても大丈夫。そう、markdownならね！\*\*  
LaTeXは添えるだけ。手順は下記。  
  
macならmactexとpandocをインストールします。  
ubuntuやwindowsならTeXliveをインストールします。  
mactexはググれば出てきます。pandocは  
brew install pandoc  
  
ubuntuなら  
sudo apt install texlive-lang-japanese  
sudo apt install texlive-xetex  
sudo apt install pandoc  
  
これでpandocでmarkdownからpdfに変換できるようになります。  
例えばDoujinshi.mdというマークダウンファイルを作って  
  
pandoc Doujinshi.md -o out.pdf \  
-V documentclass = ltjarticle --toc --latex-engine = lualatex\  
-V geometry:margin=1in -f markdown+hard\_line\_breaks --listings  
  
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、  
コードの上の```の末尾に{frame=single}と書き加えてください。  
  
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。  
良い同人ライフを！

# 初心者のための波形解析

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(地獄級)でギリギリやれますし工学部生は大抵理解しています(?)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。  
…というか、算数が苦手なので不正確な話しかできません。

## 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得たいでしょうか？  
脳波や脳磁図の特定の波長…例えばα波、β波、γ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かとかも見たいです。  
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と凹みを  
両方同じように評価するにはどうすればいいでしょう？  
その方法の一つがcos波とsin波を両方使って波を定量化するフーリエ変換です。

しかし、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きついです。  
実は完全にダメではなく、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す…でもやっぱり無理気味なのが難点です。[[37]](#footnote-37)

wavelet変換をするのですが、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。

## 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。

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…

* :元の波を表す式
* :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

かなり複雑な式っぽいですが、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、上の式を数学的に変換して公式を求めます。

そして、その計算の為には複素数(!)を用います。

## 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。

この式です。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。高校数学(地獄級)でギリギリ出てきます。下記です。

何度見ても美しい公式ですね。[[38]](#footnote-38)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ので、  
フーリエ級数の式からsinとcosを排除できます。高校数学(地獄級)で弄くり倒すと、

と変形できます。Cn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。

は元の波、はオイラーの公式を見ると…極座標で言うと半径1の円ですね。

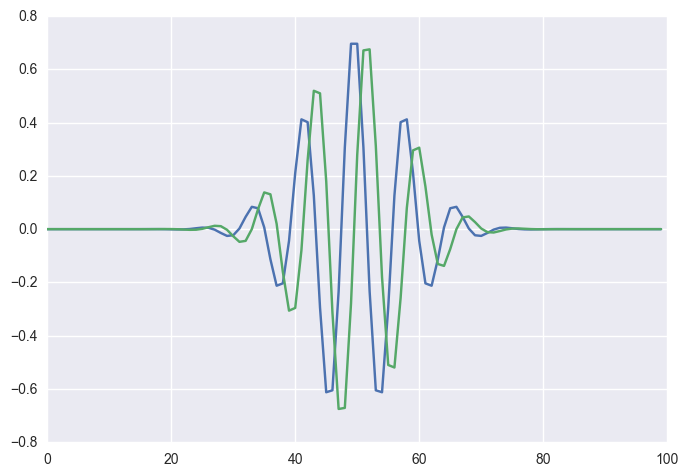
ここで知りたいのはCnです。  
Cnを求めるにはやはり高校数学(地獄級)によって下記のように書き換えられます。

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。

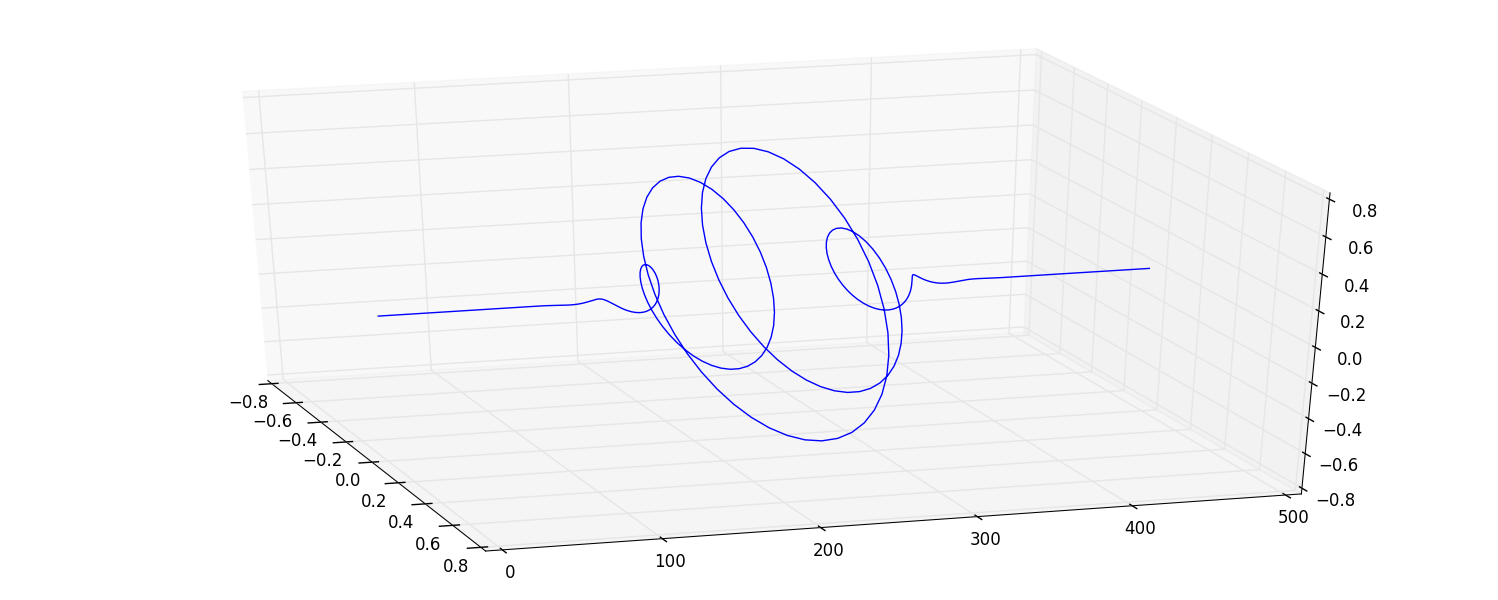
i乗とか実際に計算するのはアレなので、実際の計算をする時も  
オイラーの公式で展開…と言いたいところなのですが、  
python3なんかは複素数の演算ができるので、を計算できたりします。凄いですね！！！

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。  
既にそれをするスクリプトが開発されているからです。[[39]](#footnote-39)

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。



再掲。morlet waveletの2Dプロット



再掲。3Dプロット

## 結局何をしているのか

要するに基準となる波(エネルギー１)を物差しにして、

という感じの計算です。定数さえわかれば波が表せます。定数を表す式は上記のとおりです。  
で、両方同時に掛けるうまい方法がオイラーの公式の複素数平面なわけです。

## 臨床的な数値を算出する

こうして出て来る数値(フーリエなら係数に相当するやつ)はスペクトルと呼ばれるものです。  
実はこの時点で波の力と位相は既に計算できています。  
の絶対値の二乗が力(パワースペクトル)、そして位相がそのまま位相です。  
何故二乗？と思われるかもですが、振幅の二乗に比例しますし、  
それほど不自然ではないと僕は思うのです。

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)です。

なにしろ、の絶対値が1なのは自明なので、  
掛け算しても理屈上は振幅もエネルギーも絶対値は変わらないわけです。  
そして、位相については物差しになる波との位相の差が出てきます。

## そしてwavelet変換へ

話を戻します。この時点ではフーリエ変換です。waveletにするためには減衰する波の必要があります。  
幸い、全ての実数は二乗すると正になりますから、こんな感じでいいです。

こんな感じの式に色々と定数を付けたのがmorlet waveletです。  
何故定数をつけるかというと、減衰する波である以上、補正しないと上記のフーリエ変換のように  
エネルギーを測れないからです。複素数平面で回りながら半径を小さくしていくものです。  
3Dプロットのとおりですね。ちなみに、これは他にも色々あるやり方の一つです。

morlet waveletも計算結果が複素数平面として出てきますから、  
結果の絶対値を二乗したら力、結果を絶対値で割れば位相が定量化出来ます。

この複素数平面上で定量化した位相については、各波ごとに加算平均していきます。  
同じ位相であれば複素数平面上のベクトルを同じ方向に足し合わせていく  
事になりますから「位相の合計の絶対値」が大きくなっていきます。  
バラバラなら絶対値は0に近づくはずです。  
つまり、各波の位相のシンクロ率(位相同期性)は  
複素数平面上のベクトルの足し算によって表現されるのです。

この2つを我々の領域ではそれぞれpower、phaselocking\_factorと呼びます。

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。

## *コラム4-ニューラルネットワーク*

「脳神経をエミュレートする仮説に基づいた人工知能」の事を  
ニューラルネットワークと言います。pythonの必要性に迫られ、  
pythonと数学を学んで使いこなせたなら、十分理解できる分野です。  
そして、踏み込むべきだと思っております。  
何故なら、僕は人工知能は人間のニューロンを  
エミュレート出来ていると信じており、  
この仮説を理解することが脳科学への理解を深める  
搦手になると信じているからです

# Minimum-norm-estimationの理屈

かなり面倒いのと自身がないのと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。  
超簡単に言いますと、一般化逆行列を使って割り算した上で、  
ノイズについてベイズ推定するというものです。

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。  
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とし、ソースで発生したのをとしましょう。  
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。

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、下記のように表すとします。

ここでを下記を満たす行列とします。

この連立方程式を解きます。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。  
一時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。  
ですが、最も真実に近いっぽいのを推定することは出来ます。  
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。  
そのやり方をMinimum-norm-estimationと言います。  
(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)

つまり、条件のもとでを小さくしたいのです。  
(ノルムが小さい。中学生風に言うと絶対値)  
が小さいってことは下記の式が小さいってことです。

まず、ラグランジュの未定乗数法という公式を使います。  
普通の最小二乗法ではいけません。理由は少ない方程式から沢山の解を  
得ようとすると、最小二乗法では解が定まらないからです。  
が極値になるx,yは

の時に

の解、または

の解。…という感じの公式です。  
行列の微分をしないといけないので、行列の微分の仕方を確認しておきます。

これにより、

という式をとでそれぞれ微分して下記のようになります。

では変形していきましょう。

これで無事をとで表せました。

では、次に重み付けをしてみましょう。

# MNEの重み付け

さて、MNEは重み付けを行うことが出来ます。  
これはMEGの場合は特に大事です。何故なら、MEGは計測の  
方法によっては脳の表面の皮質の信号しか捉えられないからです。  
どの値を重視して推定していくかが設定できるわけです。  
では、重みを付けてみましょう。  
さっきは

を最小にするようなものでしたが、 これは重みが入っていません。  
重みの行列を仮にとして、その上で

が最小になるような条件を設定してあげればいいです。  
このは縦と横の長さが同じ正方行列です。  
なので、こいつはほぼほぼ逆行列を作ることが出来るはずです。  
さらに、下記も正方行列になるので逆行列が作れます。

ではそのことを踏まえて変形していきます。

凄いですね！これやこの、L1ノルムのMNEの式です。  
でも、実はこれだけじゃ不十分です…。

# L2ノルムMNEとベイズ推定

今回はラグランジュの未定乗数法で二乗したのがxだけでしたが、  
実際はも二乗して計算します。(L2ノルムといいます)  
そうしないと(説明は難しいですが)良い計算結果が出ないのです。  
また、YとXについてノイズを考慮してMAP推定するのですが、  
これはここに書くのが超絶面倒いので書きません。  
具体的にはYとXのノイズの比をCとすると下記です。

この辺りを全て理解するためにはムーアペンローズの逆行列と  
MAP推定を勉強すると良いと思いますが、それはまた別のお話。

# dSPMの理屈

さて…MNEの式をよく眺めてみましょう。  
これは

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。

どうしてもノイズはのってしまうわけですが、  
そこで、なんとかする方法としてノイズの大きさを測っておいて、  
そのノイズと本来の反応を比べる様な割り算をしてやる方法があります。  
それをdSPMといいます。  
MNEが点数とすると、dSPMは偏差値のようなものですね。  
ここでノイズの分散をC、標準偏差をHとおきます。  
このノイズというのは環境そのものなので、空室でMEG取ることでつくれます。

これで正規化したもの(偏差値的な物)ができるはずです。  
結論として

とする時

とするのが僕のdSPMに対する理解です。  
間違ってたらごめん(´・ω・｀)

# 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。
* 脳磁気科学-SQUID計測と医学応用  
  これも絶版の和書です。僕は算数も英語が苦手なのですが、  
  この本はかなり数学チックなマニアックな内容の本で、初っ端から微積が出てきておののきます。  
  でも、避けて通りながら読むことも出来るので、数学嫌いな人でも読めるかもです。  
  何が嬉しいかというと、大まかにMEGの歴史だとか目的とかが書かれています。
* 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が、元が僕にとっては難しいものなので大変でした。  
  いい本です。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。  
  基礎を学ぶには本当に良いです。その後はchainerだとかtensorflowだとか  
  触れば良いんじゃないですかね？

# 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事例：連続ウェーブレット変換を中心に  
https://www.slideshare.net/ryosuketachibana12/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！と思われるかもですが、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炎上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、学習コストがかなり高いのと、真価を発揮するのは団体戦なのとで、熟練者向けですね。  
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

本書ではjupyterやspyderで環境構築をしました。  
つまり僕のオススメなのですが、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。

* 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。
* source tree  
  gitは使うための学習コストが結構高いです。  
  こいつはgitを簡単に使いこなす為のGUIツールです。  
  他にもgithub desktopとかgit kraken等色々あります。好きなの使えばいいです。
* atom  
  お前はgithub社の回し者かといわれ言われそうですが、  
  github社製のテキストエディタです。  
  カスタマイズ可能ですが、ちょっと重いですが、大人気。  
  jupyterとは逆方向に恐竜進化したツールですね。
* vim  
  vimは暗黒のエディタである。ククク…これを用いるのも一つの方法であろう。
* 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…)

# 参考文献

(まだ途中です…)  
- 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  
- Margherita Lai, Matteo Demuru, Arjan Hillebrand, Matteo Fraschini, A Comparison Between Scalp- And Source-Reconstructed EEG Networks  
- 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  
- https://surfer.nmr.mgh.harvard.edu/fswiki/FreeSurferWiki

1. 日本のプログラマ用のSNSの一つです。 [↑](#footnote-ref-1)
2. 脳波は電気信号を捉えますが、脳磁図は磁場を捉えます。電気と違って拡散しないので空間分解能に優れますが、ノイズに弱いです。超電導やシールドルームを使うので値段も高いです。http://www.elekta.co.jp/products/functionalmapping.html [↑](#footnote-ref-2)
3. 動物実験では脳に電極刺す実験はされていますが、人には刺せません。 [↑](#footnote-ref-3)
4. 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。 [↑](#footnote-ref-4)
5. http://biorxiv.org/content/early/2017/03/29/121764 [↑](#footnote-ref-5)
6. コンピュータ言語の一つ。速度を犠牲にして、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と似ていますが、pythonは無料でオブジェクト指向の汎用言語なので、応用範囲がスマホアプリにまで及び、習得して損をすることはまずないでしょう。 [↑](#footnote-ref-6)
7. 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んぐ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！ [↑](#footnote-ref-7)
8. 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。 [↑](#footnote-ref-8)
9. ブランドにこだわらずに探せば20コアとかのマシンが40万円も出せば買えます。グラボは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。 [↑](#footnote-ref-9)
10. anacondaはosの垣根を越えているので、大丈夫なのでしょうけれど僕は試していません。 [↑](#footnote-ref-10)
11. 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。 [↑](#footnote-ref-11)
12. 仮想化ソフトの一つ。ほぼネイティブの速度が出るのが特徴です。 [↑](#footnote-ref-12)
13. 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pyenvなどを使うのもありと思うのですが、mayaviというインストールの鬼門が存在するのでanacondaが楽です。 [↑](#footnote-ref-13)
14. 僕がjupyterを推す他の理由としてはクラスタ化して並列計算でゴリ押しが出来ること、有名なスクリプト言語の殆どに対応している事が挙げられます。 [↑](#footnote-ref-14)
15. 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。 [↑](#footnote-ref-15)
16. 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。 [↑](#footnote-ref-16)
17. いわゆる鍵認証方式というやり方です。セキュリティと利便性を両立できる良いやり方であり、  
    クラスタ作る時に限らず日常生活の中でもお勧めです。パスワード入れなくてもいいので楽ですし。 [↑](#footnote-ref-17)
18. 仮想化ソフトの一つ。ほぼネイティブの速度が出るのが特徴です。 [↑](#footnote-ref-18)
19. 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。 [↑](#footnote-ref-19)
20. 脳に水を流し込むシミ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 [↑](#footnote-ref-20)
21. nVidiaのGPUを使った高速な計算ができる開発環境 [↑](#footnote-ref-21)
22. 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。 [↑](#footnote-ref-22)
23. 脳に水を流し込むシミ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 [↑](#footnote-ref-23)
24. 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。 [↑](#footnote-ref-24)
25. pythonの正式名称 [↑](#footnote-ref-25)
26. 誇張ではありません。実際に効率の悪いpythonコードを最適化すると100倍速くなったりします。最適化なしでも2倍くらい速くなることもあります。 [↑](#footnote-ref-26)
27. 生の波形を見たいときなどにはそのほうが向いてる [↑](#footnote-ref-27)
28. 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。 [↑](#footnote-ref-28)
29. jupyter独自の単位です。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。  
    スッキリ見やすい解説用コードを書けるのがjupyterの強みです。MNE公式サイトではjupyterを採用しているので、今後jupyterを前提に話していきます。 [↑](#footnote-ref-29)
30. 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です。 [↑](#footnote-ref-30)
31. 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。 [↑](#footnote-ref-31)
32. 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。 [↑](#footnote-ref-32)
33. notchフィルタもバンドパスフィルタ同様賛否両論です。 [↑](#footnote-ref-33)
34. pythonistaはリスト内包表記とか使うんでしょうが、ここは簡単のためにappend使っています。というか、この程度の処理ならappendで困りません。 [↑](#footnote-ref-34)
35. 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ることは知ってます！ [↑](#footnote-ref-35)
36. 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。これがあるため、ソースベース解析にはコンピュータ・クラスタやワークステーションが必要なのです。 [↑](#footnote-ref-36)
37. この、永遠に続く波じゃないと無理っぽい感じになることをフーリエ変換の不確定性と言います。連続する波をぶつ切りにすると、端っこが不揃いになったりして、そこの処理もしないといけないです。あまり気の進む作業じゃないです。 [↑](#footnote-ref-37)
38. ちなみに、が1の場合はオイラーの等式と言います。 [↑](#footnote-ref-38)
39. 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」と言います。 [↑](#footnote-ref-39)