FSL course: シェルスクリプト演習 *

FMRIB Group
翻訳: 根本 清貴

2014 年 2 月 20 日

* http://fsl.fmrib.ox.ac.uk/fslcourse/lectures/scripting/ から演習部分を抽出しました

スクリプト演習セッションにようこそ。この演習は多くのセクションから構成されており、非常に単純なものからはじまり、少しずつ難しいスクリプトの練習へと進んでいきます。一回ですべての演習を終えられるとは期待しないでください。

スクリプトの経験があまりないのであれば、最初のいくつかの演習に時間がかかるでしょう。しかし一度はじめたら、徐々に慣れていき、少しずつ簡単に感じるようになってくると思います。

スクリプトの経験が十分にあるならば、すぐにわかる演習はとばして後半の FSL ユーティリティに特化したスクリプトの演習にうつってください。

この演習はスクリプトの基本と様々な FSL のコマンドラインユーティリティの両方をカバーしています。以下に各演習の概要を示します。

- **最初のスクリプト**
  この演習では `bet` を使ったもっとも単純なスクリプトを扱います。
- **Echo とワイルドカード**
  `fslmerge, fslsplit` を使ってワイルドカードや echo の使い方に慣れていきます。
- **コマンドラインの引数**
  演習 1 のスクリプトを修正し、ユーザが指定した引数を`$1` などとして使います。上級編としてはより複雑な引数の使い方として `shift` を紹介します。
- **変数**
  `fslval` を用いて画像のボクセル次元を読み込み、表示します。このスクリプトは今後修正して使用します。
- **If と Test**
  if コマンドと test コマンドの正しい構文を確認し、ファイルが存在するかどうかを確認します。
- **バッチ処理**
  ディレクトリの中にある全部のファイルもしくはユーザが指定したファイルに対して実行するスクリプトを作成します。例では、`bet` や `fslmaths` の他に `basename` を使います。
- **容積と FOV を計算する**
  計算機 `bc` を用いて画像の定量化とスケーリングを行います。このために、その前で用いたスクリプトと `fslval, fslstats, fslmaths` を使います。
- **断層像一覧の作成**
  ループを使って画像の断層像の一覧を作成します。このために `slicer` と `convert` を使います。
- **刺激提示ファイルの処理**
  一般的な刺激提示ファイルを使って Feat EV ファイルをカスタマイズします。`grep, awk, sed` を使って列や項目を操作し、思い通りに出力する方法を学びます。Block-design でない fMRI の実験を行う場合に非常に便利な演習です。
- **ファイル名の操作**
  .nii.gz のような拡張子を無視するスクリプトを作成し、Feat のデザインファイルの中のパスを変更します。このために `sed, basename` を使います。
- **スクリプト上級編**
  難易度の高い演習です。背景画像の正中矢状断画像に MIP 画像を重ねあわせ、回転させるアニメーションを作成します。

1 最初のスクリプト-BET を使う

データフォルダに移動しましょう。(訳注:http://fsl.fmrib.ox.ac.uk/fslcourse/ の Data Files セクションからデータをホームディレクトリにダウンロードし、展開すると上記のディレクトリが生成されます)

```
cd ~/fsl_course_data/scripting
```

最初の演習では、画像を他の画像に位置合わせを行う前に、BET を使うスクリプトを作ります。

1. テキストファイルを編集する
   bet_vol という名前のファイルを nano テキストエディタを使ってつくりましょう。ターミナルから次のようにタイプします。
   ```
   nano bet_vol
   ```
   メモ：
   - エディタは自分のお好きなものでかまいません。
2. スクリプトを作成するこのファイルにスクリプト（実際はコマンドひとつですが）を書きます。BET を vol という名前の画像に適応し、結果を vol_brain として保存します。ターミナルで使うコマンドと同じコマンドです。
   ```
   bet vol vol_brain
   ```
   スクリプトの最初の行は#!/bin/sh であることを忘れないようにしてください。
3. スクリプトを実行する
   スクリプトを保存します。
   画像 im1 を vol としてコピーします。(拡張子.nii.gz をつけてください)
   スクリプトに実行権限をつけます。
   ```
   chmod a+x bet_vol
   ```
   スクリプトを実行します。./bet_vol とタイプしてください。(カレントディレクトリ (".") がパスに設定されていないかもしれないため、"./"が必要です)
4. 実行結果を確認する
   スクリプトを走らせた後、vol_brain ができているか、そして意味のある結果となっているかを確認します (視覚的にチェックするには、slices を使います)。
5. スクリプトにコメントをつける
   スクリプトにコメントをつけ、何をしているのかを書きます。
   コメントは行頭以外のどこにでもつけることができ、必ず行頭に#がきます。

2 Echo とワイルドカード

1. すべての画像を列挙する
   新しいスクリプト (適当な名前にしてください。訳注:listimg などでどうでしょうか) を作成し、echo を使って現在いるディレクトリとその上のディレクトリにいるすべての.nii.gz ファイルを表示してください。

2. 一部の画像を列挙する
   上記のスクリプトを修正し、vol000X.nii.gz ファイルのうち、X が偶数のファイルだけ表示されるようにしてください。
3. 画像をマージする
   上記のスクリプトをさらに修正し、先程指定した vol000X.nii.gz ファイルのうち、X が偶数の 3D 画像ファイルを単一の 4D 画像ファイルにマージしてみましょう。(ファイル名は任意ですが、my4Dvolume としてみましょう)
   このためには fslmerge を使います。
4. 出力画像を確認する
   出力された my4Dvolume.ii.gz が 5 時点の情報を持っていることを fslhd を使って確認しましょう。
   メモ: 端末から fslhd と入力し、その出力の dim4 にある数字を見てみてください。

3 コマンドラインの引数

1. コマンドラインで画像のファイル名を指定する
   (最初に使った)bet_vol スクリプトを修正し、コマンドラインの引数で指定した画像に対して BET が動作するようにします。

   つまり、もし、./bet_vol im2
   と入力すると、im2 に対して BET が走り、結果は im2_brain として保存されます。
2. テスト im2 を入力画像にしてスクリプトを走らせてみてください。
   出力画像の名前が正しいか、そして画像自体が正しく見えるか確認してください。
   メモ: 引数にはファイル名のうち、.nii.gz の部分は含めません。
3. BET に引数を渡す (上級)
   変数$@を使うとその他の引数も BET に渡すことができます (例 -f 0.4 -g 0.1)
   しかし、ファイル名は繰り返したくありません。これは shift を用いると取り除くことができます。
   一度 shift がスクリプトの中で実行されると、第 1 の引数$1 が取り除かれ、その他の引数がひとつずつ小さくなります。
   つまり、$2 は$1 になり、$3 は$2 になります。
   shift と$@を使うことで、ファイル名以外の引数を BET のオプションにセットできます。

4 コマンドと変数

1. ボクセルサイズを見つける
   画像を第 1 の引数と指定すると、pixdim1, pixdim2, pixdim3 の各々の値 (mm 単位でのボクセルサイズ) を個々の変数におさめるスクリプトを作成してください。
   ヒント: バッククオートのスライドで使った fslval の使用方法を参考にしてください。
2. ボクセルサイズを表示する
   これらの変数の値を 1 行で次の形で表してください

`1.0 * 2.0 * 1.0`

数字が変数にセットされた値です。

ヒント: *の前にバックスラッシュを使うことを忘れないでください

3. スクリプトを走らせてみてください。後で使いますのでとっておいてください。

5 引数が妥当かを確認する

この演習では再び `bet_vol` を修正し、正しい引数が与えられているかを確認します。前のバージョンでは引数が指定されないとエラーとなっていたことに注意してください。

1. 引数の数を確認する
   `bet_vol` を修正し、`$#`がゼロでないかどうかを test することで引数が実際に指定されたかどうかを確認します。
   もし test が真の値を返すならば、警告 (とスクリプトの使い方) を表示し、`exit 1` でスクリプトを止めます。(訳注: 原文では-1 となっていますが、exit 1 が正しいと思われます)
2. ファイルの妥当性を確認する
   スクリプトの中で、`${1}.nii.gz` が存在して読み込み可能かを test します (`help test` を見てください)。もし存在しなければエラーメッセージを表示し、終了します。
3. スクリプトを試す
   スクリプトが動作するかを確認します。最初に存在するファイルを指定し、次に存在しないファイルを指定します。

6 画像のバッチ処理

1. 拡張子を除いたファイル名 (basename) を変数として指定する
   ”baserun” という名のスクリプトを準備します。このスクリプトでは、あるディレクトリにあるすべてのファイルの拡張子を除いたファイル名 (vol0001.nii.gz ならば vol0001) を表示 (echo) します。このためには `basename` コマンドを使う必要があるでしょう。ヒント: `basename vol0001.nii.gz .nii.gz` の出力は `vol0001` になります。
2. 複数の入力を指定する
   今のスクリプトを修正してディレクトリのすべてのファイルではなく、引数に指定したファイルをすべて使うようにしましょう。
   もし
   `./baserun im2.nii.gz im3.nii.gz`
   とタイプすると結果は
   `im2`
   `im3`
   となります。
3. BET を複数回実行する

BET をあるディレクトリにあるすべての画像に対して行います。
出力画像は 画像ファイル_brain となります。
`./baserun im2.nii.gz im3.nii.gz`
とタイプして、BET がこの 2 つの画像に対して行われたかを確認してください。

7 ボクセルの容積を計算する

1. ボクセルの容積を計算する
   pixdim1, pixdim2, pixdim3 を見つけたスクリプトを修正してボクセルの容積を mm^3 で計算し、その結果を表示するようにしてください。
2. 画像の容積を計算する
   同様に dim1, dim2, dim3 を見つけて画像全体の容積を mm^3 で計算してください。
3. Field of View (有効視野) を計算する
   各方向の FOV を mm で求め、表示してください。

8 Lightbox Viewer (画像一覧の作成)

(訳注: Lightbox に適切な日本語訳が見当たりませんでした。医療用語で言えばシャウカステンを思っていただくのがよいでしょうか。この文脈での Lightbox Viewer は、ある個人のボリューム画像から断層像の一覧を作成という意味になります)

1. 加算ループ
   `while` を使って 0 から第 1 引数で指定した値まで数えるループを作ってください。その値を画面に表示してください。
2. 画像ファイルからスライスを生成する
   前のスクリプトを使って、`im3` のボリューム画像から最初の N 番目のスライスを個々の画像として生成するスクリプトを書いてください。
   スライス画像を作成するには、次のコマンドを用います。
   `slicer imagename -z -num output_num.ppm`
   ここで `num` は (0 から始まる) スライス番号を指します。
3. 画像を見る
   個々のスライス画像を `display` を使って見てみましょう。
4. スライス数の読み込み
   画像の総スライス数を自動で決めるために `fslval` を使ってください。
5. すべての画像をひとつにマージする (上級)
   `pngappend` を使って個々のスライスを lightbox スタイルの 1 ファイルを作成してみてください。

9 刺激提示ファイルを処理する

1. 実験の刺激提示サンプルファイル
   stim1.txt ファイルは event-related の実験における刺激提示タイミングの情報を含んでいます。形式は以下のとおりです:
   Event No. Condition Code Onset Time Comment
   このファイルをみて何が書かれていて、どのような書式になっているか確認してください。
2. ある条件の行を抽出する
   grep を使って、b1 を含むすべての行を抽出してください。
3. Onset Time 列を抽出する
   awk を使ってこれらの b1 が入っている行から Onset Time 列を抽出してください。
   そしてその値を変数として保存してください。
4. 列を追加する
   Onset Time 列の後に列を 2 つ追加してください。ひとつは fixed duration で値はすべての行で 30 となり、もうひとつは value で値はすべての行で 1.0 です。Onset, fixed duration, value の 3 列があると FEAT のフォーマットになります。
   ヒント: 値を for でループさせます。
   結果を stim_b1.txt として保存してください。
5. 他の条件での処理
   条件 b2 についても上と同じ事を繰り返してください。
6. Onset time を変更する (上級)
   (訳注: 原文は【TR】を変更するですが、上記の方がわかりやすいと思ったので変更しました)
   もし Onset time の情報が秒でなく、ボリューム数だったならば (訳注: stim1.txt の Onset Time はボリューム数です)、bc をつかってボリューム数を秒数に変更しましょう。TR は 3.5 秒です。

10 ファイル内容とファイル名の変更

カレントディレクトリ内にある *design.fsf* ファイルは標準的な 1st level での FEAT のセットアップファイルです。

sed を使って、ファイル中にある
/usr/local/fmrib/www/fslcourse/fsl_course_data/melodic/denoise
をすべて
/Users/fsluser/fsl_course_data/scripting
に変更し、新しいファイルとして保存しましょう。

FEAT のためにこのようにして新しいデザインファイルをつくることで、同じデザインを異なるディレクトリで異なる被験者に対して使うことができます。正しいセットアップファイルができたならば、コマンドラインで FEAT を小文字でタイプすることで FEAT を呼び出すことができます。

```
feat design.fsf
```

11 動画を作成する

このセクションは上級者向けです。ここでは回転する脳の断層像に”Maximum Intensity Projection” を重ねあわせます。

機能画像と背景画像がペアになったサンプルを 2 つ準備しています。ひとつは低解像度 (zstat2/example_func) で、スクリプトをテストするときに便利です。もうひとつは高解像度 (hr_zstat1/highres) で最終的に使用するためのものです。

出力はアニメーション GIF ですので、ウェブブラウザで見ることができます。

1. 入力パラメータ
   入力パラメータは以下になります。
   - zstat 画像
   - 背景画像
   - incremental rotation 回転角度 (フレームが変わる毎に何度回転するか)
   - z 値の閾値
   - 出力ファイル名
2. 入力パラメータの確認
   - スクリプトの入力パラメータの数が間違っていたら usage が出るようにしてください。
   - (スクリプト内で) 入力画像が実際にあるか確認するようにしてください。
   - 背景画像と zstat 画像の dimension が同じかどうかを確認してください。
3. 回転行列 (rotation matrix) の作成
   x-y 平面の中心を通る垂線を軸に回転する行列が必要となります。(0,0,0) は画像の中心ではないので、もう少し頭を使わなければなりません。
   - 画像の中心が origin になる行列を作成します
   - z 軸を中心に回転 (incremental rotation) する行列を作ります。
     - ヒント 1: bc を使って角度 (incremental angle) をラジアンに変換しなければいけません: `inc_rad=‘echo "scale=10; $inc_deg * 3.1417 /180"|bc‘ ;`
     - ヒント 2: **bc -l** を使って sin と cos を計算できます: `cosinc=‘echo "c($inc_rad)" | bc -l‘; and sininc=‘echo "s($inc_rad)"|bc -l‘;`
   - 最初の行列の逆行列を作成することで origin を画像の中心に移動します
   - これらの行列を結合し、x-y 平面の中心を通る垂線を軸に回転する行列を生成します。
4. 背景画像の生成
   MIP 画像と一緒に回転する背景画像の一断面が必要です。おすすめは正中矢状断像です。fslmaths を-roi オプションと一緒に使うことで、正中矢状断像のスライス以外はすべて値が 0 で大きさが背景画像と同一の画像をつくることができます。この画像を MIP 画像の下で回転させます。
5. 動画の作成

while ループで回転角度が 360 度に達するまで徐々に回転させます。このループの中で以下を実行する必要があります:

- fslmaths と-Xmax オプションを使って、ある角度における zstat と正中矢状断像の MIP 像を作ります (出力は 2D です)
- overlay を使って、ある閾値以上の zstat の MIP 像を正中矢状断像にレンダリングします
- slicer を使って、2D 画像の ppm 画像を作ります
- convert を使って、ppm 画像を gif 画像に変換します。
- convert_xfm を使って、次の回転のための回転行列を作成します。
- flirt を使って、zstat 画像と正中矢状断像を次の位置に回転します。
- gifmerge を使ってすべての gif ファイルをアニメーション gif にマージします。

6. 不要なファイルの削除
   作成したファイルで不要なものは削除しましょう。

付録 A　演習の回答例

(訳注: 最後の例を含め、訳者の環境 (Ubuntu 12.04) でスクリプトが動作するか確認しています。なお、オリジナルサイトに回答例がなかったものについては、訳者が例を呈示しています。)

A.1　最初のスクリプト-BET を使う

```
#!/bin/sh

# Apply bet to the image called "vol" in this directory
bet vol vol_brain
```

A.2　Echo とワイルドカード

A.2.1　すべての画像を列挙する

```
#!/bin/sh

echo *.nii.gz ../*.nii.gz
```

(訳注: もし親ディレクトリに nii.gz ファイルがなかったら、echo の結果は文字通り../*.nii.gz と表示されます)

A.2.2　一部の画像を列挙する

```
#!/bin/sh

echo vol000[24680].nii.gz
```

A.2.3　画像をマージする

```
#!/bin/sh

fslmerge -t my4Dvolume vol000[24680].nii.gz
```

A.3　コマンドラインの引数

A.3.1　コマンドラインで画像のファイル名を指定する

```
#!/bin/sh

bet $1 $1_brain
```

A.3.2　BET に引数を渡す (上級)

```
#!/bin/sh

filename=$1
shift
bet $filename ${filename}_brain $@
```

(訳注: これを使うときは、`./bet_vol im2 -f 0.4 -g 0.1` のような場合です。このとき、第 1 の引数である im2 が変数 `filename` に代入されます。その後、`shift` によって、im2 は取り除かれ、残りの-f 0.4 -g 0.1 が`$1, $2, $3, $4` にセットされます。`$@`によってこれらがまとめて引数として渡されます)

A.4　コマンドと変数

```
#!/bin/sh

sizex=‘fslval $1 pixdim1‘
sizey=‘fslval $1 pixdim2‘
sizez=‘fslval $1 pixdim3‘

echo $sizex \* $sizey \* $sizez
```

A.5　引数が妥当かを確認する

```
#!/bin/sh

if [ $# -lt 1 ] ; then
  echo "Usage: $0 <filename> [options]" ;
  exit 1 ;
fi

if [ ! -f ${1}.nii.gz ] ; then
  echo "File ${1}.nii.gz is not a regular file" ;
  exit 2 ;
fi

filename=$1
shift
bet $filename ${filename}_brain $@
```

(訳注: 上記で exit 1 と exit 2 とあります。exit は正常であれば 0 を返し、エラーは 0 以外を返します。そし

て exit の結果は変数$?におさめられます。この場合、どちらも 1 を返すと、どちらでエラーを起こしたかわからなくなりますので、片方は 1、もう片方は 2 を返すようになっていると考えられます)

A.6　画像のバッチ処理

A.6.1　拡張子を除いたファイル名 (basename) を変数として指定する

```
#!/bin/sh

for F in * ; do
  basename $f .nii.gz
done
```

A.6.2　複数の入力を指定する

```
#!/bin/sh

for F in $@ ; do
  basename $f .nii.gz
done
```

(訳注: 上記 2 つの違いは最初の文の*と$@です。$@は全引数リストの特殊変数で、$1 $2 … $n と同じ意味になります)

A.6.3　BET を複数回実行する

```
#!/bin/sh

for F in $@ ; do
  B=‘basename $F .nii.gz‘
  echo "running BET on $B"
  bet $B ${B}_brain
done
```

(訳注: 問題を言葉とおりにとらえると、ディレクトリの中にあるすべての画像ファイルに対してとなっていますので、下記が正解と思います。ここでは画像の拡張子を指定することでディレクトリのすべての画像を指定しています。上記では、自分で画像ファイルを指定しなければなりません)

```
#!/bin/sh

for F in *.nii.gz ; do
  B=‘basename $F .nii.gz‘
  echo "running BET on $B"
  bet $B ${B}_brain
```

```
done
```

A.7　ボクセルの容積を計算する

A.7.1　ボクセルの容積を計算する

```
#!/bin/sh

sizex=‘fslval $1 pixdim1‘
sizey=‘fslval $1 pixdim2‘
sizez=‘fslval $1 pixdim3‘

voxvol=‘echo "$sizex * $sizey * $sizez" | bc -l‘
echo "Voxel volume is $voxvol (in mm^3)"
```

(訳注: pixdim1 は 1 つのボクセルの x 軸方向の長さです。同様に pixdim2 は y 軸方向、pixdim3 は z 軸方向の長さを示します。ボクセルの容積はこれら 3 つの値をかけあわせればよいことになります)

A.7.2　画像の容積を計算する

```
#!/bin/sh

sizex=‘fslval $1 pixdim1‘
sizey=‘fslval $1 pixdim2‘
sizez=‘fslval $1 pixdim3‘

voxvol=‘echo "$sizex * $sizey * $sizez" | bc -l‘
echo "Voxel volume is $voxvol (in mm^3)"

nx=‘fslval $1 dim1‘
ny=‘fslval $1 dim2‘
nz=‘fslval $1 dim3‘

imvol=‘echo "$nx * $ny * $nz * $voxvol" | bc -l‘
echo "Image volume is $imvol (in mm^3)"
```

(訳注: ここで dim1 は画像の x 軸方向のボクセルの数を示します。dim2, dim3 は同様に y 軸方向、z 軸方向のボクセル数を示します。今、画像の容積を求めたいので、dim1, dim2, dim3 の値をかけあわせて、さらに先程求めたボクセルの容積 (`$voxvol`) をかければよいことになります)

A.7.3　Field of View (有効視野) を計算する

```
#!/bin/sh
```

```
sizex=‘fslval $1 pixdim1‘
sizey=‘fslval $1 pixdim2‘
sizez=‘fslval $1 pixdim3‘

voxvol=‘echo "$sizex * $sizey * $sizez" | bc -l‘
echo "Voxel volume is $voxvol (in mm^3)"

nx=‘fslval $1 dim1‘
ny=‘fslval $1 dim2‘
nz=‘fslval $1 dim3‘

imvol=‘echo "$nx * $ny * $nz * $voxvol" | bc -l‘
echo "Image volume is $imvol (in mm^3)"

fovx=‘echo "$sizex * $nx" | bc -l‘
fovy=‘echo "$sizey * $ny" | bc -l‘
fovz=‘echo "$sizez * $nz" | bc -l‘

echo "FOV is $fovx by $fovy by $fovz (in mm)"
```

(訳注: 有効視野は画像の 3 軸方向の長さを求めればよいことになります。x 軸方向だったら、x 軸のボクセルの長さ × x 軸方向のボクセルの数で長さが求まります。つまり pixdim1 と dim1 の値をかければよいことになります)

A.8　Lightbox Viewer

(訳注: ここはオリジナルに回答例がありませんでしたので、訳者が回答例を記載しています)

A.8.1　**加算ループ**

```
#!/bin/sh

if [ $# -lt 1 ] ; then
  echo "Usage: $0 <filename> [options]" ;
  exit 1 ;
fi

a=0
while [ $a -lt $1 ] ; do
 a=‘echo "$a + 1" | bc‘
```

```
done
echo $a
```

(訳注: 前半は引数があるかどうかの確認です。後半は、まず、a という変数に初期値 0 を投入しています。その後、test 文で$a が引数で示した値より小さいならば、$a に 1 を足したものを新たに a に代入しています。a が引数より大きくなるまでこのループが続きます)

A.8.2　スライス画像の生成

```
#!/bin/sh

if [ $# -lt 1 ] ; then
echo "Usage: $0 <filename> [options]" ;
exit 1 ;
fi

num=0
while [ $num -lt $1 ] ; do
 num=‘echo "$num + 1" | bc‘
 slicer im3 -z -${num} output_${num}.ppm
done
echo "The number of slices are ${num}."
```

A.8.3　画像の確認

A.8.4　スライス数の読み込み

```
slices=‘fslval $1 dim3‘
```

A.8.5　すべての画像をひとつにマージする (上級)

```
#!/bin/sh

if [ $# -lt 1 ] ; then
 echo "Usage: $0 <filename> [options]" ;
 exit 1 ;
fi

slices=‘fslval $1 dim3‘
num=0
while [ $num -lt $slices ] ; do
 num=‘echo "$num + 1" | bc‘
 output=$1_output_${num}.png
```

```
 slicer $1 -z -${num} $output
 inputarg=‘echo "$inputarg + $output"‘
done

echo "The number of slices are ${num}."

pngappend ‘echo $inputarg | sed ’s/+//’‘ $1_lightbox.png
```

(訳注: 最後の pngappend ですが、使い方は `pngappend input1 + input2 + input3 output` となっています。このため、上記の `inputarg=‘echo "$inputarg + $output"‘`で、`+ image1 + image2 + image3 ...` という変数を作り、最初の + が不要なので、sed を用いて削除しています。それが `‘echo $inputarg | sed ’s/+//’‘`の部分です。)

A.9　刺激提示ファイルを処理する

A.9.1　実験の刺激提示サンプルファイル

A.9.2　ある条件の行を抽出する

```
grep b1 stim1.txt
```

A.9.3　タイミング列を抽出する

```
b1times=‘grep b1 stim1.txt | awk ’{print $3}’‘
```

A.9.4　列を追加する

```
#!/bin/sh

filename=stim1.txt

b1times=‘grep b1 $filename | awk ’{print $3}’‘
for t in $b1times ; do
  echo "$t 30.0 1.0" >> stim1_b1.txt
done
```

A.9.5　他の条件での処理

```
#!/bin/sh

filename=stim1.txt

b1times=‘grep b1 $filename | awk ’{print $3}’‘
for t in $b1times ; do
```

```
  echo "$t 30.0 1.0" >> stim1_b1.txt
done

b2times=‘grep b2 $filename | awk ’{print $3}’‘
for t in $b2times ; do
  echo "$t 30.0 1.0" >> stim1_b2.txt
done
```

A.9.6　Onset time を変更する (上級)

- 基本的な方法

```
#!/bin/sh

filename=stim1.txt
tr=3.5

b1times=‘grep b1 $filename | awk ’{print $3}’‘
for t in $b1times ; do
  tsec=‘echo $t \* $tr | bc‘
  echo "$tsec 30.0 1.0" >> stim1_b1.txt
done

b2times=‘grep b2 $filename | awk ’{print $3}’‘
for t in $b2times ; do
  tsec=‘echo $t \* $tr | bc‘
  echo "$tsec 30.0 1.0" >> stim1_b2.txt
done
```

- エレガントな方法

```
#!/bin/sh

filename=stim1.txt
tr=3.5

for cond in b1 b2 ; do
  times=‘grep ${cond} $filename | awk ’{print $3}’‘
  for t in $times ; do
    tsec=‘echo $t \* $tr | bc‘
    echo "$tsec 30.0 1.0" >> stim1_${cond}.txt
  done
```

```
    done
```

A.10　ファイル内容とファイル名の変更

```
sed 's/\/usr\/local\/fmrib\/www\/fslcourse\/fsl_course_data\/melodic\/denoise
/\/Users\/fsluser\/fsl_course_data\/scripting/g' design.fsf > design2.fsf
```

(訳注: ディレクトリを置換する場合、sed の区切り文字を/でなく下記のように@などにすると便利です)

```
sed 's@/usr/local/fmrib/www/fslcourse/fsl_course_data/melodic/denoise/
@/Users/fsluser/fsl_course_data/scripting/g' design.fsf > design2.fsf
```

A.11　動画の作成

(訳注: オリジナル通りのソースですが、Ubuntu では 12.04 以降では gifmerge のコマンドがないなどいくつか問題があり、このままでは動作しませんでした。訳者の実力ではまだ全部デバッグできないので、オリジナルソースのまま掲載します)

```
#!/bin/sh

if [ $# -lt 5 ];then
    echo ""
   echo "Usage: spinning_mip <zstat> <struct> <incremental angle> <zthresh> <output.gif>"
    echo ""
    echo "incremental angle should be in  degrees and will be rounded to integer."
    echo ""
    echo ""
    exit 1;
fi
zstat=`basename $1 .nii.gz`;

struct=`basename $2 .nii.gz`;
incdeg=`echo "$3 / 1"|bc`;
inc=`echo "scale=10; $3 * 3.1417 /180"|bc` ;
zthresh=$4;
output=$5;

#check all the files exist
if [ ! -e ${zstat}.nii.gz ];then
    echo "$1 - Not a valid image"
    exit 2
```

```
fi

if [ ! -e ${struct}.nii.gz ];then
    echo "$2 - Not a valid image"
    exit 3
fi

# grab the dimensions of the two images
xz=`${FSLDIR}/bin/fslval $zstat dim1`;
yz=`${FSLDIR}/bin/fslval $zstat dim2`;
zz=`${FSLDIR}/bin/fslval $zstat dim3`;
xs=`${FSLDIR}/bin/fslval $struct dim1`;
ys=`${FSLDIR}/bin/fslval $struct dim2`;
zs=`${FSLDIR}/bin/fslval $struct dim3`;

# check all the dimensions match
if [ ! ${xz} -eq ${xs} -o ! ${yz} -eq ${ys} -o ! ${zz} -eq ${zs} ];then
    echo "$1 and $2 have different image dimensions - fool"
    exit 4;
fi



## create a matrix which translates image from centre to (0,0,0)
xpixdim=`${FSLDIR}/bin/fslval $struct pixdim1`;
ypixdim=`${FSLDIR}/bin/fslval $struct pixdim2`;
zpixdim=`${FSLDIR}/bin/fslval $struct pixdim3`;

xfov=`echo "$xz * $xpixdim"|bc`;
yfov=`echo "$yz * $ypixdim"|bc`;
zfov=`echo "$zz * $zpixdim"|bc`;

xtrans=`echo "$xfov / 2"|bc`;
ytrans=`echo "$yfov / 2"|bc`;
ztrans=`echo "$zfov / 2"|bc`;



echo "1 0 0 -$xtrans" > ${$}cent_to_origin.mat ;
echo "0 1 0 -$ytrans" >> ${$}cent_to_origin.mat ;
echo "0 0 1 -$ztrans" >> ${$}cent_to_origin.mat ;
```

```
echo "0 0 0 1" >> ${$}cent_to_origin.mat ;

# invert this matrix to translate back to center
${FSLDIR}/bin/convert_xfm -omat ${$}origin_to_cent.mat -inverse ${$}cent_to_origin.mat;
##create a matrix to incrementally rotate around z-axis
cosinc=‘echo "c($inc)" | bc -l‘;
sininc=‘echo "s($inc)"|bc -l‘;

echo "$cosinc -$sininc 0 0" > ${$}incrot_orig.mat;
echo "$sininc $cosinc 0 0" >>${$}incrot_orig.mat;
echo "0 0 1 0">>${$}incrot_orig.mat;
echo "0 0 0 1">>${$}incrot_orig.mat;

#concatinate the three matrices to give you your overall incremental rotation
${FSLDIR}/bin/convert_xfm  -omat ${$}tmp.mat -concat ${$}incrot_orig.mat ${$}cent_to_origin.mat;
${FSLDIR}/bin/convert_xfm -omat ${$}incrot_cent.mat -concat ${$}origin_to_cent.mat ${$}tmp.mat;



#find the mid-sagittal slice to render on and the max zstat
halfx=‘echo "$xz / 2"|bc‘
zmax=‘fslstats $zstat -R| awk ’{print $2}’‘

#initialsie things b4 the rotation loop
totinc=0;
cp ${zstat}.nii.gz ${$}rot_zstat.nii.gz;
${FSLDIR}/bin/fslmaths ${struct} -roi $halfx 1 0 $yz 0 $zz 0 1 ${$}${struct}_mid_sag
cp ${$}${struct}_mid_sag.nii.gz ${$}rot_struct.nii.gz;

echo "1 0 0 0"> ${$}tot_rot.mat
echo "0 1 0 0">> ${$}tot_rot.mat
echo "0 0 1 0">> ${$}tot_rot.mat
echo "0 0 0 1">> ${$}tot_rot.mat



### Nearly there!!
while [ $totinc -lt 360 ];do
    echo "$totinc of 360 completed"
    #The mip
    ${FSLDIR}/bin/fslmaths ${$}rot_zstat -Xmax ${$}rot_zstat_mip;
    #The background slice
```

```
    ${FSLDIR}/bin/fslmaths ${$}rot_struct -Xmax ${$}rot_struct_mip;
    #smooth it a bit to make it look nice
    ${FSLDIR}/bin/fslmaths ${$}rot_struct_mip -s 1.5 ${$}rot_struct_mip
    #Do the rendering
   ${FSLDIR}/bin/overlay 0 1 ${$}rot_struct_mip -a ${$}rot_zstat_mip $zthresh $zmax ${$}rend_mip;
    #create a ppm
    ${FSLDIR}/bin/slicer ${$}rend_mip -x 0 ${$}rend${totinc}.ppm
    #convert it to a gif
    ${FSLDIR}/bin/convert ${$}rend${totinc}.ppm ${$}rend${totinc}.gif

    #create the next rotation matrix
    #(note - we are always rotating the original images
    # to avoid successive interpolations)
   ${FSLDIR}/bin/convert_xfm -omat ${$}tot_rot.mat -concat ${$}incrot_cent.mat ${$}tot_rot.mat;

    #Do the next rotations
   ${FSLDIR}/bin/flirt -ref $zstat -in $zstat -applyxfm -init ${$}tot_rot.mat -o ${$}rot_zstat;
   ${FSLDIR}/bin/flirt -ref ${$}${struct}_mid_sag -in ${$}${struct}_mid_sag -applyxfm -init ${$}tot_ro

    #remember how  far we've rotated.
    totinc=`echo "$totinc + $incdeg"|bc`;

done
#merge all the gifs we've made
gifmerge -10 -l0 ${$}rend?.gif ${$}rend??.gif ${$}rend???.gif > $output

#remove temporary files
rm ${$}*;
```