

画像解析ソフトのコマンド

筑波大学医学医療系精神医学

根本 清貴

本日の内容

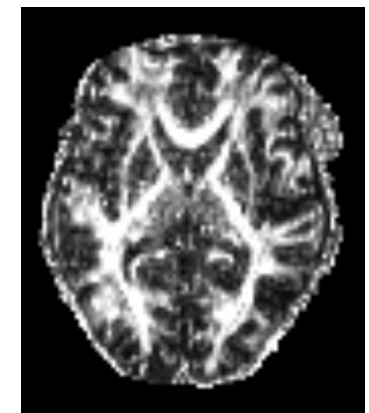
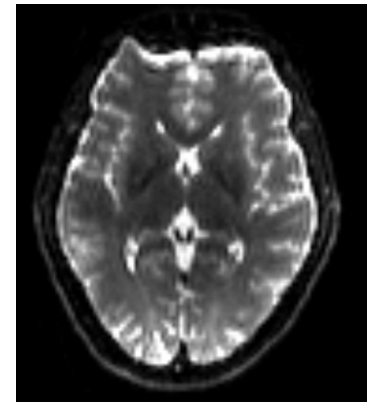
- 第1部:UNIX系OSのお作法を知る(兼予習)
- 第2部:画像解析ソフトのコマンドに慣れる
- 第3部:テキストを処理してみる
- 第4部:明日から使えるシェルスクリプトを知る
 1. 繰り返し
 2. 条件分岐

勉強会のルール

- ターミナルでタイプするものは、青色 (0000cc) で表示
 - 例: `$ fslhd V_ID001.nii.gz`
 - \$はタイプする必要はない
- スクリプトに記載する内容は緑色 (007e00) で表示
- コマンドやスクリプトではあるが、タイプしなくていいものは、紫色 (9933ff) で表示
- #以降は、解説でありタイプする必要はない
- 「フォルダ」=「ディレクトリ」
 - Linux/UNIXは、「ディレクトリ」を好む

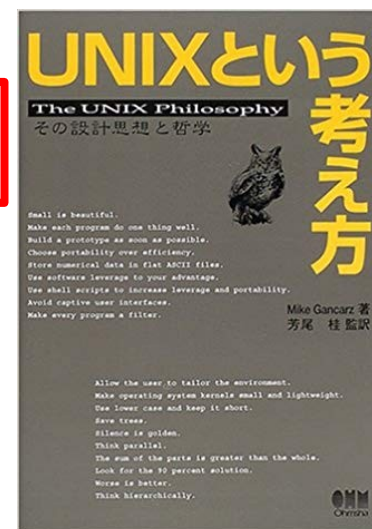
拡散MRI画像からFA画像を生成する

- DICOM画像→NIFTI画像の生成
 - `dcm2nii`
- 渦電流補正
 - `eddy_correct`
- B0画像の取り出し
 - `fslroi`
- 脳マスクの作成
 - `bet`
- FA画像の生成
 - `dtifit`



UNIXの思想

- 小さいものは美しい
- 各プログラムが一つのことをうまくやるようにせよ
- できる限り早く原型（プロトタイプ）を作れ
- 効率よりも移植しやすさを選べ
- 単純なテキストファイルにデータを格納せよ
- ソフトウェアを槌子(てこ)として利用せよ
- 効率と移植性を高めるためにシェルスクリプトを利用せよ
- 拘束的なユーザーインターフェースは作るな
- 全てのプログラムはフィルタとして振る舞うようにせよ



UNIX哲学の脳画像解析への応用

- 1つのコマンドはひとつの処理
 - MRI画像の前処理は複数の段階を踏む
 - ひとつの段階にひとつのコマンドを用いる
- ソフトウェアをてことして利用する
 - この意味するところは、「他の人の作ったソフトは利用させてもらう」
 - DICOM → NIFTI 変換ツールは、dcm2nii(x) というよいツールがあるので、FSLは、独自のツールを準備していない
- この背景思想を理解することで理解が進む
 - 脳画像はコマンドの組み合わせ＝スクリプト化が前提！

脳画像解析ツールのヘルプ

- FSLなどのコマンドのヘルプは、ネットよりも、プログラムそのものに記載されている方が充実していることが多い
- ヘルプは、たいてい、オプション `-h` か、引数やオプションを指定せずにタイプすることで見ることができる

```
$ dcm2niix -h
```

```
$ eddy_correct -h
```

```
$ fslroi -h
```

```
$ bet -h
```

```
$ dtifit -h
```

データのコピー

- あとで復習ができるように、データを `~/shell_practice` にコピー

```
$ cd ~/git/shell-basic/data/
```

```
$ ls
```

```
$ cp -rv * ~/shell_practice/
```

```
$ cd ~/shell_practice/
```

```
$ ls
```

```
UnixIntro mri pair scripts textprocessing zip
```

- `cp -rv` は再帰的に(`-r`)メッセージを表示しながら(`-v`)コピー
- 本日、以降の作業ディレクトリは `~/shell_practice`

拡散MRI画像からFA画像を生成する

- DICOM画像→NIFTI画像の生成
 - `dcm2nii`
- 渦電流補正
 - `eddy_correct`
- B0画像の取り出し
 - `fslroi`
- 脳マスクの作成
 - `bet`
- FA画像の生成
 - `dtifit`

dcm2nii

- Chris Rordenが作成している DICOM画像
→NIFTI画像 変換ツール
- 長らく dcm2nii が普及しているが、その後継
- 日本でまだ十分に知られていないので紹介
- 今後主流になってくる
- MRicroGLに搭載されている

dcm2nix のヘルプを確認

- ターミナルから以下をタイプ

```
$ dcm2nix -h | less
```

- 表示内容を次スライドでじっくり確認

Usage（使用法）が最も大事

- Usage はコマンドの使用法であり、一番大事な記載
- < > に囲まれているところは、「必須」という意味
- [] に囲まれているところは、「オプション」であり必須ではない
- 今、dcm2niix の usage は以下
 - **usage: dcm2niix [options] <in_folder>**
- 必須なのは、dcm2niix の後に「入力フォルダを指定すること」とわかる
- ヘルプの表示を [q] で終了

dcm2niix をデフォルトで実行

```
$ pwd # ~/shell_practice
```

```
$ cd mri/
```

```
$ ls subj1
```

```
diffusion volume
```

```
$ dcm2niix subj1
```

```
$ ls subj1
```

```
次スライド
```

```
$ ls subj2
```

```
diffusion volume
```

```
$ dcm2niix subj2
```

```
$ ls subj2
```

```
次スライド
```

dcm2nii の出力

```
$ ls subj1
```

```
diffusion
```

```
subj1_3DT1_20180912172445_2.json
```

```
subj1_3DT1_20180912172445_2.nii
```

```
subj1_Diffusion30axis_20180912172445_3.bval
```

```
subj1_Diffusion30axis_20180912172445_3.bvec
```

```
subj1_Diffusion30axis_20180912172445_3.json
```

```
subj1_Diffusion30axis_20180912172445_3.nii
```

```
volume
```

- ファイル名がまどろっこしい
- 出力ディレクトリを変えられないか？ → オプションを確認

オプション

Options :

-1..-9 : gz compression level (1=fastest, 9=smallest)

-b : BIDS sidecar (y/n, default y)

-ba : anonymize BIDS (y/n, default y)

-f : filename (%a=antenna (coil) number, %c=comments, %d=description, %e echo number, %f=folder name, %i ID of patient, %j seriesInstanceUID, %k studyInstanceUID, %m=manufacturer, %n=name of patient, %p=protocol, %s=series number, %t=time, %u=acquisition number, %z sequence name; default '%f_%p_%t_%s')

-h : show help

-i : ignore derived, localizer and 2D images (y/n, default n)

-t : text notes includes private patient details (y/n, default n)

-m : merge 2D slices from same series regardless of study time, echo, coil, orientation, etc. (y/n, default n)

-o : output directory (omit to save to input folder)

-p : Philips precise float (not display) scaling (y/n, default y)

-s : single file mode, do not convert other images in folder (y/n, default n)

-t : text notes includes private patient details (y/n, default n)

-v : verbose (n/y or 0/1/2 [no, yes, logorrheic], default 0)

-x : crop (y/n, default n)

-z : gz compress images (y/i/n, default n) [y=pigz, i=internal, n=no]

BIDS; Brain Imaging Data Structure





- By using this standard you will benefit in the following ways:
 - It will be easy for another researcher to work on your data. To understand the organization of the files and their format you will only need to refer them to this document. This is especially important if you are running your own lab and anticipate more than one person working on the same data over time. By using BIDS you will save time trying to understand and reuse data acquired by a graduate student or postdoc that has already left the lab.
 - There is a growing number of data analysis software packages that can understand data organized according to BIDS.
 - Databases such as OpenNeuro.org, LORIS, COINS, XNAT, SciTran and others will accept and export datasets organized according to BIDS. If you ever plan to share your data publicly (nowadays some journals require this) you can speed up the curation process by using BIDS.
 - There are validation tools (also available online) that can check your dataset integrity and let you easily spot missing values.

BIDS関連ファイル

- 拡張子が .json として保存されている
- 以下で確認

```
$ cd subj1
```

```
$ less subj1_Diffusion30axis_20180912172445_3.json
```

- タブを上手に使うこと!
 - Tabキーを  で表すと
 - lessの後は s  D  j  でいけるはず

jsonファイルの中身(抜粋)

```
"Manufacturer": "Siemens",  
"ManufacturersModelName": "Verio",  
"MagneticFieldStrength": 3,  
"FlipAngle": 90,  
"EchoTime": 0.081,  
"RepetitionTime": 14.1,  
"SliceTiming": [  
    0,  
    7.1725,  
    0.1875,  
    7.36,  
"PhaseEncodingDirection": "j-",
```

TE, TRなどの情報が記載されている

Slice Timingが interleaved であることがわかる

位相方向は、j- すなわちA>>P 方向

dcm2nii の出力ファイル名

-f : filename (%a=antenna (coil) number, %c=comments, %d=description, %e echo number, %f=folder name, %i ID of patient, %j seriesInstanceUID, %k studyInstanceUID, %m=manufacturer, %n=name of patient, %p=protocol, %s=series number, %t=time, %u=acquisition number, %z sequence name; default '%f_%p_%t_%s')

- 今のファイル名は
 - subj1_3DT1_20180912172445_2.nii
 - subj1_Diffusion30axis_20180912172445_3.nii
- フォルダ名とプロトコル名だけ使えば今回はよさそう
 - ファイル名のオプションは、%f_%p とする

dc2nifti の出力ディレクトリ

-o : output directory (omit to save to input folder)

- 指定すれば出力ディレクトリを変更できる
- nifti というディレクトリを作成し、そこに出力することとする

dcm2niix をオプション指定で実行

```
$ pwd # ~/shell_practice/mri/subj1
```

```
$ cd .. # ~/shell_practice/mri/
```


```
$ ls
```

```
subj1 subj2
```

```
$ mkdir nifti # niftiディレクトリを作成
```

```
$ dcm2niix -f %f_%p -o nifti/ subj1
```

```
$ dcm2niix -f %f_%p -o nifti/ subj2
```

 キーを使って行末の "1" を "2" に変えればよい

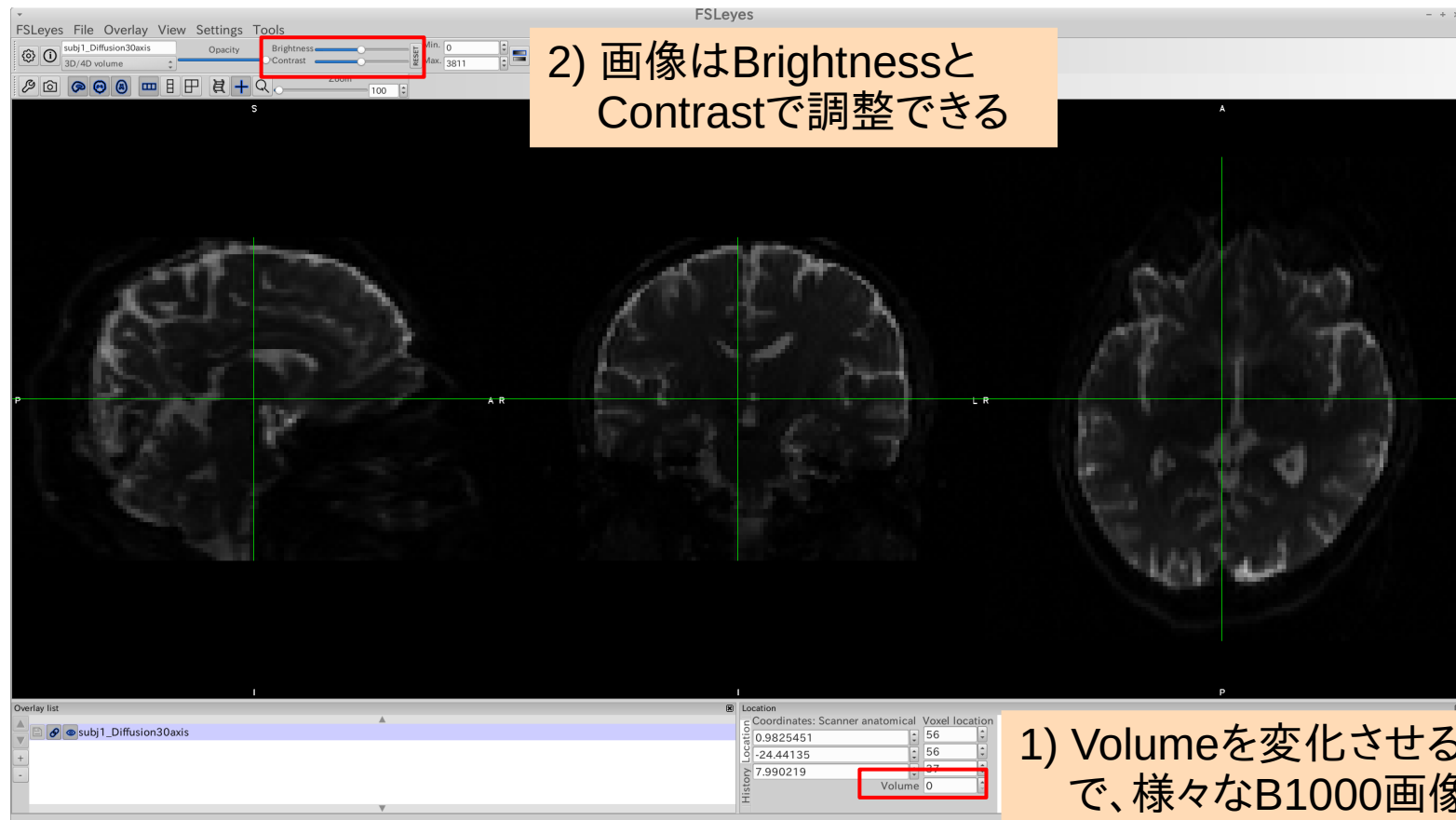
```
$ ls nifti/
```

```
subj1_3DT1.json      subj2_3DT1.json
subj1_3DT1.nii       subj2_3DT1.nii
subj1_Diffusion30axis.bval subj2_Diffusion30axis.bval
subj1_Diffusion30axis.bvec subj2_Diffusion30axis.bvec
subj1_Diffusion30axis.json subj2_Diffusion30axis.json
subj1_Diffusion30axis.nii subj2_Diffusion30axis.nii
```

fsleyes で拡散MRIを確認

```
$ cd nifti/
```

```
$ fsleyes subj1_Diffusion30axis.nii &
```



& の意味

- コマンドの最後に & をつけると、そのコマンドは、「バックグラウンド」で実行される
- ターミナルでそのまま他の作業ができる
- もし、先の `fsleyes` で & をつけないと、ターミナルは `fsleyes` が起動している間は何も受け付けなくなる

拡散MRI画像からFA画像を生成する

- DICOM画像→NIFTI画像の生成
 - `dcm2niix`
- 渦電流補正
 - `eddy_correct`
- B0画像の取り出し
 - `fslroi`
- 脳マスクの作成
 - `bet`
- FA画像の生成
 - `dtifit`

eddy_correct

- 拡散MRIの渦電流によるゆがみを補正する
- 現行のFSLでは、eddy (eddy_openmp/cuda) が推奨されているが、eddy_correct は簡易な補正方法としてまだ残っている
- これもヘルプを確認

```
$ eddy_correct -h
```

eddy_correct -h

- Usage: eddy_correct <4dinput> <4doutput>
<reference_no> [<interp>]
 - Choose interp from {trilinear,spline} def – trilinear
- 必須で指定するものは、4次元入力画像、4次元出力画、reference_no
 - 出力画像は自分でファイル名を決める
 - eddy current correction ということでファイル名の後に _ecc とつけることとする
- reference_no は、B0画像のある位置
 - bval ファイルを見るとわかる(次スライド)
- オプションとして、補完方法(interp → interpolation) が trilinear か spline があるが、デフォルト (def → default) は trilinear であり、特に何もする必要がない

B0 ファイルの位置の確認

```
$ pwd                # ~/shell_practice/mri/nifti
```

```
$ cat subj1_Diffusion30axis.bval
```

```
0 1000 1000 1000 1000 1000 1000 1000 1000
1000 1000 1000 1000 1000 1000 1000 1000 1000
1000 1000 1000 1000 1000 1000 1000 1000 1000
1000 1000 1000 1000 1000 1000 1000 1000
```

- 最初が0であとは1000
- FSLでは、reference_noは0から数える → B0の位置は 0 となる

変数の利用


- FSLのツールでは、nifti形式の拡張子(.nii, nii.gz)は省略して指定できる
 - `dmri.nii.gz` → `dmri` とだけ指定すればよい
- このルールは、「変数」を使うときに威力を発揮する
- 変数の指定方法は簡単
 - 変数名=値
 - ポイント:=の前後には空白入れない(Bashでの規則)
- 変数を利用するときには、変数名の前に `$` をつける
 - 具体例は次スライドで説明

変数の設定

- 変数を使うところ
 - 何度も長いファイル名をタイプしたくない
 - 同じ文字列は再利用したい
- `dmri` という変数を設定することとし、そこに `subj1_Diffusion30axis` という値を代入することとする
- `echo` コマンドで、変数に何の値が代入されているかを確認できる

```
$ echo $dmri    #何も表示されない(変数dmriは空)
```

```
$ dmri=subj1_Diffusion30axis
```

- #変数に代入するときも  を使うことができる!

```
$ echo $dmri
```

```
    subj1_Diffusion30axis
```

- 変数はターミナルを閉じるまで有効

変数の利用法

- 変数を使うときは、変数名の前に \$ をつける
- 変数名を {} でくくると、文字列をつなげることができる
 - `dmri=subj1_Diffusion30axis`
 - `${dmri}.nii` → `subj1_Diffusion30axis.nii`
 - `${dmri}.bval` → `subj1_Diffusion30axis.bval`
 - `${dmri}_ecc` → `subj1_Diffusion30axis_ecc`
 - `a$dmri` → `asubj1_Diffusion30axis`
 - `b${dmri}.txt` → `bsubj1_diffusion30axis.txt`

eddy_correct に変数を応用する

- Usage: eddy_correct <4dinput> <4doutput>
<reference_no>
 - 4dinput: subj1_Diffusion30axis とすると
 - 4doutput: subj1_Diffusion30axis_ecc として
- 変数を上手に使って以下をタイプ

```
$ pwd # ~/shell_practice/mri/nifti
```

```
$ echo $dmri
```

```
subj1_Diffusion30axis
```

```
$ eddy_correct $dmri ${dmri}_ecc 0 #5-8分程度かかる
```

```
$ ls *_ecc* #ファイル名に _ecc がついているものを表示
```

```
subj1_Diffusion30axis_ecc.ecclog
```

```
subj1_Diffusion30axis_ecc.nii.gz
```

拡散MRI画像からFA画像を生成する

- DICOM画像→NIFTI画像の生成
 - `dcm2niix`
- 渦電流補正
 - `eddy_correct`
- B0画像の取り出し
 - `fslroi`
- 脳マスクの作成
 - `bet`
- FA画像の生成
 - `dtifit`

fslroi

- 画像から関心領域を取り出す
- 4次元画像から任意の3次元画像を取り出すこともできる
- 今は、B0画像だけ取り出し、B0画像に対して頭蓋骨の除去を行う

```
$ fslroi -h
```

fslroi -h

Usage: fslroi <input> <output> <xmin> <xsize> <ymin>
<ysize> <zmin> <zsize>

fslroi <input> <output> <tmin> <tsize>

fslroi <input> <output> <xmin> <xsize> <ymin>
<ysize> <zmin> <zsize> <tmin> <tsize>

Note: indexing (in both time and space) starts with 0 not 1! Inputting -1 for a size will set it to the full image extent for that dimension.

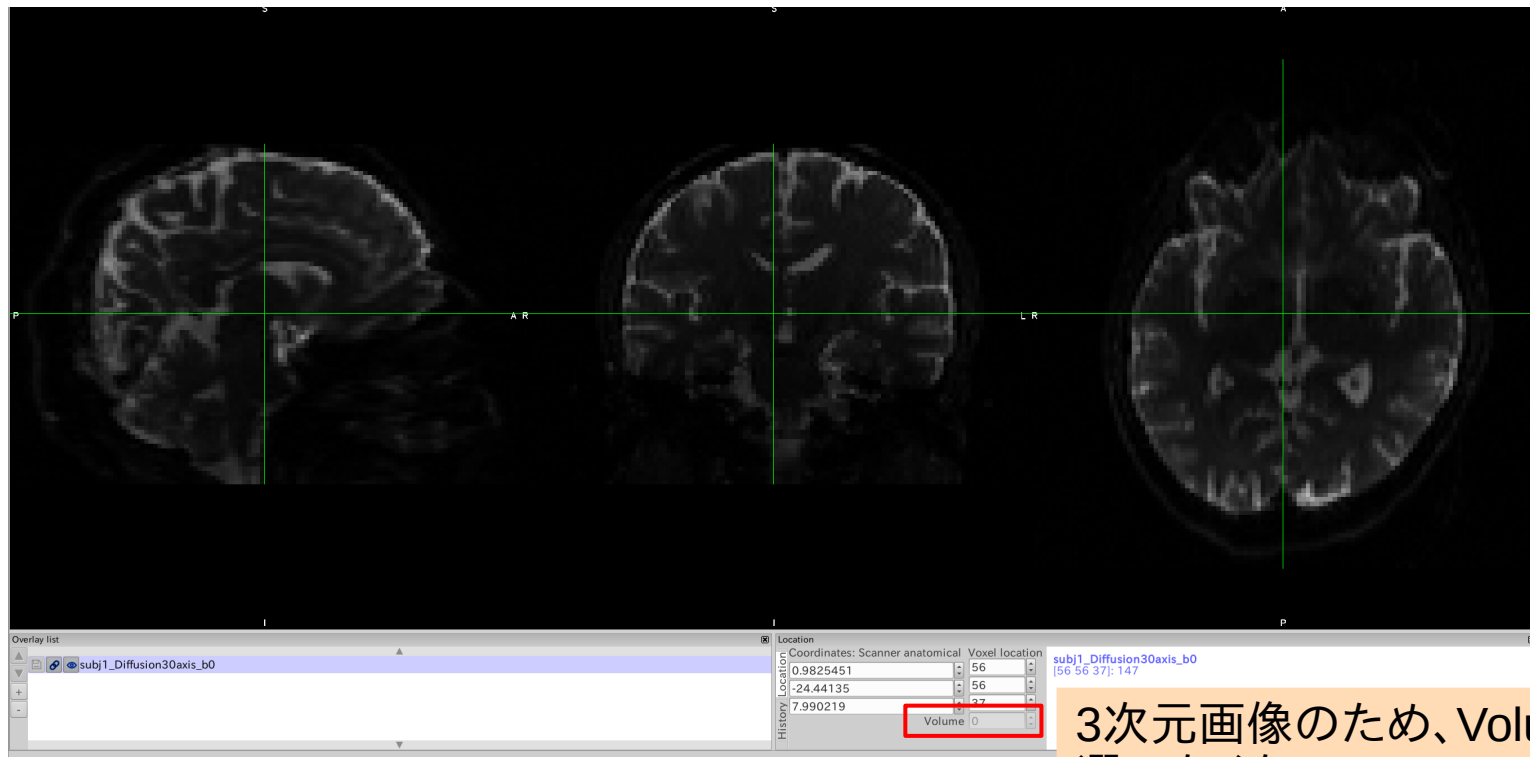
- 今は、入力画像から、B0画像＝一番最初の画像だけを取り出したい
 - この場合、<tmin> が 0 となり、<tsize> が 1 となる
- 出力画像は、B0画像なので、`${dmri}_b0` とする

fslroi で B0画像のみ取り出す

```
$ fslroi ${dmri}_ecc ${dmri}_b0 0 1
```

```
$ fsleyes ${dmri}_b0 &
```

- 4次元画像ではなく、3次元画像になっている



3次元画像のため、Volumeは
選べなくなっている

bet

- brain extraction tool
- 頭蓋骨除去によく使われる
- 脳を取り出し、マスク画像を作成することもできる
- FA画像の計算のために、計算領域を脳にしぼることが必要なため、bet を用いて脳のマスクを作成する
- 入力画像には `${dmri}_b0` を使用し、出力画像のファイル名は `${dmri}_brain` としたい

\$ bet -h で使い方を確認

bet -h

Usage: **bet** <input> <output> [options] #input と outputが必須

Main bet2 options:

- o generate brain surface outline overlaid onto original image
- m generate binary brain mask**
- s generate approximate skull image
- n don't generate segmented brain image output
- f <f> fractional intensity threshold (0→1); default=0.5; smaller values give larger brain outline estimates** #defaultではだめなことが多い
- g <g> vertical gradient in fractional intensity threshold (-1→1); default=0; positive values give larger brain outline at bottom, smaller at top**
- r <r> head radius (mm not voxels); initial surface sphere is set to half of this**
- c <x y z> centre-of-gravity (voxels not mm) of initial mesh surface.**
- t apply thresholding to segmented brain image and mask
- e generates brain surface as mesh in .vtk format

bet でマスク作成

```
$ bet ${dmri}_b0 ${dmri}_brain -f 0.3 -m
```

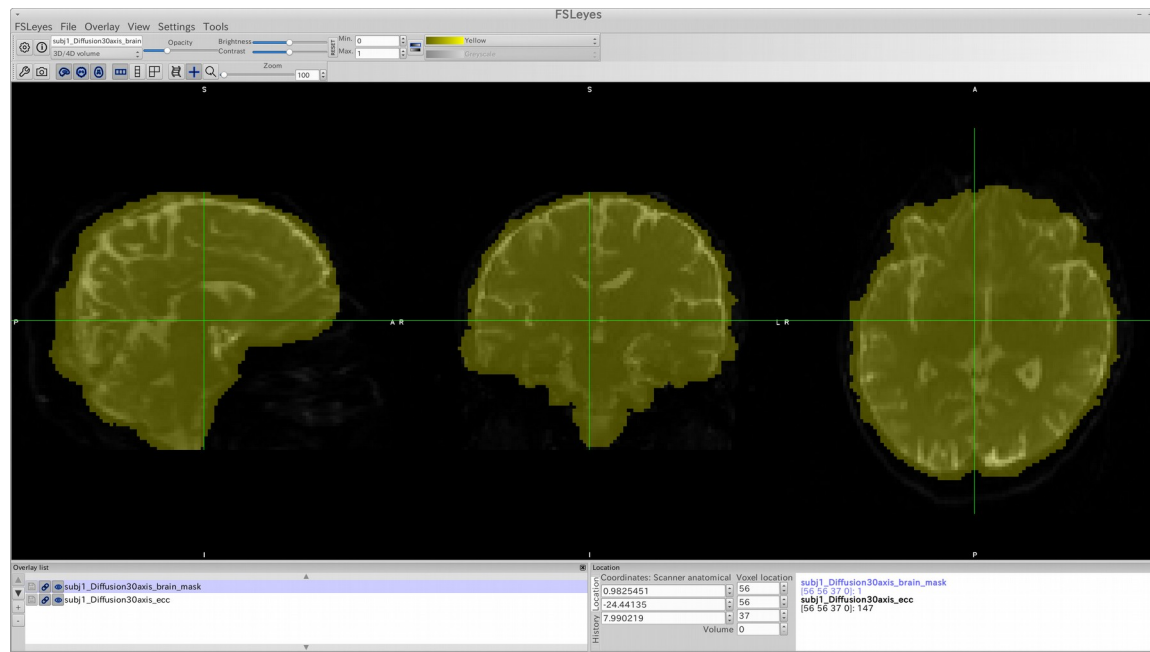
```
$ ls *brain*
```

```
subj1_Diffusion30axis_brain.nii.gz
```

```
subj1_Diffusion30axis_brain_mask.nii.gz
```

```
$ fsleyes ${dmri}_b0 ${dmri}_brain_mask -a 40 -cm yellow &
```

#fsleyesの -a オプションと -cm オプションは何を意味するでしょうか？



拡散MRI画像からFA画像を生成する

- DICOM画像→NIFTI画像の生成
 - `dcm2niix`
- 渦電流補正
 - `eddy_correct`
- B0画像の取り出し
 - `fslroi`
- 脳マスクの作成
 - `bet`
- FA画像の生成
 - `dtifit`

dtifit

- 拡散MRI画像から、FA画像やMD画像など、拡散テンソル画像を生成するプログラム

`$ dtifit -h` で使い方を確認

Usage:

`dtifit -k <filename>`

`dtifit --verbose`

Compulsory arguments (You MUST set one or more of):

`-k, --data` dti data file

`-o, --out` Output basename

`-m, --mask` Bet binary mask file

`-r, --bvecs` b vectors file

`-b, --bvals` b values file

dtifit の引数

-k, --data `${dmri}_ecc`
-o, --out `subj1` ←ここは任意に決められる
-m, --mask `${dmri}_brain_mask`
-r, --bvecs `${dmri}.bvec`
-b, --bvals `${dmri}.bval`

- `--out` で指定する値で出力ファイルの名前が決まる
 - `--out=taro` とすると、出力ファイルのファイル名は `taro_FA`, `taro_MD`, `taro_V1` などとなる

コマンドを複数行で入力

- コマンドは、¥ (\ で表されることもある)を入れると改行しても同じコマンドの続きとみなされる
- dtifit のようなプログラムのように引数が多いプログラムは、\ を上手に使って表現すると理解しやすい
 - 実際に入力する際は、1行でよい(\ は入力しなくてよい)

```
$ dtifit --data=${dmri}_ecc \  
        --out=subj1 \  
        --mask=${dmri}_brain_mask \  
        --bvecs=${dmri}.bvec \  
        --bvals=${dmri}.bval
```

```
$ ls
```

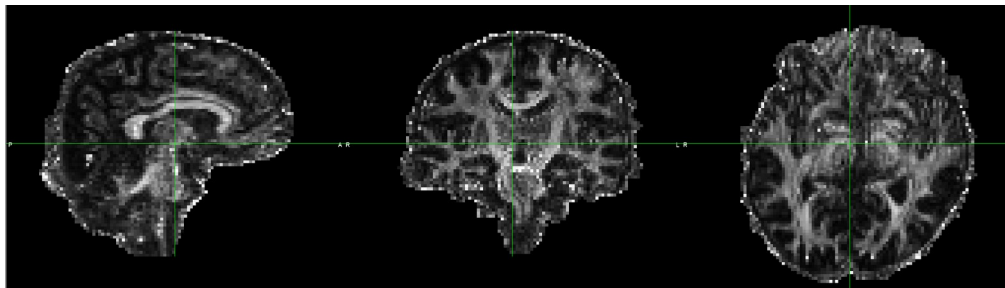
オプションの使い分け

- dtifit は実際は引数ではあるが、形式としてはオプションのような指定の仕方をとっている
- オプションには、ハイフンひとつのものとハイフン2つのものがあるが、ハイフン1つの場合、半角スペースを入れて指定、ハイフン2つのものは、空白を入れずに=を入れて指定という場合が多い
- dtifitでハイフン一つの指定方法を使うと以下のように示される

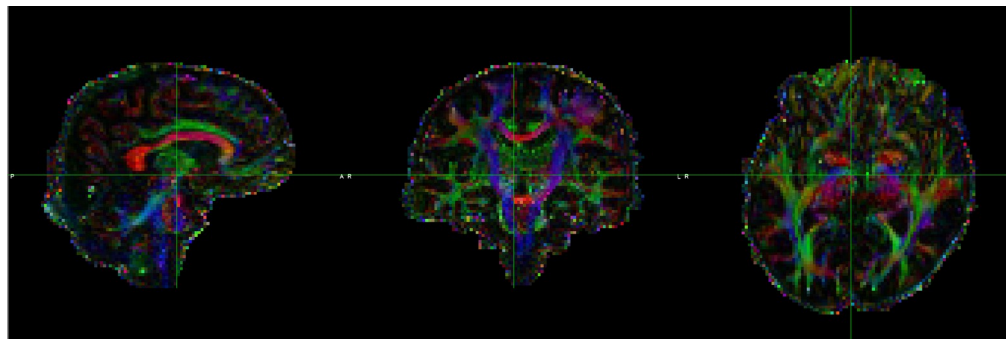
```
$ dtifit -k ${dmri}_ecc \  
          -o subj1 \  
          -m ${dmri}_brain_mask \  
          -r ${dmri}.bvec \  
          -b ${dmri}.bval
```

FA画像の確認

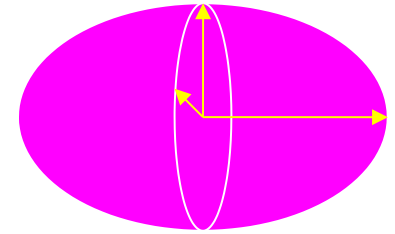
```
$ fsleyes subj1_FA.nii.gz &
```



```
$ fsleyes subj1_FA.nii.gz subj1_V1.nii.gz \  
-ot rgbvector \  
-mo subj1_FA.nii.gz &
```



V1画像

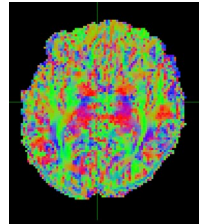


- V1画像は、拡散MRIから作成されたテンソルのうち、スカラーがもっとも大きいベクトルを画像化したもの
- ベクトルが3次元であるため、x軸方向、y軸方向、z軸方向の情報をもった4次元画像として作成されている
- `fsleyes` で見る時、3-direction vector image (RGB) か 3-direction vector image (Line) で表示する
- RGB: DTIベクトルは x軸方向に Red, y軸方向に Green, z軸方向に Blue で表示される。紫のような色は、RGBの混合→ベクトルの向きが x, y, z の混合で表示されているということ
- Line: DTIベクトルは、小さな線の集合として表示される。x, y, z軸方向にそれぞれ R, G, B の色が割り当てられている
- RGB モードの時は、信号強度をFA画像などで修飾することができる。つまり、FAが大きい領域は色合いが強くすることができる

V1画像のさまざまな表示

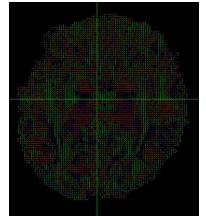
- RGBベクトルで表示

```
$ fsleyes subj1_V1 -ot rgbvector &
```



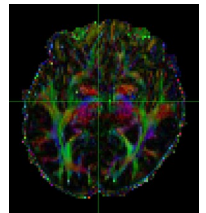
- Lineベクトルで表示

```
$ fsleyes subj1_V1 -ot linevector &
```



- RGBベクトル画像を FA画像で modulate

```
$ fsleyes subj1_FA subj1_V1 \  
-ot rgbvector -mo subj1_FA &
```



#subj1_FA, subj1_V1の順番に指定することで、subj1_FA
が下のレイヤーに来て、subj1_V1 が上にくる

-mo subj1_FA で、subj1_FA の値を使うことを指定

変数を使うことのメリット

- 変数に違う値を代入すれば、これまでのコマンドをすべて再利用できる!
- subj2に対して同じ作業をしたい場合、

```
$ dmri=subj2_Diffusion30axis
```

とすれば、これまでうったコマンドをほとんど再利用できる

- `history` や `↑` `↓` キーを上手に利用し、少し変更するだけでよい

演習

1. subj2 でのFA画像の作成

```
$ dmri=subj2_Diffusion30axis
```

として、subj2に対してFA画像を作成してください

`history` を上手に使ってみましょう

2. `fsleyes` のヘルプは簡易版とフル版があります

a) フル版のヘルプを表示するためのオプションを探してください

b) フル版のヘルプを `less` を使って表示するにはどうしたらよいのでしょうか？

演習の回答例(1)

1. subj2 でのFA画像の作成

```
$ dmri=subj2_Diffusion30axis
```

として、subj2に対してFA画像を作成してください

```
$ eddy_correct $dmri ${dmri}_ecc 0
```

```
$ fslroi ${dmri}_ecc ${dmri}_b0 0 1
```

```
$ bet ${dmri}_b0 ${dmri}_brain -f 0.3 -m
```

```
$ fsleyes ${dmri}_ecc ${dmri}_brain_mask -a 40 -cm yellow &
```

```
$ dtifit --data=${dmri}_ecc --out=subj2 \
```

```
--mask=${dmri}_brain_mask --bvecs=${dmri}.bvec \
```

```
--bvals=${dmri}.bval
```

```
$ fsleyes subj2_FA &
```

```
$ fsleys subj2_FA subj2_V1 -ot rgbvector -mo subj2_FA &
```

演習の回答例(2)

2. **fsleyes** のヘルプは簡易版とフル版があります

a) フル版のヘルプを表示するためのオプションを探してください

```
$ fsleyes -h から
```

```
-fh, --fullhelp  Display all FSleyes  
options and exit
```

b) フル版のヘルプを **less** を使って表示するにはどうしたらよいでしょうか？

```
$ fsleyes -fh | less  #パイプを使う
```

質疑応答