将Melodic图形界面转换成Bash脚本

Alex / 2017-10-19 / free_learner@163.com / learning-archive.org

更新于2023-06-05，主要是文字排版上的更新，内容基本保持不变。

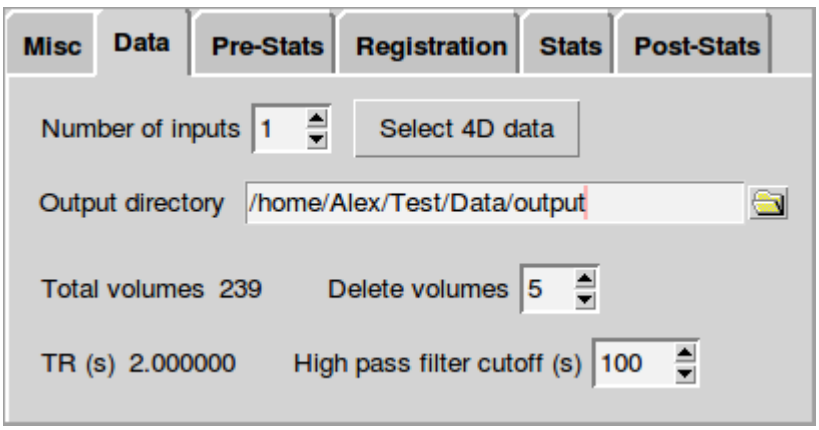
一、为什么要将图形界面转换成脚本呢?

使用Melodic的图形界面可以方便地进行功能像预处理和独立成分分析，但是图形界面的流程是固定的，随着各种新的分析工具的出现，有必要替换或增减整个流程中的某些步骤。将图形界面转换成对应的脚本就是便于根据自己的需求来调整整个分析流程。这里提供的是一种“笨”办法，如果你会Tcl语言，也许直接看源码就可以了。

二、使用Melodic图形界面分析数据

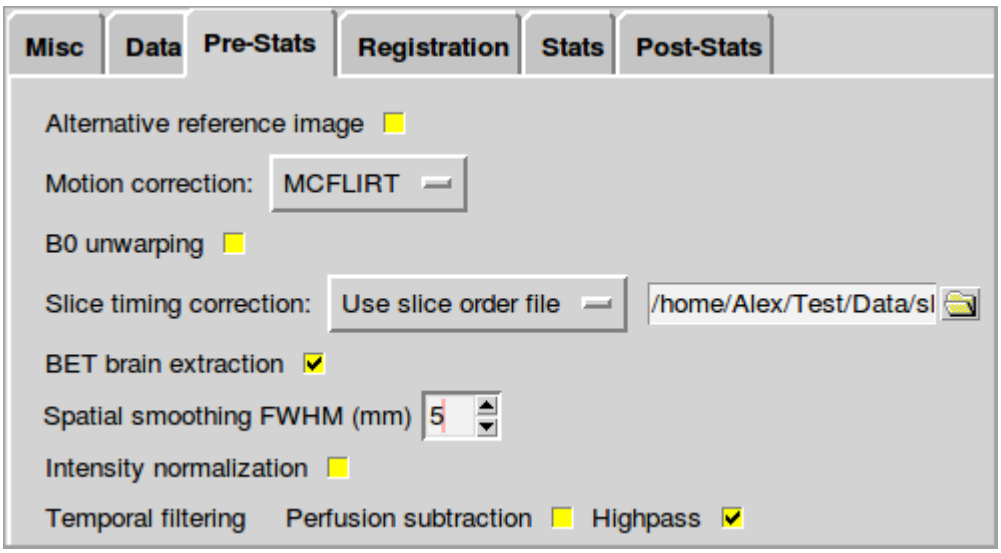
将图形界面转换成脚本的思路是先使用图形界面分析数据，通过查看生成的`report_log.html`文件和${FSLDIR}/tcl目录下的`featlib.tcl`文件来尽可能还原整个分析流程，最后用整理出的脚本再次分析数据，比较前后两次分析结果的差异。这里使用图形界面分析一个被试的数据作为例子，假设该被试有一个T1结构像并且已经去掉了颅骨，命名为`mprage_brain.nii.gz`；有一个239个时间点的静息态功能像，命名为`rest.nii.gz`；有一个包含功能像扫描顺序的文本文件（用于时间层校正），命名为`sliceOrder.txt`。

1. 输入数据



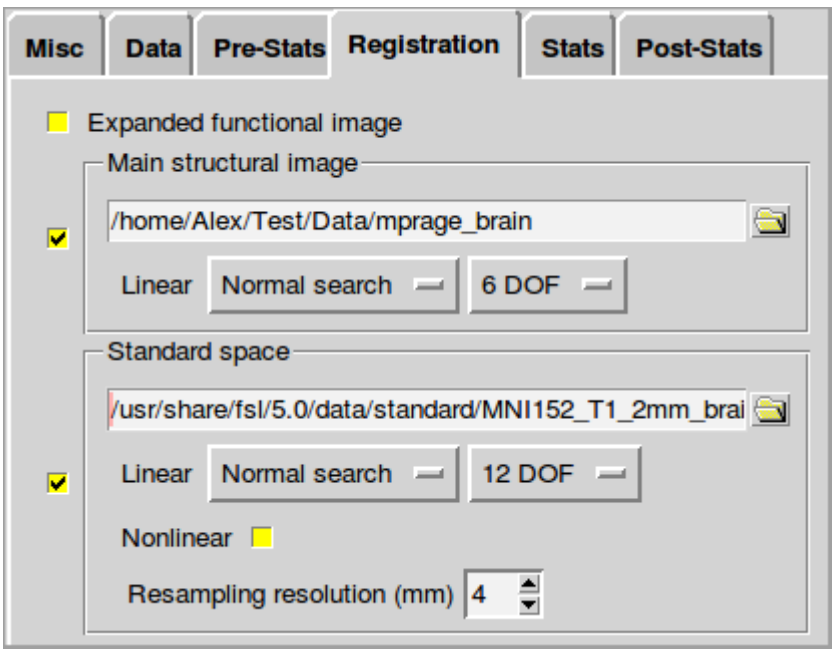
打开Melodic图形界面，通过Select 4D data选择被试的功能像`rest.nii.gz`，会自动设置Total volumes和TR；选择输出目录Output directory；设置去除前5个时间点的数据；其余为默认设置。

2. 功能像预处理



时间层校正选择Use slice order file以及相应的文本文件。之所以使用这个选项，是因为可选的扫描顺序只有三种（Regular up, Regular down以及interleaved），而实际情况往往是复杂的，所以使用自定义的文件个人认为是最方便的；其余均为默认设置。

3. 配准



勾选Main structural image以及选择去掉颅骨的结构像文件`mprage_brain.nii.gz`；为了节约时间将默认的BBR配准换成了6 DOF；其余设置不变（配准到Standard space勾选Nonlinear效果更好，同样出于简化分析的目的，没有勾选Nonlinear）。

4. 保存和运行

选择Save保存刚才的设置，下次需要重新处理可以使用Load，避免重复设置；选择Go开始处理数据。

三、使用report_log.html和featlib.tcl重建脚本

上面的分析做完以后，会在输出目录里生成一个名为`report_log.html`的文件，这个文件里保存了分析过程中用到的部分命令和一些中间结果；另外通过`featlib.tcl`这个文件来补充`report_log.html`里不清楚的部分。就是通过这两个文件，将图形界面的分析转换成脚本。为了叙述简洁，将不同的代码按照功能进行了归类。

1. 去除前5个时间点的数据

```
mkdir -p /home/Alex/Test/Data/output.ica
cd /home/Alex/Test/Data/output.ica
fslmaths /home/Alex/Test/Data/rest prefiltered_func_data -odt float
total_volumes=`fslnvols prefiltered_func_data`
total_volumes=`expr $total_volumes - 5`
fslroi prefiltered_func_data prefiltered_func_data 5 ${total_volumes}
target_vol_number=`expr $total_volumes / 2`
fslroi prefiltered_func_data example_func ${target_vol_number} 1
```

首先新建了输出目录并进入该目录；将输入文件`rest.nii.gz`转换成浮点数（float）类型并命名为`prefiltered_func_data.nii.gz`（在FSL里可以省略后缀nii.gz）；使用fslnvols计算数据有多少个时间点；使用fslroi去掉前5个时间点的数据；使用fslroi选出中间时间点的数据，用于后面的配准。如果查看`report_log`文件，只输出了Total original volumes = 239，但并不知道239是怎么得到的。所以打开`featlib.tcl`文件，搜索“Total original volumes”，不难找到如下的几行代码：

```
# check npts
set total_volumes [ exec sh -c "${FSLDIR}/bin/fslnvols $funcdata 2>
]
fsl:echo $logout "Total original volumes = $total_volumes"
```

虽然我并不懂这两行代码的准确意思，但大概可以猜到是用fslnvols输出功能像的总时间点数，并赋值给`total_volumes`，即`total_volumes=`fslnvols prefiltered_func_data``。所以`report_log`读不通的地方就需要到`featlib`里去查一下，后面采用同样的方法的地方不再重复。

2. 功能像到结构像的配准

```
mkdir -p /home/Alex/Test/Data/output.ica/reg
cd /home/Alex/Test/Data/output.ica/reg
imcp /home/Alex/Test/Data/output.ica/example_func example_func
fslmaths /home/Alex/Test/Data/mprage_brain highres
fslmaths /usr/share/fsl/5.0/data/standard/MNI152_T1_2mm_brain standard
```

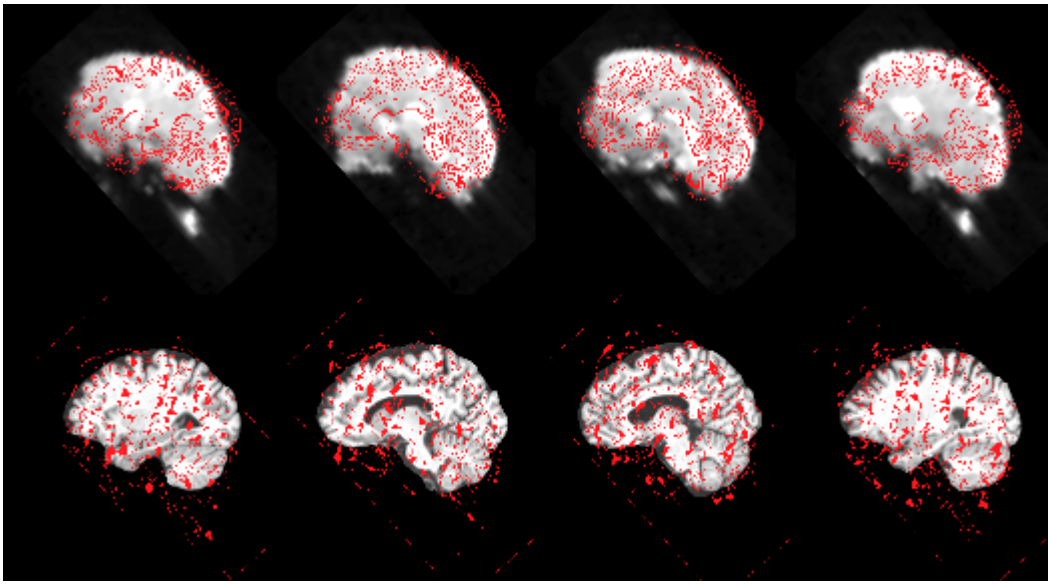
在输出目录下新建一个reg文件夹用于存放配准的结果并进入该目录；将`example_func`复制到该目录下，后面的分析都在reg文件夹下进行；将被试的结构像`mprage_brain`复制到当前目录并命名为`highres`；将MNI152的2mm模板复制到当前目录并重命名为`standard`。

```
flirt -in example_func -ref highres -out example_func2highres \
      -omat example_func2highres.mat -cost corratio -dof 6 \
      -searchrx -90 90 -searchry -90 90 -searchrz -90 90 \
      -interp trilinear
convert_xfm -inverse -omat highres2example_func.mat example_func2highres.mat
```

使用6个自由度的线性变换将`example_func`配准到`highres`，得到从`example_func`到`highres`的线性变换矩阵`example_func2highres.mat`；将得到的`example_func2highres.mat`求逆矩阵，得到从`highres`到`example_func`的线性变换矩阵`highres2example_func.mat`。

```
slicer example_func2highres highres -s 2 -x 0.35 sla.png -x 0.45 slb.png \
       -x 0.55 slc.png -x 0.65 sld.png -y 0.35 sle.png -y 0.45 slf.png \
       -y 0.55 slg.png -y 0.65 slh.png -z 0.35 sli.png -z 0.45 slj.png \
       -z 0.55 slk.png -z 0.65 sll.png
pngappend sla.png + slb.png + slc.png + sld.png + sle.png + slf.png \
       + slg.png + slh.png + sli.png + slj.png + slk.png \
       + sll.png example_func2highres1.png
slicer highres example_func2highres -s 2 -x 0.35 sla.png -x 0.45 slb.png \
       -x 0.55 slc.png -x 0.65 sld.png -y 0.35 sle.png -y 0.45 slf.png \
       -y 0.55 slg.png -y 0.65 slh.png -z 0.35 sli.png -z 0.45 slj.png \
       -z 0.55 slk.png -z 0.65 sll.png
pngappend sla.png + slb.png + slc.png + sld.png + sle.png + slf.png \
       + slg.png + slh.png + sli.png + slj.png + slk.png \
       + sll.png example_func2highres2.png
pngappend example_func2highres1.png - example_func2highres2.png \
       example_func2highres.png
rm -f sl?.png example_func2highres2.png example_func2highres1.png
```

首先以变换后的功能像在下、结构像在上，使用slicer生成XYZ三个方向的切面图，然后用pngappend横向拼接起来；然后以结构像在下、功能像在上重复上述过程；最后将两次截图纵向合在一起并删除多余文件。由下图可以看到配准出现严重错误、前后颠倒。



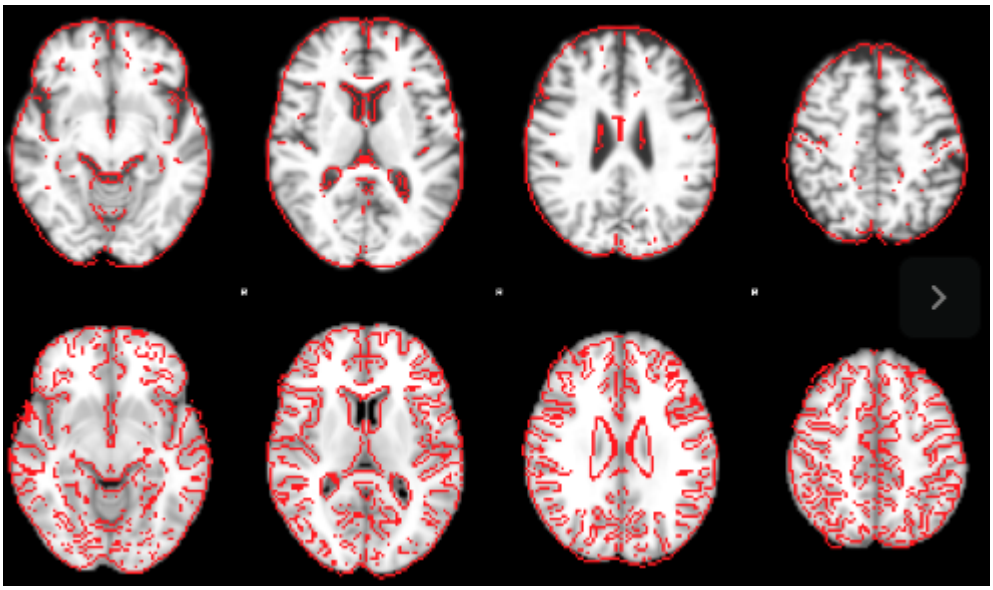
3. 结构像到MNI152标准空间的配准

```
flirt -in highres -ref standard -out highres2standard \
      -omat highres2standard.mat -cost corratio -dof 12 \
      -searchrx -90 90 -searchry -90 90 -searchrz -90 90 \
      -interp trilinear
convert_xfm -inverse -omat standard2highres.mat highres2standard.mat
```

使用12个自由度的线性变换将`highres`配准到`standard`，得到从`highres`到`standard`的线性变换矩阵`highres2standard.mat`；将得到的`highres2standard.mat`求逆矩阵，得到从`standard`到`highres`的线性变换矩阵`standard2highres.mat`。

```
slicer highres2standard standard -s 2 -x 0.35 sla.png -x 0.45 slb.png \
       -x 0.55 slc.png -x 0.65 sld.png -y 0.35 sle.png \
       -y 0.45 slf.png -y 0.55 slg.png -y 0.65 slh.png \
       -z 0.35 sli.png -z 0.45 slj.png \
       -z 0.55 slk.png -z 0.65 sll.png
pngappend sla.png + slb.png + slc.png + sld.png + sle.png + slf.png \
       + slg.png + slh.png + sli.png + slj.png + slk.png \
       + sll.png highres2standard1.png
slicer standard highres2standard -s 2 -x 0.35 sla.png -x 0.45 slb.png \
       -x 0.55 slc.png -x 0.65 sld.png -y 0.35 sle.png \
       -y 0.45 slf.png -y 0.55 slg.png -y 0.65 slh.png \
       -z 0.35 sli.png -z 0.45 slj.png \
       -z 0.55 slk.png -z 0.65 sll.png
pngappend sla.png + slb.png + slc.png + sld.png + sle.png + slf.png \
       + slg.png + slh.png + sli.png + slj.png + slk.png \
       + sll.png highres2standard2.png
pngappend highres2standard1.png - highres2standard2.png highres2standard.png
rm -f sl?.png highres2standard1.png highres2standard2.png
```

首先以变换后的结构像在下、MNI152模板在上，使用slicer生成XYZ三个方向的切面图，然后用pngappend横向拼接起来；然后以MNI152模板在下、结构像在上重复上述过程；最后将两次截图纵向合在一起并删除多余文件。



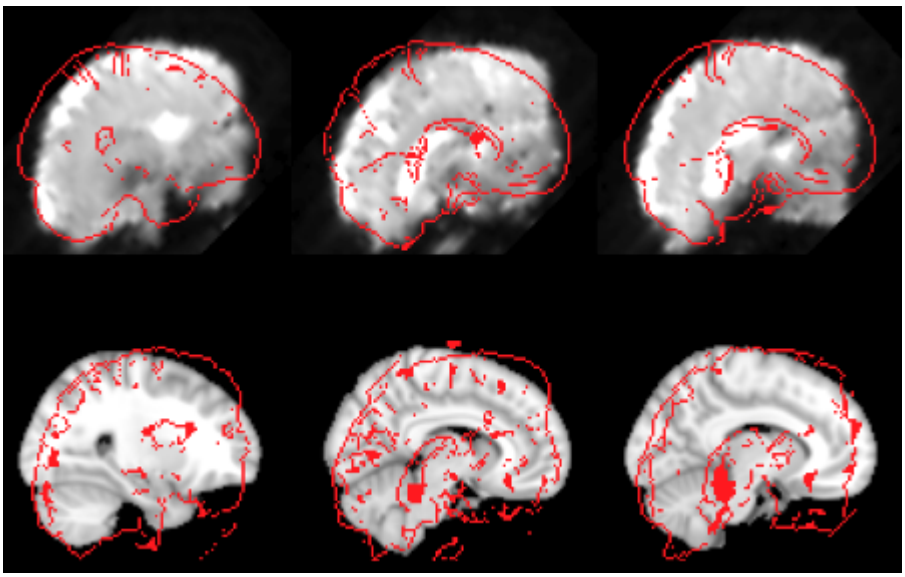
4. 功能像到MNI152标准空间的配准

```
convert_xfm -omat example_func2standard.mat \
      -concat highres2standard.mat example_func2highrest.mat
flirt -ref standard -in example_func -out example_func2standard \
      -applyxfm -init example_func2standard.mat -interp trilinear
convert_xfm -inverse -omat standard2example_func.mat example_func2standard.mat
```

将从`example_func`到`highres`的线性变换矩阵`example_func2highres.mat`和从`standard`到`highres`的线性变换矩阵`standard2highres.mat`结合起来，得到从`example_func`到`standard`的线性变换矩阵`example_func2standard.mat`；根据得到的从`example_func`到`standard`的线性变换矩阵，将`example_func`转换到到MNI152标准空间；将`example_func2standard.mat`求逆矩阵，得到从`standard`到`example_func`的线性变换矩阵`standard2example_func.mat`。

```
slicer example_func2standard standard -s 2 -x 0.35 sla.png -x 0.45 slb.png \
       -x 0.55 slc.png -x 0.65 sld.png -y 0.35 sle.png -y 0.45 slf.png \
       -y 0.55 slg.png -y 0.65 slh.png -z 0.35 sli.png -z 0.45 slj.png \
       -z 0.55 slk.png -z 0.65 sll.png
pngappend sla.png + slb.png + slc.png + sld.png + sle.png + slf.png \
       + slg.png + slh.png + sli.png + slj.png + slk.png \
       + sll.png example_func2standard1.png
slicer standard example_func2standard -s 2 -x 0.35 sla.png -x 0.45 slb.png \
       -x 0.55 slc.png -x 0.65 sld.png -y 0.35 sle.png -y 0.45 slf.png \
       -y 0.55 slg.png -y 0.65 slh.png -z 0.35 sli.png -z 0.45 slj.png \
       -z 0.55 slk.png -z 0.65 sll.png
pngappend sla.png + slb.png + slc.png + sld.png + sle.png + slf.png \
       + slg.png + slh.png + sli.png + slj.png + slk.png \
       + sll.png example_func2standard2.png
pngappend example_func2standard1.png - example_func2standard2.png \
       example_func2standard.png
rm -f sl?.png example_func2standard1.png example_func2standard2.png
```

首先以变换后的功能像在下、MNI152模板在上，使用slicer生成XYZ三个方向的切面图，然后用pngappend横向拼接起来；然后以MNI152模板在下、功能像在上重复上述过程；最后将两次截图纵向合在一起并删除多余文件。由于功能像到结构像的配准出现错误，因此从功能像到MNI152标准空间的配准也是有误的。



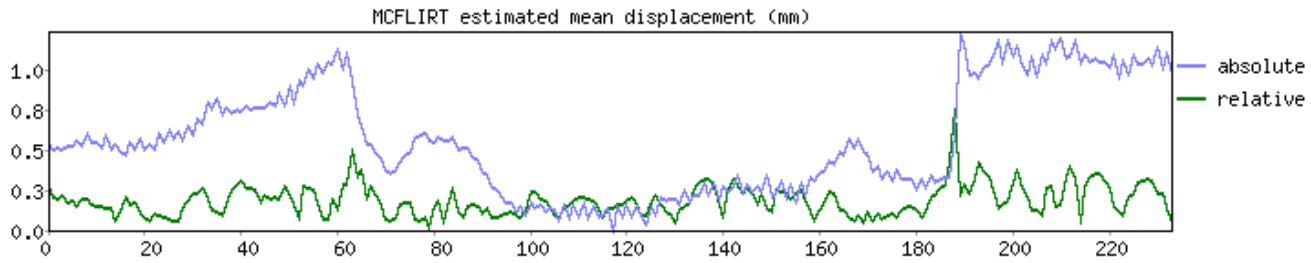
5. 头动校正

```
mcflirt -in prefiltered_func_data -out prefiltered_func_data_mcf -mats \
    -plots -reffile example_func -rmsrel -rmsabs -spline_final
mkdir -p mc
mv -f prefiltered_func_data_mcf.mat prefiltered_func_data_mcf.par \
    prefiltered_func_data_mcf_abs.rms prefiltered_func_data_mcf_abs_mean.rms \
    prefiltered_func_data_mcf_rel.rms prefiltered_func_data_mcf_rel_mean.rms mc
cd mc
```

以`example_func`为参考点进行头动校正；新建名为mc的文件夹，将上一步头动校正的结果移动到该文件夹并进入该目录。

```
fsl_tsplot -i prefiltered_func_data_mcf.par \
           -t 'MCFLIRT estimated rotations (radians)' -u 1 --start=1 \
           --finish=3 -a x,y,z -w 640 -h 144 -o rot.png
fsl_tsplot -i prefiltered_func_data_mcf.par \
           -t 'MCFLIRT estimated translations (mm)' -u 1 --start=4 --finish=6 \
           -a x,y,z -w 640 -h 144 -o trans.png
fsl_tsplot -i
prefiltered_func_data_mcf_abs.rms,prefiltered_func_data_mcf_rel.rms \
           -t 'MCFLIRT estimated mean displacement (mm)' -u 1 -w 640 -h 144 \
           -a absolute,relative -o disp.png
```

根据头动校正的结果画出每一个时间点的转动和平移大小，以及相对和绝对平均位移，用于检查头动，以其中一张图为例：



6. 时间层校正

```
cd /home/Alex/Test/Data/output.ica
TR=`fslval prefiltered_func_data_mcf pixdim4`
slicetimer -i prefiltered_func_data_mcf --out=prefiltered_func_data_st \
           -r ${TR} --ocustom=/home/Alex/Test/Data/sliceOrder.txt
```

7. 去除功能像非脑组织

```
fslmaths prefiltered_func_data_st -Tmean mean_func
bet2 mean_func mask -f 0.3 -n -m
immv mask_mask mask
fslmaths prefiltered_func_data_st -mas mask prefiltered_func_data_bet
```

得到功能像的平均图像，用于去除非脑组织，也即得到功能像的mask；使用bet2估计功能像的mask，默认会加上`_mask`的后缀，所以用immv重命名；用估计的mask去掉功能像的非脑信号。

```
int_2_98=`fslstats prefiltered_func_data_bet -p 2 -p 98`
int2=`echo ${int_2_98} | cut -d ' ' -f1`
int98=`echo ${int_2_98} | cut -d ' ' -f2`
intensity_threshold=`echo "${int2} + (${int98} - ${int2}) * 0.1" | bc`
fslmaths prefiltered_func_data_bet -thr ${intensity_threshold} \
         -Tmin -bin mask -odt char
median_intensity=`fslstats prefiltered_func_data_st -k mask -p 50`
fslmaths mask -dilF mask
fslmaths prefiltered_func_data_st -mas mask prefiltered_func_data_thresh
```

计算第2和第98百分位数，并以10%设置阈值；对每个时间点的图像去掉信号值低于阈值的体素，然后求取各图像的交集并二值化得到新的mask；计算时间层校正后的功能像的中位数，后面会用到；将前面的新的mask向外膨胀若干个体素；用新的mask去除功能像的非脑组织。

8. 空间平滑

```
fslmaths prefiltered_func_data_thresh -Tmean mean_func
FWHM=5
smoothsigma=`echo "scale=10;${FWHM}/2.355" | bc`
susan_int=`echo "scale=10; (${median_intensity} - ${int2} ) * 0.75" | bc`
susan prefiltered_func_data_thresh $susan_int $smoothsigma 3 1 1 \
      mean_func $susan_int prefiltered_func_data_smooth
fslmaths prefiltered_func_data_smooth -mas mask prefiltered_func_data_smooth
```

9. grand-mean scaling

```
normmean=10000
scaling=`echo "scale=10;${normmean}/${median_intensity}" | bc`
fslmaths prefiltered_func_data_smooth -mul $scaling \
         prefiltered_func_data_intnorm
```

将中位数值调整为10000。

10. 高通滤波

```
fslmaths prefiltered_func_data_intnorm -Tmean tempMean
hp_sigma_vol=`echo "scale=10;100/(2*${TR})" | bc`
lp_sigma_vol=-1
fslmaths prefiltered_func_data_intnorm -bptf $hp_sigma_vol $lp_sigma_vol \
         -add tempMean prefiltered_func_data_tempfilt
imrm tempMean
fslmaths prefiltered_func_data_tempfilt filtered_func_data
rm -rf prefiltered_func_data*
```

滤波后的文件重命名为`filtered_func_data`并删除多余的文件。

11. 进行ICA

```
melodic -i filtered_func_data -o filtered_func_data.ica -v \
        --nobet --bgthreshold=1 --tr=${TR} -d 0 --mmthresh=0.5 \
        --report --guireport=../../report.html
```

四、使用脚本重新分析数据

可以看到整个Melodic图形界面的分析流程包括三部分：仅涉及功能像自身的预处理（简称预处理）、配准（涉及结构像和标准空间）和后处理。预处理和配准是相互独立的，这里仅使用刚才这里的脚本重新做预处理，然后与图形界面分析的结果进行比较。

1. 整理后的预处理脚本

```
## 去除前5个数据点
mkdir -p /home/Alex/Test/Data/output.ica
cd /home/Alex/Test/Data/output.ica
fslmaths /home/Alex/Test/Data/rest prefiltered_func_data -odt float
total_volumes=`fslnvols prefiltered_func_data`
total_volumes=`expr $total_volumes - 5`
fslroi prefiltered_func_data prefiltered_func_data 5 ${total_volumes}
target_vol_number=`expr $total_volumes / 2`
fslroi prefiltered_func_data example_func ${target_vol_number} 1
## 头动校正
mcflirt -in prefiltered_func_data -out prefiltered_func_data_mcf -mats -plots -
reffile example_func -rmsrel -rmsabs -spline_final
mkdir -p mc
mv -f prefiltered_func_data_mcf.mat prefiltered_func_data_mcf.par
prefiltered_func_data_mcf_abs.rms prefiltered_func_data_mcf_abs_mean.rms
prefiltered_func_data_mcf_rel.rms prefiltered_func_data_mcf_rel_mean.rms mc
## 时间层校正
TR=`fslval prefiltered_func_data_mcf pixdim4`
slicetimer -i prefiltered_func_data_mcf --out=prefiltered_func_data_st -r ${TR}
--ocustom=/home/Alex/Test/Data/sliceOrder.txt
## 去除非脑组织
fslmaths prefiltered_func_data_st -Tmean mean_func
bet2 mean_func mask -f 0.3 -n -m
immv mask_mask mask
fslmaths prefiltered_func_data_st -mas mask prefiltered_func_data_bet
int_2_98=`fslstats prefiltered_func_data_bet -p 2 -p 98`
int2=`echo ${int_2_98} | cut -d ' ' -f1`
int98=`echo ${int_2_98} | cut -d ' ' -f2`
intensity_threshold=`echo "${int2} + (${int98} - ${int2}) * 0.1" | bc`
fslmaths prefiltered_func_data_bet -thr ${intensity_threshold} -Tmin -bin mask
-odt char
median_intensity=`fslstats prefiltered_func_data_st -k mask -p 50`
fslmaths mask -dilF mask
fslmaths prefiltered_func_data_st -mas mask prefiltered_func_data_thresh
## 空间平滑
fslmaths prefiltered_func_data_thresh -Tmean mean_func
FWHM=5
smoothsigma=`echo "scale=10;${FWHM}/2.355" | bc`
susan_int=`echo "scale=10; (${median_intensity} - ${int2} ) * 0.75" | bc`
susan  prefiltered_func_data_thresh $susan_int $smoothsigma 3 1 1 mean_func
$susan_int prefiltered_func_data_smooth
fslmaths prefiltered_func_data_smooth -mas mask prefiltered_func_data_smooth
## grand-mean scaling
normmean=10000
scaling=`echo "scale=10;${normmean}/${median_intensity}" | bc`
fslmaths prefiltered_func_data_smooth -mul $scaling
prefiltered_func_data_intnorm
## 高通滤波
fslmaths prefiltered_func_data_intnorm -Tmean tempMean
hp_sigma_vol=`echo "scale=10;100/(2*${TR})" | bc`
```

```
lp_sigma_vol=-1
fslmaths prefiltered_func_data_intnorm -bptf $hp_sigma_vol $lp_sigma_vol -add
tempMean prefiltered_func_data_tempfilt
imrm tempMean
fslmaths prefiltered_func_data_tempfilt filtered_func_data
rm -rf prefiltered_func_data*
```

2. 两次分析的比较

比较的方法就是将两次分析得到的`filtered_func_data`（即最终分析结果）在每个体素上求相关，如果两次分析结果完全一样，那么每个体素上的相关系数为1。

```
3dTcorrelate -prefix ${output}/corr_func_data.nii.gz \
             ${GUI}/filtered_func_data.nii.gz
${Scripts}/filtered_func_data.nii.gz
```

使用AFNI的3dTcorrelate计算逐个体素的相关系数。正如预期一样，两次分析的结果是完全一样的，如下图：