PANDA实现细节

Alex / 2019-09-09 / free_learner@163.com / AlexBrain.cn

更新于2023-08-19，主要是文字排版上的更新，内容基本保持不变。

通过学习PANDA的函数，了解DWI数据处理的细节。根据我的理解，PANDA处理DWI数据主要依赖FSL和DTK，其中FSL用于进行涡流校正、拟合DTI模型和TBSS、概率性纤维追踪，而DTK用于进行确定性概率追踪。PANDA通过在Matlab里调用这些软件来实现流程化处理。我感兴趣的是调用这些软件的细节，因此忽略掉Matlab的部分。我使用的PANDA版本是1.3.1，下面就是对主要函数逐个进行分析。

一、涡流校正、配准和其他预处理

`g_average`：

该函数将多个run的DWI数据平均起来，具体地，就是不同run的每个volume相加平均，不同run的每个volume对应的b-vector在每个方向上相加平均，不同run的每个volume对应的b-value相加平均。

`g_merge`：

该函数将多个run平均以后的DWI图像合并起来，成为一个4D数据。

`g_BET`：

该函数对b0图像去除颅骨。

```
bet b0 nodif_brain -f 0.25 -m -n -R
```

`g_BetT1`：

该函数对T1图像去除颅骨。

```
fslswapdim T1 LR PA IS T1_Swap
bet T1_Swap T1_Swap_Bet -f 0.5 -R
```

fslswapdim就是重新定义X/Y/Z轴所代表的方向，比如LR表示X轴，PA表示Y轴，IS表示Z轴。

`g_EDDYCURRENT`：

该函数进行涡流校正和头动校正，并相应地调整b-vector。

```
flirt -in Volume -ref B0_file -nosearch -o VolumeEddy -paddingsize 1 -omat
affine.mat
avscale affine.mat | head -5 | tail -4 > affine.txt
(Matlab) M=load('affine.txt'); RotM=M(1:3,1:3); bvec_new=RotM*bvec;
```

`g_extractB0`：

该函数提取B0图像，如果有多个B0图像，就配准到第一个B0像然后进行平均。

```
fslroi dwi.nii.gz b0_Template 0 1
flirt -in Other_b0 -ref b0_Template -nosearch -o b0_Flirt -paddingsize 1
```

思路是先提取第一个b0图像，然后将其他b0图像配准到第一个b0图像上，最后进行平均。

`g_FAtoT1`：

该函数将FA图像配准到T1图像。

```
flirt -in FA -ref T1_bet -cost corratio FAtoT1.mat
convert_xfm -omat T1toFA.mat -inverse FAtoT1.mat
applywarp -i T1_bet -r FA --premat=T1toFA.mat -o T1toFA
```

`g_T1toMNI152`：

该函数将T1图像配准到MNI152模板。

```
fsl_reg T1_file $FSLDIR/data/standard/MNI152_T1_2mm_brain out_file
```

`g_IndividualParcellated`：

该函数将标准空间的分区模板转换到个体空间。

```
applywarp -i Template -o native_Partition -r FA -w T1toMNI152_warp_inv --
postmat=T1toFA.mat --interp=nn
fslmaths FA -bin FA_binarise
fslmaths native_Partition -mul FA_binarise native_Partition
```

原理就是根据从FA->T1和T1->MNI152的变换关系，将位于MNI152空间的分区模板投射到个体空间。同时与FA的mask相乘，只保留FA非0的区域。

`g_applywarp`：

该函数根据FNIRT估计得到的转换场将图像转换到标准空间。

```
applywarp -i raw_file -o outputfile -r ref_file -w warp_file --rel
```

`raw_file`和outputfile表示变换前后的图像，`ref_file`表示参考空间图像，用于定义输出文件的分辨率信息，`warp_file`表示变换场。

`g_invwarp`：

该函数生成非线性变换的逆变换。

```
invwarp -w WarpField -r RefVolume -o InvWarpField
```

`g_JHUatlas_1mm`：

该函数根据JHU模板提取DTI指标。

```
fslstats DTImetric -k atlas_roi -M >> metric_roi.txt
```

`g_NIIcrop`：

该函数缩小FOV，同时修改相应的头信息qform和sform。

思路是通过brain mask获得大脑的边界，然后在边界的基础上向外扩展几毫米。

```
fslroi input_file crop_file Xmin Xsize Ymin Ysize Zmin Zsize
fslorient -getsform input_file
(Matlab) updated_sform=sform;
updated_sform(4)=sform(4)+sform(1)*Xmin+sform(2)*Ymin+sform(3)*Zmin;
updated_sform(8)=sform(8)+sform(5)*Xmin+sform(6)*Ymin+sform(7)*Zmin;
updated_sform(12)=sform(12)+sform(9)*Xmin+sform(10)*Ymin+sform(11)*Zmin;
fslorient -setsform new_sform crop_file
```

qform和sform表示的是从voxel空间到解剖空间（anatomical space）的映射关系，更多信息参见：

- https://nifti.nimh.nih.gov/nifti-1/documentation/nifti1fields/nifti1fields_pages/qsform.html
- https://github.com/ANTsX/ANTs/wiki/How-does-ANTs-handle-qform-and-sform-in-NIFTI-1-images%3F
- https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/Orientation%20Explained

在我所使用的FSL版本里使用fslroi会自动更新qform和sform。

`g_T1Cropped`：

该函数缩小图像的FOV，类似于`g_NIIcrop`。

`g_CalculateROIQuantity`：

该函数计算分区模板中的不同ROI的数量。

`g_DataParameters`：

该函数读取输入文件的部分参数信息并保存到一个文件。

`g_OrientationPatch`：

该函数通过改变或交换XYZ的方向调整b-vector。

`g_resample`：

该函数将图像采样到指定的分辨率。

```
fslcreatehd dim1 dim2 dim3 dim4 voxelsize1 voxelsize2 voxelsize3 pixdim4 0 0 0
datatype ref_file
flirt -in in_file -applyxfm -init $FSLDIR/etc/flirtsch/ident.mat -out out_file
-paddingsize 0 -interp trilinear -ref ref_file
```

思路是首先获得重采样前的FOV (field of view)值，即图像的边长，然后根据要求的分辨率计算出重采样后的体素个数，最后根据这些信息创建一个参考文件，使用flirt进行重采样。

`g_smooth`：

该函数使用指定宽度的平滑核对图像进行空间平滑。

```
(Matlab): sigma = kernel_size / 2.3548
fslmaths in_file -s sigma out_file
```

fslmaths要去指定平滑核（高斯函数）的标准差，需要先将半高宽转换成标准差。

`g_Split_ROI`：

该函数将一个分区模板分成多个ROI文件，看起来要求不同分区的编号是连续的，这可能不符合实际情况。

二、TBSS部分

`g_dtifit`：

该函数在每个体素拟合DTI模型并计算RD指标

```
dtifit -k data_file -m mask_file -r bvec_file -b bval_file -o out_file
fslmaths L2 -add L3 -div 2 L23m
```

`g_BeforeNormalize`：

该函数对FA图像做一些处理，为后面的配准作准备。具体地，将FA大于1的部分替换为1，对图像做一些收缩（ero），并将最后一层变为0；并生成一个FA图像的mask文件。

```
fslmaths FA_in -min 1 -ero -roi 1 $X 1 $Y 1 $Z 0 1 FA_out
fslmaths FA_out -bin FA_mask
```

X/Y/Z表示各个方向上最后一层（层数增加的方向）。

`g_BeforeNormalize_2`：

该函数对非FA的其他DTI指标做一些处理，方法是用FA的mask乘上其他指标。

```
fslmaths File_in -mul FA_mask File_out
```

`g_FAnormalize`：

该函数将FA图像配准到模板（默认是`FMRIB58_FA`）

```
fsl_reg FA FA_template FA2template -e -FA
```

`g_dismap`：

该函数生成平均的FA图像和mask图像、FA的skeleton图像和距离图像。

```
tbss_skeleton -i mean_FA -o mean_FA_skeleton
fslmaths mean_FA_skeleton -thr 0.2 -bin mean_FA_skeleton_mask
fslmaths mean_FA_mask -mul -1 -add 1 -add mean_FA_skeleton_mask
mean_FA_skeleton_mask_dst
distancemap -i mean_FA_skeleton_mask_dst -o mean_FA_skeleton_mask_dst
```

`g_2skeleton`：

该函数将根据DTI模型得到的指标（FA/MD/AD/RD）投射到skeleton上。

```
tbss_skeleton -i mean_FA -p thresh mean_FA_skeleton_mask_dst
${FSLDIR}/data/standard/LowerCingulum_1mm all_FA all_FA_skeletonised -a non_FA
```

`mean_FA`表示所有被试平均的FA图像，thresh表示FA的阈值，`mean_FA_skeleton_mask_dst`表示每个体素到skeleton的最近距离的图像，`all_FA`表示所有被试的FA图像，`all_FA_skeletonised`表示所有被试的投射到skeleton后的FA图像，`non_FA`表示其他指标（比如MD）。

`g_MergeSkeleton`：

该函数将每个被试的skeletonised后的数据合并成一个4D数据。

三、概率性纤维追踪

`g_bedpostX_preproc`：

该函数将DWI数据分成若干个slice，作为xfibres的输入。使用fslslice实现，细节不再赘述。

`g_bedpostX`：

该函数估计每个体素的纤维朝向分布。

```
xfibres --data=data_slice_0001 --mask=nodif_brain_mask_slice_0001 -b bvals -r
bvecs --forcedir --logdir=log_dir --nf=3 --fudge=1 --bi=1000 --nj=1250 --se=25
--model=2 --cnonlinear
```

对每个slice使用xfibres估计纤维朝向分布。上述参数为FSL bedpostx的默认参数，与PANDA默认参数略有不同。

`g_bedpostX_postproc`：

该函数将xfibres估计得到的每个slice的参数合并起来（ph/th/f等），使用fslmerge实现，细节不再赘述。

`g_OPDtrackNETpre`：

该函数将分区模板的每一个脑区做为seed mask，同时将除该脑区以外的其他脑区作为terminate mask。

`g_OPDtrackNET`：

该函数估计每个seed mask出发，到达其他mask的steamline的数量。

```
probtrackx --mode=seedmask -l -c 0.2 -S 1000 --steplength=0.5 -P 5000 --
stop=terminate_mask -x seed_mask --forcedir --opd --s2tastext --
targetmasks=all_seed_masks.txt -s merged -m nodif_mask_brain --dir=output_dir
```

`all_seed_masks.txt`是包含所有seed masks的绝对路径的一个文本文件，也就是将所有seed masks同时作为target masks。在FSL的官网上只有probtrackx2的介绍，看起来probtrackx和probtrackx2用法有所不同。

`g_PDtrackNETpre/g_PDtrackNET`：

这两个函数的作用类似于`g_OPDtrackNET/g_OPDtrackNETpre`，差别在于PD是估计streamline length、OPD是估计streamline count。这一点可能probtrackx和probtrackx2有所差别。

`g_track4NETpost_fdt`：

该函数计算从每个seed mask出发，到达其他mask的平均streamline count。该信息已经保存在`fdt_path.nii.gz`文件里，只需要根据不同mask计算均值。同时还会计算不同mask的体素个数。

```
fslstats fdt_path -k roi_mask -M -V
```

`g_ProbabilisticNetworkMergeResults`：

该函数构造连接概率矩阵，思路就是将两个脑区间的streamline count的总和除以从seedmask发出的总的streamline的数量（等于体素个数x每个体素的streamline数量，默认从每个体素发出5000条，考虑到方向性，所以乘以2）。从A脑区出发到达B脑区的连接不等于从B脑区出发到达A脑区的连接，因此该矩阵是一个非对称矩阵。

```
(Matlab) FDT2target = target_Voxel_matrix .* target_meanFDT_matrix;
ProbabilisticMatrix = FDT2target ./ seed_Voxel_matrix ./ 10000;
ProbabilisticMatrix(find(isnan(ProbabilisticMatrix))) = 0;
```

四、确定性纤维追踪

`g_DeterministicTracking`：

该函数使用DTK进行张量拟合和确定性纤维追踪。

```
dti_recon dwi.nii.gz dti -gm DTKBvecs -ot nii.gz
dti_tracker dti dti.trk -fact -at 35 -m nodif_brain_mask.nii.gz -m2
dti_fa.nii.gz 0.1 1
spline_filter dti.trk 0.5 dti_S.trk
```

其中`dti_recon`表示拟合张量模型，`dti_tracker`表示进行确定性纤维束追踪，`spline_filter`表示对追踪结果进行平滑。DTKBvecs表示符合DTK格式的梯度参数文件，具体构造方法：

```
(Matlab) Bvec = Bvec';
Bval = Bval';
fid = fopen(ForDTKBVector_filename, 'a+');
for j = 1:size(Bvec,1)
     fprintf(fid,'%2.4f%s%2.4f%s%2.4f%s%2.4f\n',Bvec(j,1),', ',Bvec(j,2),',
',Bvec(j,3), ', ',Bval(j,1));
end
```

其中Bvec和Bval表示原始的梯度向量和梯度值文件，DTKBvecs就是将两个文件合并成一个，并进行了转置（行变列）。

`g_DeterministicNetwork`：

该函数根据纤维束追踪的结果，计算任意两个脑区的纤维长度、纤维数量等指标。由于这部分内容网上能找到的资料较少，主要是根据自己的猜测。

`g_readTrack`：

```
function [f]=g_readTrack(filename)
fid=fopen(filename, 'rb', 'l');
f.id_string                 = fread(fid, 6, '*char')';% image dimenson
f.dim                       = fread(fid, 3, 'short')';% voxel size
f.voxel_size                = fread(fid, 3, 'float')';
f.origin                    = fread(fid, 3, 'float')';%Orign of the image
volume.
f.n_scalars                 = fread(fid, 1, 'short')';
f.scalar_name               = fread(fid, [20,10], '*char')';
f.n_properties              = fread(fid, 1, 'short')';
f.property_name             = fread(fid, [20,10], '*char')';
f.vox_to_ras                = fread(fid, [4,4], 'float')';
f.reserved                  = fread(fid, 444, '*char');
f.voxel_order               = fread(fid, 4, '*char')';
f.pad2                      = fread(fid, 4, '*char')';
f.image_orientation_patient = fread(fid, 6, 'float')';
f.pad1                      = fread(fid, 2, '*char')';
f.invert_x                  = fread(fid, 1, 'uchar');
f.invert_y                  = fread(fid, 1, 'uchar');
f.invert_z                  = fread(fid, 1, 'uchar');
f.swap_xy                   = fread(fid, 1, 'uchar');
f.swap_yz                   = fread(fid, 1, 'uchar');
f.swap_zx                   = fread(fid, 1, 'uchar');
f.nFiberNr                   = fread(fid, 1, 'int')';
f.version                   = fread(fid, 1, 'int')';
f.hdr_size                  = fread(fid, 1, 'int')';% Preallocate
f.fiber(1).nFiberLength = 0;
f.fiber(1).xyzFiberCoord = single(zeros(3, 100000));
ii=0;
 while feof(fid) == 0
    ii=ii+1;
     A= fread(fid, 1, 'int');
     if(A~=0)
    f.fiber(ii).nFiberLength=A;
    f.fiber(ii).xyzFiberCoord = fread(fid, [3+f.n_scalars
f.fiber(ii).nFiberLength], 'float=>float')';
         if f.n_properties
           f.fiber(ii).props=fread(fid,f.n_properties,'float');
         end
f.fiber(ii).xyzFiberCoord=double(f.fiber(ii).xyzFiberCoord./repmat(f.voxel_size
,f.fiber(ii).nFiberLength,1));
        f.nFiberNr=ii;
    end

end
fclose(fid);
```

该函数是对.trk文件进行信息提取，生成一个名为f的结构体，其中f.fiber存放着每条纤维的长度和坐标位置，根据坐标位置可以计算每条纤维的长度。

根据代码来看，似乎是计算的欧式距离：

```
global Length_Matrix_Fiber
for i=1:size(f.fiber,2)
    TP_1(i,1:3)=f.fiber(i).xyzFiberCoord(1,:);
    TP_2(i,1:3)=f.fiber(i).xyzFiberCoord(end,:);
    Length_Matrix_Fiber(i,1)=sum(sqrt(sum((Fibers{i}(1:end-
1,:).*repmat(Voxel_size(1,:),(length(Fibers{i})-1),1)-Fibers{i}
(2:end,:).*repmat(Voxel_size(1,:),(length(Fibers{i})-1),1)).^2,2)));
end
```

然后根据每条纤维的起点和终点坐标，给出每条纤维的前点和终点各属于什么脑区：

```
TP_1_ijk=TP_1';
TP_2_ijk=TP_2';
TP_1_ijk = floor(TP_1_ijk(1:3,:)+1);
TP_2_ijk = floor(TP_2_ijk(1:3,:)+1);
Index_TP_1= sub2ind(size(Atlas),TP_1_ijk(1,:), TP_1_ijk(2,:), TP_1_ijk(3,:));
Index_TP_2= sub2ind(size(Atlas),TP_2_ijk(1,:), TP_2_ijk(2,:), TP_2_ijk(3,:));
Index_TP_1_LabelIndex=Atlas(Index_TP_1);
Index_TP_2_LabelIndex=Atlas(Index_TP_2);
```