**Preprocessing Pipeline Manual (DTU v 1.0)**

1. Introduction 2

2. Individual Pipeline Optimization 3

i. Overview 3

ii. General Requirements 5

iii. Pipeline Inputs 5

File #1: {subject input}.txt 5

File #2: {pipeline list}.txt 7

iv. Running Individual Pipelines 9

PART-1 11

PART-2 13

PART-3 (Spatial normalization) 14

Running all three parts 15

Other switches for Run\_Pipelines.py 16

EXAMPLES 18

QUALITY CONTROL 18

4. Analysis Models 19

i. Analysis Models 19

ii. ‘split\_info’ Structure 20

iii. Task Design and Analysis Choice 21

iv. Considerations for Model Selection 22

5. Group-Level Pipeline Optimization 23

6. References 23

Appendix A: Tips for Installing AFNI and FSL Software Packages 25

Appendix B: Individual Pipeline Optimization Script Details 25

Pipeline Scripts #1 and #2 27

Pipeline Script #3: Pipeline\_STEP3.m 28

Appendix C: Spatial Normalization Script Details 30

i. Inter-Subject Alignment 30

Inter-Subject Alignment Script #1: Mask\_GroupT1template.out 30

Inter-Subject Alignment Script #2: Prep\_Transform.out 31

Inter-Subject Alignment Script #3: Run\_Transform.out 31

Appendix D: Sample Quality Control Figures 33

i. Quality Control-1 33

ii. Quality Control-2 35

# 1. Introduction

This manual provides instructions for running our software package, which optimizes preprocessing pipelines in BOLD fMRI (Blood Oxygenation Level Dependent functional MRI). The procedure implemented in this software identifies a set of preprocessing steps (“pipeline”) specific to each dataset, which optimizes NPAIRS performance metrics of prediction and reproducibility for fMRI analysis results (Strother et al., 2002). This procedure has been shown to significantly improve signal detection, reliability of brain activations, and sensitivity to brain-behaviour correlations (Churchill et al., 2012a,b ). The pipeline software can also be used to simply automatically batch-process fMRI datasets, if no appropriate analysis models are available for optimization (i.e. resting-state connectivity studies).

The pipeline scripts are coded in Matlab/Octave, and called using Python wrapper functions, developed in the Sungrid Engine (SGE) computing environment for rapid large-scale processing of datasets. They perform automated testing of preprocessing pipelines in fMRI, using a few simple input textfiles. This framework tests all possible combinations of including/excluding 11 different pipeline steps, and a variety of different analysis models. Current preprocessing options include AFNI utilities (Analysis of Functional NeuroImaging; (Cox RW, 1996)), along with a set of Matlab/Octave functions developed in-house. All steps are either widely used in the fMRI literature, or demonstrated to be important in prior studies of pipeline optimization (e.g. Tegeler et al., 1999; La Conte et al., 2003; Shaw et al., 2003; Strother et al., 2004; Zhang et al., 2009; Churchill et al., 2012a,b)

The optimization pipeline is ***analysis-driven***, as it evaluates the quality of analysis results for each pipeline, via (prediction, reproducibility) metrics. Therefore, the chosen analysis model and task contrast determines which pipelines are selected. Analysis techniques are “modular” – you can choose from a list of available models, and even try multiple ones on the same dataset/pipelines.

The pipeline software also includes a procedure for performing automated spatial normalization of subjects to an anatomical template, using FSL utilities. This enables users to compare pipeline results across subjects and datasets.

# 2. Individual Pipeline Optimization

## i. Overview

The pipeline are run from a single wrapper script:

**Run\_Pipelines.py**

which calls four scripts in sequence:

1. **pipeline\_wrapper.py** (runs preprocessing pipelines)

2. **optimization\_wrapper.py** (selects the optimal pipeline set)

3. **spatial\_normalization\_wrapper.py** (normalizes preprocessed images to a structural reference)

4**. mask\_tissue\_wrapper.py**(Generates a group mask)

These scripts call a number of functions in “scripts\_matlab”, “scripts\_other” folders.

This pipeline automatically tests a set of 11 different preprocessing algorithms, and the effects of including or excluding these steps during preprocessing of fMRI data. The steps are listed in the table below, in the order in which they are performed. In addition, see the Pipeline Workflow Figures at the end of this section. Note that multiple steps require the AFNI software package (afni.nimh.nih.gov/afni); spatial normalization of brain volumes also requires the FSL software package (fsl.fmrib.ox.ac.uk/fsl). Both are freely available, and must be installed in your computing environment in order to run the automated pipelines; see **Appendix A** for further information on the installation of these software packages.

|  |  |  |  |
| --- | --- | --- | --- |
| **PIPELINE STEP** | **DESCRIPTION** | **SOFTWARE** | **OPTIONS** |
| Motion Correction (MOTCOR) | Performs rigid-body alignment of individual fMRI volumes to a “reference” volume, in order to minimize confounds due to head motion. | AFNI’s 3dvolreg | OFF/ON |
| Censoring and Interpolation (CENSOR) | Identifies outliers in the fMRI volume timeseries, in Principal Component (PC) space. Outliers volumes are discarded and voxels values are replaced with interpolated values from neighbouring volumes. | Octave script developed in-house (Campbell et al.,2013) | OFF/ON |
| Physio. correction; external measures  (RETROICOR) | Performs retrospective correction of physiological noise, by regressing out BOLD signal that is correlated with cardiac and respiratory phase data, which are measured during the scanning session | AFNI’s 3dretroicor (Glover et al. 2000) | OFF/ON |
| Slice-Timing Correction (TIMECOR) | Performs temporal interpolation of fMRI voxel values, so that values at all brain slices correspond to the same time-point. Corrects for echo-time (TE) delays between axial slices in standard EPI BOLD. | AFNI’s 3dTshift | OFF/ON |
| Spatial Smoothing (SMOOTH) | Decreases spatial noise variance by convolving the fMRI brain volume with a 3D isotropic Gaussian kernel. This step results in some signal bias (decreased resolution) | AFNI’s 3dmerge | Full width at half max (FWHM) of the kernel can be varied |
| Temporal Detrending (DETREND) | Corrects for low-frequency noise effects, by fitting Legendre polynomials of order [0…*k*] at each voxel, and regressing them out of the timeseries | Octave script developed in-house | Maximum polynomial order *k* can be varied |
| Motion Parameter Regression (MOTREG) | Performs PCA decomposition on the head movement parameters from MOTCOR, and regresses out PCs that account for >85% of variance. This corrects for residual motion artifact. | Octave script developed in-house | OFF/ON |
| Task Covariates (TASK) | Includes the task design as a covariate when regressing out DETREND, MOTREG and GSPC1 (see below). This step protects against over-regressing task-related BOLD signal. | Octave script developed in-house | OFF/ON |
| Global Signal Removed Using PC#1 (GSPC1) | Estimates and regresses out the 1st PC of the fMRI data. It is used to control for global signal modulations, which predominantly appear in PC#1 | Octave script developed in-house | OFF/ON |
| Physio. correction; data-driven  (PHYPLUS) | Uses the data-driven PHYCAA+ algorithm to estimate and remove physiological noise components. Uses a multivariate component model (similar to PCA or ICA) to identify noise with a strong vascular component. | Octave script developed in-house (Churchill and Strother, 2013) | OFF/ON |
| Regressing out the mean time-series of a user defined area.  (CUSTOMREG) | Removes the mean time-series of a set of areas specified by a NIFTI mask file. It can be used to remove the white matter mean time-series, i.e. suggested by many studies. | Octave script developed in-house | OFF/ON |

This software package allows users to test all possible combinations of choices for these pipeline steps. For every tested pipeline, the subject data is preprocessed and analyzed, using a user-selected analysis model, which produces performance metrics of (prediction, reproducibility). The script then compares performance metrics across all tested pipelines, and identifies the pipeline that optimizes performance.

We provide results for three different pipeline optimization approaches, ordered by increasing model flexibility:

Standard, conservative pipeline (STD): this approach is the literature standard, which is included for comparison purposes. This approach applies a fixed set of all commonly-used preprocessing steps, to each fMRI dataset. This includes all steps except TASK, GSPC1, PHYPLUS, with DETREND order given by AFNI’s heuristic.

Fixed optimization (FIX): selects the single fixed set of steps, applied across all subject datasets, that gives highest average (prediction, reproducibility).

Individual optimization (IND): selects the combination of pipelines steps that optimizes (prediction, reproducibility) specific to each subject dataset.

## ii. General Requirements

1. 4D fMRI dataset, NIFTI format (i.e. with a ‘.nii’ extension)
2. T1 anatomical volume, NIFTI format

(**optional - only needed for spatial norm.**)

1. Cardiac and Respiratory data, converted into .1D design files; requires {name}.puls.1D and {name}.resp.1D file extensions (respectively). (**optional - only needed if RETROICOR is included in pipeline**)
2. A Matlab/Octave “.mat” file, containing a “split\_info” structure, which specifies parameters of the analysis model. For example, this includes task condition onsets. More details about “split\_info” fields are in ***Section 2(iii) Pipeline Inputs***. See also ***Section 4. Analysis*** for more details on the available analysis models.

**(optional – this can be omitted if you are not doing analysis/optimization)**

1. Two formatted text files (.txt) are required as input:
   1. File #1: {subject input}.txt
   2. File #2: {pipeline list}.txt

\*Curly brackets {} denote variable names that must be specified by the user

## iii. Pipeline Inputs

### File #1: {subject input}.txt

Each line in this text file corresponds to a different fMRI dataset being processed (e.g. if there are 10 subjects with 2 runs each, there will be twenty lines in the text file). The fields in each line are:

1. **IN**={ (path)/(input nifti file) }
2. **OUT**={ (path)/(output prefix) }
3. **PHYSIO**={ (path)/(physiological data prefix) }
   1. **PHYSIO** data only required if RETROICOR is included in pipeline
4. **STRUCT**={ (path)/(T1 anatomical nifti file) }
   1. **STRUCT** data only required to perform spatial normalization
5. **DROP**=[{# scans to drop at start},{# scans to drop at end}]
   1. Used to discard non-equilibrium and instructional scan volumes

**ex**. DROP=[2,2] will discard the first two volumes and last two volumes from the time series

**ex**. DROP=[0,0] does not discard any volumes

1. **TASK**={(path)/(task info matfile)}
   1. Refers to a formatted .mat file, containing a “split\_info” structure
   2. “split\_info” fields define the splitting and task structure for analysis. The elements of the split\_info matfile depend on the chosen analysis model (see ***4. Analysis*** fora detailed description of analysis models)
   3. In split\_info, task onsets are 1-relative (e.g. first volume=1), and indexed relative to the first un-dropped.

**ex.** if task 1 starts at volume 3 and DROP=[2,0], then the index for task 1 would start at volume 1.

* 1. See below for an example of “split\_info” fields, for linear discriminant analysis (LDA)

‘split\_info’ fields in LDA analysis

split\_info.idx\_cond1\_sp1 = vector of volume indices for task condition 1, for data split 1

split\_info.idx\_cond2\_sp1 = vector of volume indices for task condition 2, for data split 1

split\_info.idx\_cond1\_sp2 = vector of volume indices for task condition 1, for data split 2

split\_info.idx\_cond2\_sp2 = vector of volume indices for task condition 2, for data split 2

split\_info.drf = scalar value of range (0,1), indicating the fraction of full-date PCA subspace to keep during PCA-LDA analysis

split\_info.type = ‘block’ (this needs to be a fixed parameter)

split\_info.TR\_MSEC = integer specifying TR (acquisition rate) in milliseconds

1. **CUSTOMREG**={(path)/(nifti file)}
   1. **CUSTOMREG** the nifti file that is in the STRUCT file’s space.

Notes about {subject input}.txt:

* For **IN, OUT, PHYSIO,** list the directory path/filename for the data being preprocessed
  + Ex. IN=mydirectory/subject1\_directory/subject1\_data.nii
* Each field is separated with a space in the line. **All** fields must be included in every line (i.e. for every dataset being optimized)
* See section ***4. Analysis*** (and the header of the “run\_analyses\_wrapper.m” file) for more information on the available analysis models
* Some fields are not always necessary or available for pipeline optimization, depending on the data. For example, if you do not have physiological measures, you cannot use **PHYSIO**
  + If this is the case, just give it a “dummy” name, e.g. PHYSIO=xyz and it will run without problems

Example Text for File#1:

For 5 different subject datasets, you would produce a textfile with the 5 lines:

|  |
| --- |
| IN=my\_dir/subj1\_task.nii OUT=new\_dir/subj1 PHYSIO=my\_dir/physio/subj1 DROP=[2,1] TASK=taskInfo\_subj1.mat NOISEROI=my\_dir/roi/subj1.nii  IN=my\_dir/subj2\_task.nii OUT=new\_dir/subj2 PHYSIO=my\_dir/physio/subj2 DROP=[2,1] TASK=taskInfo\_subj2.mat NOISEROI=my\_dir/roi/subj2.nii  IN=my\_dir/subj3\_task.nii OUT=new\_dir/subj3 PHYSIO=my\_dir/physio/subj3 DROP=[2,1] TASK=taskInfo\_subj3.mat NOISEROI=my\_dir/roi/subj3.nii  IN=my\_dir/subj4\_task.nii OUT=new\_dir/subj4 PHYSIO=my\_dir/physio/subj4 DROP=[2,1] TASK=taskInfo\_subj4.mat NOISEROI=my\_dir/roi/subj4.nii  IN=my\_dir/subj5\_task.nii OUT=new\_dir/subj5 PHYSIO=my\_dir/physio/subj5 DROP=[2,1] TASK=taskInfo\_subj5.mat NOISEROI=my\_dir/roi/subj5.nii |

### File #2: {pipeline list}.txt

This is a text file listing the choices for each of the 10 pipeline steps that can be tested

|  |  |  |
| --- | --- | --- |
| Entry in File | Pipeline Step | Options |
| MOTCOR=[{options}] | Rigid-body motion correction | “0”,”1” or “0,1” |
| CENSOR=[{options}] | Censoring outlier spikes in data | “0”,”1” or “0,1” |
| RETROICOR=[{options}] | Physiological noise correction #1 | “0”,”1” or “0,1” |
| TIMECOR=[{options}] | Slice-timing correction | “0”,”1” or “0,1” |
| SMOOTH = [{options}] | Gaussian spatial smoothing | Comma-separated list of FWHM smoothing kernels (mm), e.g. “0,6,8” |
| DETREND=[{options}] | Temporal detrending | Comma-separated list of Legendre polynomial detrending order, e.g. “1,2,3” |
| MOTREG=[{options}] | Regression of head motion parameters | “0”,”1” or “0,1” |
| TASK=[{options}] | Include task design for DETREND+MOTREG | “0”,”1” or “0,1” |
| GSPC1=[{options}] | Remove PC#1 to correct for global signal | “0”,”1” or “0,1” |
| PHYPLUS=[{options}] | Physiological noise correction #2 | “0”,”1” or “0,1” |
| CUSTOMREG=[{options}] | Removing the mean time-series in an ROI specified by CUSTOMREG field in File #1. | “0”,”1” or “0,1” |

* All entries must be placed within square brackets with **no spaces**

Ex. MOTCOR=[0,1]

* For entries MOTCOR, CENSOR, TIMECOR, RETROICOR, MOTREG, TASK, PHYPLUS, GSPC1:
  + “0”= do not apply
  + “1” = apply
  + “0,1” = try with AND without
* Note that DETREND, MOTREG and GSPC1 are performed as part of a general linear model (GLM)
  + If TASK=1, we include the task regressor to control against over-regression of the task signal

Example Text for File #2

If you wanted to test MOTCOR, CENSOR, and DETREND (orders 0 to 3), and leave all other steps OFF (with a spatial smoothing scale of 6mm), you would have a text file with the following lines:

|  |
| --- |
| MOTCOR=[0,1]  CENSOR=[0,1]  RETROICOR=[0]  TIMECOR=[0]  SMOOTH=[6]  DETREND=[0,1,2,3]  MOTREG=[0]  TASK=[0]  GSPC1=[0]  PHYPLUS=[0]  CUSTOMREG=[1] |

## iv. Running Individual Pipelines

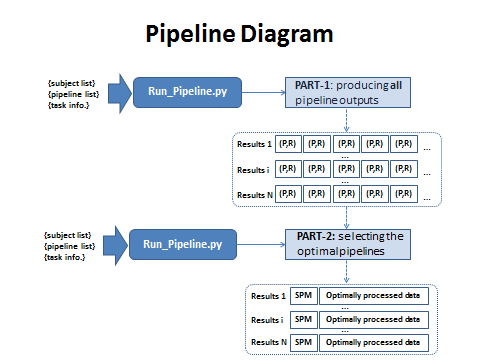
Pipeline optimization should be run using the high-performance Sun Grid Engine (SGE). This framework allows individual subjects (specified in the {subject input}.txt file) to be submitted as individual jobs to a set of computing clusters, greatly accelerating computational speed. This is all done by running the wrapper “**Run\_Pipelines.py**”. The command line structure is defined below, along with important outputs (see Appendix for further output details).

**Preparatory details**:

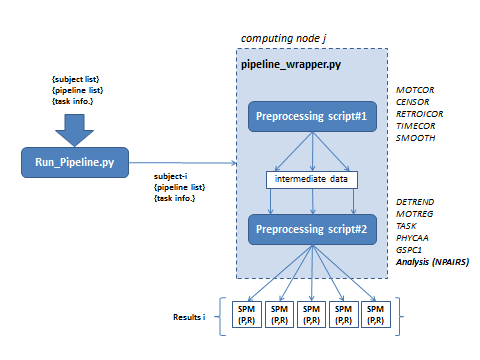
* All pipelines are run from command lines in the terminal
* Users should be logged into one of cluster servers (e.g. grid01, grid02…); see <https://itswiki.compute.dtu.dk/index.php/Main_Page> for further details
* Make sure that the pipeline software “Run\_Pipelines.py” and folders “scripts\_matlab” and “scripts\_other” are in path
* Check that both AFNI and FSL software packages are in your path
* The pipeline input textfiles {subject input}.txt and {pipeline list}.txt should be in your current working directory

When the scripts are run on the SGE system, each line in the {subject input}.txt file (i.e. each subject dataset) is extracted into a separate text-file and stored in a “Submission\_temp” directory; it is then submitted as a job via the “qsub” command. Each job has an associated number, in order to track progress of scripts. These job numbers appear on the screen as soon as the script is run. Once the system begins to process the specific job, a text file is created in the working directory, labeled with the wrapper name and job number, which tracks the job. Once the job is completed, the text file will have a time (in seconds) at the end, which is the length of time it took to complete the job. If there is an issue with output files, you can go back to these text files to see if there was an error during the processing.

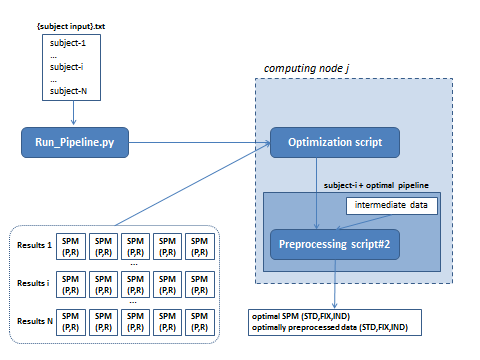
Typing “qstat –f” in the command line will show jobs that are currently being processed as well as pending jobs, identified by job number.



**Figure 1: schematic of overall pipeline optimization process**



**Figure 2: more detailed schematic of pipeline part-1 (running pipelines)**



**Figure 3: more detailed schematic of pipeline part-2 (optimizing pipelines)**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### PART-1

**INPUT:** runs all preprocessing pipelines specified in {pipeline list}.txt and the chosen analysis model, for all datasets in the {subject input}.txt file. Produces output brain maps (SPMs; Statistical Parametric Maps) and NPAIRS performance metrics. The syntax is:

**./Run\_Pipelines.py –p 1 –i {subject input}.txt –c {pipeline list.txt} –a {analysis model} [other arguments]**

Where:

1. **“–p 1”** specifies that you are running part-1
2. **“-i {subject input}.txt”** specifies the input text file
3. **“-c {pipeline list}.txt”** gives the list of chosen preprocessing steps
4. **“-a {analysis model}”** specifies the chosen analysis approach (e.g. “-a LDA” performs a linear discriminant analysis).

**If you don’t want to (or can’t) do analysis, type “-a NONE”**

1. **Other aguments:** depending on the analysis model used (-a MODEL) some specific arguments have to be used (See Table 2).

**OUTPUT**: This will depend on your analysis choice:

* If no analysis model is chosen (“-a NONE”), you **cannot** do pipeline optimization after running PART-1. Instead, the script will output all subject datasets, preprocessed with each chosen pipeline. **WARNING**: if you choose many different pipeline options, this will produce a *lot* of data!!

For each dataset defined in your {subject input}.txt file, you will get a set of preprocessed versions of that dataset, with the following syntax:

(path)/(output\_prefix)\_ m[0/1]c[0/1]p[0/1]t[0/1]s[fwhm]d[ord]r[0/1]x[0/1]g[0/1/2/3]y[0/1].nii

Where choices in pipeline steps are denoted by single letter+number combinations: {m = MOTREG, c = CENSOR, p = RETROICOR, t = TIMECOR, s = SMOOTH, d = DETREND, r = MOTREG, x = TASK, g = GSPC1+CUSTOMREG, y = PHYCAA+}. In this notation, [0/1]=step turned off/on, [fwhm] = smoothing kernel, [ord] =detrending order, and for g[0/1/2/3], {0=OFF, 1=GSPC1, 2=CUSTOMREG, 3=GSPC1+CUSTOMREG). For example, if you specified in {pipelines}.txt:

MOTCOR=[0,1]

CENSOR=[1]

RETROICOR=[0]

TIMECOR=[1]

SMOOTH=[6]

DETREND=[1,2,3]

MOTREG=[1]

TASK=[0]

GSPC1=[1]

PHYPLUS=[0]

CUSTOMREG=[0]

and the first line of {subject input}.txt had output OUT=new\_dir/subj1. This would create the following preprocessed NIFTI datasets for that subject:

/new\_dir/subj1\_ m0c1p0t1s6d1r1x0g1y1.nii

/new\_dir/subj1\_ m0c1p0t1s6d2r1x0g1y1.nii

/new\_dir/subj1\_ m0c1p0t1s6d3r1x0g1y1.nii

/new\_dir/subj1\_ m1c1p0t1s6d1r1x0g1y1.nii

/new\_dir/subj1\_ m1c1p0t1s6d2r1x0g1y1.nii

/new\_dir/subj1\_ m1c1p0t1s6d3r1x0g1y1.nii

* If an analysis model is specified, you will instead get three “.mat” files, summarizing analysis results of pipelines. For each line of {subject input}.txt, with entry OUT={path}/{name}, we produce

1. **Brain maps** ({path}/results1\_spms\_{name}.mat). Contains the following:

IMAGE\_set = activation maps computed for each pipeline. This is a (pipelines x 1) cell array, where each entry is a (voxels x components) matrix of images

2. **Temporal measures** ({path}/results2\_temp\_{name}.mat). Contains the following:

TEMP\_set = BOLD timeseries associated with the activations in IMAGE\_set. This is a (pipelines x 1) cell array, where each entry is a (time x components) timeseries matrix

3. **Performance metrics** ({path}/results3\_stats\_{name}.mat). Contains the following:

METRIC\_set = cell array with one entry per pipeline, providing performance metrics for a given analysis model. e.g. the LDA model produces metrics of prediction (P), reproducibility (R) and negative of distance (Dneg) from (P=1,R=1). Thus for pipeline q, we have entries: METRIC\_set{q}.P, METRIC\_set{q}.R, METRIC\_set{q}.Dneg

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### PART-2

**INPUT:** identifies the optimal pipelines based on PART-1 output, and records SPMs, performance metrics and optimally preprocessed data. The syntax is:

**./Run\_Pipelines.py –p 2 –i {subject input}.txt –m {optimization metric} –o {output prefix}**

Where:

1. **“–p 2”** specifies that you are running part-2,
2. **“-i {subject input}.txt”** specifies the input text file
3. **“-m {optimization metric}”** chooses the optimization criterion which is being maximized; multiple performance metrics are often available \*
4. **“-o {output prefix}”** specifies the path+prefix for summary output

data (in .mat file format).

\* **NOTE**: the “-m {optimization metric}” options depends on the analysis model. However, there will generally be three available options: “R” (reproducibility), “P” (prediction) and “Dneg” (negative of Euclidean distance from perfect (P=1,R=1)). If possible, we usually recommend using “-m Dneg”, since this metric ensures that pipelines optimize both prediction **and** reproducibility values.

**OUTPUT**: a single .mat file and set of NIFTI volumes.

1. **Optimization results** (mat file) ({out\_prefix}'pipeline\_opt\_',{optimize\_metric},'\_results.mat') , contains the following:

Each cell entry has 3 fields of "std", "fix", "ind", for the 3 optimization methods. Cells correspond to individual subjects, listed in {subject input}.txt.

SPM\_opt = set of SPMs, for each optimized subject pipeline

TEMP\_opt = set of BOLD timeseries, for each optimized subject pipeline

METRIC\_opt = set of performance metrics for each optimized subject pipeline

There is also a "pipeline\_sets" structure with the following fields:

pipeline\_sets.(std/fix) = (pipelines x 10) a matrix which specifies the pipeline steps used for fixed and standard optimization methods

pipeline\_sets.ind = (subject x pipeline step) matrix of design vectors specifying the pipeline steps used in each individually optimized subject

2. **Optimal pipeline SPMs** (NIFTI volumes)

a) Produces 4D nifti file of three optimized pipeline SPMs, ordered (STD, FIX,

IND) along the 4th dimension (time-axis):

{path},'/matfiles/niftis\_’,{name},’/Images\_’,{name},’\_’,{optimize\_metric},’\_Std\_Fix\_Ind.nii'

b) It also produces optimally preprocessed time-series data, for (STD, FIX, IND) pipelines:

{path},'/matfiles/niftis\_',{prefix},'/Preprocessed\_opt\_',{optimize\_metric},'\_',{prefix},'\_STD.nii'

{path},'/matfiles/niftis\_',{prefix},'/Preprocessed\_opt\_',{optimize\_metric},'\_',{prefix},'\_FIX.nii'

{path},'/matfiles/niftis\_',{prefix},'/Preprocessed\_opt\_',{optimize\_metric},'\_',{prefix},'\_IND.nii'

\* Refer to the Appendix for further details on the underlying optimization scripts, and other pipeline outputs. If the SGE is unavailable, the pipelines can be run directly from command line, described in **Appendix B**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### PART-3 (Spatial normalization)

Spatial normalization transforms all subject data into the same template space; this is required in order to compare brain activation patterns across subjects. As with pipeline optimization, spatial normalization should be run on the high-performance Sungrid Engine (SGE) to accelerate computational speed. The command line structure is defined below, along with important outputs (see **Appendix C** for further output details).

**./Run\_Pipelines.py –p 3 –i {subject input}.txt –r { reference volume } –v {voxel size} –o {output prefix}**

Where:

1. **“–p 1”** specifies that you are running part-1
2. **“-i {subject input}.txt”** specifies the input text file
3. **“-r {reference volume}”** gives path+name of the T1 anatomical reference template
4. **“-v {voxel size}”** (Optional) specify the voxel size for the spatially normalized nifti files. Example: -v 4.0 means isotropic voxels size of 4.0mm3. –v ‘4.0 3.0 5.0’ means voxel size equal to 4.0x3.0x5.0mm. If a file is passed as a parameter (e.g. –v /example/a.nii) the program will set the output voxel size and FOV equal to those in the parameter file. If –v option is not used, then the program leaves the voxel size intact.
5. **“-o {output prefix}”** gives path+name of the consensus brain mask.

**OUTPUT**: per line of {subject input}.txt, with field OUT={path}/{name}, we produce:

A “spat\_norm” folder containing transformation matrices, spatially

normalized T1 anatomicals, and brain masks in new template space

Spatially normalized SPMs and preprocessed data; all spatially normalized

NIFTI volumes have “\_sNorm.nii” appended to them; this includes all

OUTPUT data described in Run\_Pipelines part-2 (above).

Also, weproduce a set of mask and brain tissue results:

{output prefix}.nii: a consensus brain mask

{output prefix}\_Mean\_NN\_WM.nii: a set of mean tissue masks, ordered

along the 4th dimension (time): mean EPI, non-neuronal (vascular), and

white matter.

{output prefix},'\_outputs.mat': a .mat file storing statistics on the

quality of spatial normalization. Used as input in Quality Control 2 (described above, in the Running Individual Pipelines section).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### Running all three parts

It is possible to submit all three parts in one command line. Run\_Pipeline.py will submit the all three parts together. The part 2 jobs will be on hold until the part 1 jobs are finished, and the part 3 jobs will be on hold until the part 2 jobs are completed. To submit all jobs, you should not specify the part number when running Run\_Pipelines.py, and pass all necessary parameters for all three steps. The command line structure is defined below:

**./Run\_Pipelines.py –i {subject input}.txt –c {pipeline list.txt} –a {analysis model} –m {optimization metric} –o {output prefix} –r { reference volume } –v {voxel size} [other arguments]**

|  |
| --- |
| \*NOTE: If {reference volume} is not specified with switch –r, then only part 1, and 2 are submitted. |

### Other switches for Run\_Pipelines.py

**TABLE 1. General switches**

|  |  |
| --- | --- |
| **Switch** | **Description** |
| **-h, --help** | **show help message and exit** |
| **-p PART**  **(--part PART)** | **select pipeline optimization step, 1: part 1, 2: part 2, 3: Spatial normalization, 0: All three steps (default 0)** |
| **-i FILE.txt,**  **(--inputdata=FILE.txt)** | **FILE.txt contains the input and output data paths** |
| **-c PIPELINE,**  **(--pipeline PIPELINE)** | **The preprocessing steps** |
| **-a ANALYSIS,**  **(--analysis=ANALYSIS)** | **determines analysis model (LDA, GNB, GLM, erCVA, erGNB, erGLM, SCONN,gPCA)** |
| **-m METRIC,**  **(--metric=METRIC)** | **Optimization metric** |
| **-o OUT\_PREFIX,**  **(--out\_prefix=OUT\_PREFIX)** | **output file prefix** |
| **--dospnormfirst** | **(optional) First normalize the data to a reference (specified by switch -r), then perform the preprocessing optimization (Not recommended).** |
| **--contrast=CONTRAST** | **(optional) desired task contrast, it is necessary when more than two contrasts are defined in the split info files, syntax: --contrast "CON1-CON2,CON2-CON3", where CON1, CON2 are condition names (The condition number may be used instead, e.g. –contrast 1-2,2-3 ). Default is --contrast 1-2.**  **(See Subsection 4.ii for more details)** |
| **-r REFERENCE,**  **(--reference=REFERENCE)** | **Structural reference which is used in the spatial normalization step, i.e. -p,--part=3** |
| **-n NUMCORES,**  **(--numcores=NUMCORES)** | **(optional) number of threads used for each job (not**  **effective for some SGE systems)** |
| **-q QUEUE,**  **( --queue=QUEUE)** | **(optional) SGE queue name, default is bigmem\_16.q**  **(For HPCVL SGE –q abaqus.q has to be used).** |
| **-k KEEPMEAN,**  **(--keepmean=KEEPMEAN)** | **(optional) determines whether the ouput nifti files**  **contain the mean scan (Default KEEPMEAN=0, i.e. removes the mean scan)** |
| **-v VOXELSIZE,**  **(--voxelsize=VOXELSIZE)** | **(optional) determines the output voxel size of nifti file** |
| **-e ENVIRONMENT** | **(optional) determine which software to use to run the**  **code: matlab or octave(default)** |

**TABLE 2. Model specific switches**

|  |  |  |
| --- | --- | --- |
|  | **Switch** | **Descriptions** |
| **GLM** | **--convolve VALUE** | **VALUE=Binary value, for whether design matrix should be convolved with a standard SPMG1 HRF. 0 = do not convolve and 1 = perform convolution** |
| **GNB** | **--decision\_model MODEL** | **MODEL= string specifying type of decision boundary. Either: ‘linear’ for a pooled covariance model or ‘nonlinear’ for class-specific covariances** |
| **LDA** | **--drf FRACTION** | **FRACTION=Scalar value of range (0,1), indicating the fraction of full-date PCA subspace to keep during PCA-LDA analysis. A drf of 0.3 is recommended as it has been found to be optimal in previous studies.** |
| **erGLM** |  | **This model does not have any specific switch** |
| **erGNB** | **--Nsplit NUMBER** | **NUMBER=Number of equal sized splits to break the data into, in order to perform time-locked averaging. Must be at least 2, with even numbers >=4, recommended to obtain robust covariance estimates** |
| **--WIND SIZE** | **SIZE = window size to average on, in TR (usually in range 6-10 TR)** |
| **erCVA** | **--Nsplit NUMBER** | **NUMBER= number of equal sized splits to break the data into, in order to perform time-locked averaging. Must be at least 2, with even numbers >=4, recommended to obtain robust covariance estimates** |
| **--WIND SIZE** | **SIZE = window size to average on, in TR (usually in range 6-10 TR)** |
| **--drf FRACTION** | **FRACTION= scalar value of range (0,1), indicating the fraction of full-date PCA subspace to keep during PCA-LDA analysis. A drf of 0.3 is recommended as it has been found to be optimal in previous studies.** |
| **--subspace COMP** | **COMP = string specifying either: 'onecomp' = only optimize on CV#1 or 'multicomp' = optimize on full multidimensional subspace** |
| **SCONN** | **--spm FORMAT** | **FORMAT =string specifying format of output SPM. Options include ‘corr’ (map of voxelwise seed correlations) or ‘zscore’ (Z-scored map of reproducible correlation values)** |

**TABLE 3. Switches for group preprocessing optimization**

|  |  |
| --- | --- |
| **--autodetect** | **GROUP preprocessing pipeline optimization. Automatically detects subjects in the input file and optimizes for each subject independently, the lines in input files that have same structural image (STRUCT) and same output directory (OUT) are considered as one subject (See Section 5 for more details)** |
| **--N\_resample NUMBER** | **Only used when the group optimization is performed.**  **NUMBER= scalar indicating the number of resampling splits to perform (and average over)** |

### EXAMPLES

(EX.1) Let us assume we have some resting-state dataset with no over task design. Our goal is just to preprocess the data specified in “input.txt” with a few different pipeline combinations, specified in “param.txt”. The syntax for generating these preprocessed data is:

**./Run\_Pipelines.py –i /example/input.txt –c /example/param.txt –a NONE–r /example/MNI152\_1mm.nii –v 4.0**

(EX.2) Let assume we have four conditions in the data ATT, DMS, PMT, and RT. The goal is to run pipeline optimization with LDA for two task contrasts: 1) ATT and DMS vs baseline, 2) DMS vs RT. The optimization metric is Dneg (See Performance metrics subsection). The syntax for submitting all three parts is as follow:

**./Run\_Pipelines.py –i /example/input.txt –c /example/param.txt –a LDA --drf 0.3 –m Dneg –o preprocessed –r /example/MNI152\_1mm.nii –v 4.0 --contrast ‘ATT+DMS-baseline,DMS-RT’**

Because we used LDA model, according to the Table 2, we have to define drf.

The pipeline will optimize for both ATT and DMS vs baseline, and DMS vs RT task contrasts.

(EX.3) Let assume the goal is to optimizing the preprocessing of an event-related dataset using the model erCVA. Also, we do not want to run the spatial normalization part (part 3). The syntax is as follow:

**./Run\_Pipelines.py –i /example/input.txt –c /example/param.txt –a erCVA --Nsplit 4 --WIND 6 --drf 0.3 --subspace onecomp –m Dneg –o preprocessed**

We did not specify the reference image using –r. Then the program only submits the jobs related to Parts 1, and 2. The model erCVA needs four arguments: N\_split, WIND, drf, and subspace.

Here, we did not specify --contrast. Consequently the program optimizes for the condition 1 vs condition 2.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### QUALITY CONTROL

There are two additional scripts that produce diagnostic plots summarizing results of PART-1 and PART-2. They are currently only available to run in a Matlab environment, and **cannot** be directly run on the SGE, as plotting tools are not available in Octave. See **Appendix D** for examples of QC output figures.

**Quality Control-1**: to be run after completing PART-1. Produces output plots showing: (1) head motion stats, (2) potential SPM artifact (e.g. motion, global signal, white matter), (3) the sensitivity of performance metrics to pipeline choice, and (4) preprocessing steps that have the greatest impact on results.

Syntax:

**Pipeline\_QC1**( {subject input}.txt, save\_prefix )

Output is saved to “QC1\_images” local directory

**Quality Control-2**: to be run after completing PART-2 ***and*** spatial normalization (see **3. Spatial Normalization of fMRI Data** below). Produces output plots showing: (1) optimal pipeline choices, (2) optimal performance metrics, (3) statistics on quality of spatial normalization, (4) between-subject SPM similarity, and (5) group-level PCA results for each pipeline.

Syntax:

**Pipeline\_QC2**({subject input}.txt, optimize\_matfile\_name, newmaskname,save\_prefix )

Where:

1. **optimize\_matfile\_name:** string specifying the full path and name of the summary optimization file created during PART-2 of pipeline optimization
2. **newmaskname:** string specifying the full path and name of the new group-level consensus mask (produced during spatial normalization, described in the following section)
3. **save\_prefix:** string specifying prefix for saved plots

Output is saved to “QC\_images” local directory

# 4. Analysis Models

The optimization pipelines are analysis-driven; model choice determines what brain signal you are able to measure, as well as what pipeline choices optimize the detection of this signal. The analysis models that are currently available are listed below. Models that are shaded provide both prediction and reproducibility metrics; other models only estimate reproducibility, and should be interpreted with caution.

## i. Analysis Models

|  |  |  |
| --- | --- | --- |
| Acronym | Name | Description |
| GLM | Standard General Linear Model | Basic univariate analysis for pre-defined stimulus paradigm |
| GNB | Gaussian Naïve Bayes | Predictive univariate analysis for block design, with 2 task conditions |
| LDA | Linear Discriminant Analysis | Predictive multivariate analysis for block design, with 2 task conditions |
| erGLM | Event-related General Linear Model with HRF estimation | Basic univariate analysis for event-related task design; models a single hemodynamic response function (HRF) |
| erGNB | Event-related Gaussian Naïve Bayes | Predictive univariate analysis for event-related design, with 1 task type onset |
| erCVA | Event-related Canonical Variates Analysis | Predictive multivariate analysis for event-related design, with 1 task type onset |
| SCONN | Seed-based Connectivity analysis | Voxel-wise correlations with seed Region of Interest (ROI) |
| gPCA | Principal Component generalization | Predictive generalization of covariance defined by PCA model |

## ii. ‘split\_info’ Structure

The {subject input}.txt file has a field called ‘TASK’ which refers to the a matfile containing ‘split\_info’ structure. The same matfile can be used for all subject datasets, if the analysis fields are the same across datasets. If analysis fields vary across subjects (e.g. randomized stimulus onsets), different matfiles must be used.

Any fields in the ‘split\_info” structure that refer to a volume index must **start from 0** before any scans that have been dropped using the field DROP in the input file.

The required fields are listed below:

split\_info.unit : (sec/msec/TR). Specifies the unit of the onset times which can be sec, msec, or TR. If not specified unit = TR.

split\_info.TR\_MSEC : integer specifying TR (acquisition rate) in milliseconds

Defining task conditions:

Task condition 1:

split\_info.cond(1). onsetlist : vector list of integers representing stimulus onset times in TR,

split\_info.cond(1). blklength : vector list of integers representing stimulus lengths in TR,

split\_info.cond(1). name : task condition name

Task condition 2:

split\_info.cond(2). onsetlist : vector list of integers representing stimulus onset times in TR,

split\_info.cond(2). blklength : vector list of integers representing stimulus lengths in TR,

split\_info.cond(2). name : task condition name

Task condition 3:

split\_info.cond(3). onsetlist : vector list of integers representing stimulus onset times in TR,

split\_info.cond(3). blklength : vector list of integers representing stimulus lengths in TR,

split\_info.cond(3). name : task condition name

...

Other fields:

split\_info.seed\_name : Necessary when the SCONN model is used. string giving location of a seed ROI volume. This must be a 3D binary volume of same dimensions as input fMRI data. Voxels in ROI must equal ‘1’ and non-ROI voxels must equal ‘0’

\*NOTES

1. Multiple Conditions: It is possible to define multiple conditions.
2. Backward Compatibility: The previous version split\_info structure can be used. If the model specific switches are not used (TABLE 2), then the program look the split\_info for the value for those switches (parameters).
3. Condition Names: Using name for conditions is optional, but if condition names are not defined, condition numbers have to be used to refer to the conditions.
4. BASELINE condition: The name ‘*baseline*’ refers to resting state (or fixation). If no condition with name *baseline* is defined, then the scans that do not belong to the any of the defined conditions will be considered as *baseline* condition. To prevent rank deficiency of the design matrix, the baseline condition will not be modeled by a regressor in the GLM design matrix.
5. EVENT related data: If the data is event-related, blklength 0 should be used for the onsets.
6. SCONN analysis: If the seed-based analysis SCONN is used, the file seed\_name has to be defined.

## iii. Task Design and Analysis Choice

One of the first considerations when choosing a model is the form of the stimulus

that you are trying to measure. This depends entirely on your experimental design.

Broadly speaking, there are three categories of fMRI task design:

1. **Block design:**

A fixed stimulus is presented for an extended duration, e.g. longer than the time-to-peak for a standard HRF ( around 6-10 sec.). A block design analysis typically looks for difference in “mean” signal between different experimental stimulus blocks. Two-condition analysis is most common because it is easiest to characterize as a simple difference. Multi-condition analyses are rarer, due to their relative complexity. **The pipeline framework currently only provides full optimization capabilities for 2-condition block designs.**

Available Models:

* + 1. GLM (general linear model)\*
    2. GNB (predictive GLM)
    3. LDA (linear discriminant)

*\* You CAN analyze multiple conditions simultaneously using GLM, but we only produce a set of preprocessed analysis results, and NOT the optimized pipelines. This is partly due to the difficulty of jointly optimizing a subspace consisting of multiple dimensions (SPMs). We would recommend optimizing each stimulus individually, unless you have good reason to do otherwise.*

1. **Event-Related design:**

A brief stimulus event is presented (e.g. of shorter duration than HRF time-to-peak). These analyses estimate the dynamic BOLD response curve itself, along with the associated brain pattern(s). Unlike Block design, you can capture time information about the shape of the HRF; however this comes at the expense of decreased estimation power, and a more complex, potentially unstable signal space. **As with block design, the pipeline currently only supports full optimization of a single stimulus type.**

Available Models:

* + 1. erGLM (general linear model)
    2. erGNB (predictive GLM)
    3. erCVA (canonical variates analysis)

1. **Single-condition design:**

This is where models only expose subjects to a single stimulus type. The most common example is "resting-state" (e.g., no overt task demands), but other examples may occur (e.g., movie viewing, performing mental arithmetic, enforced rumination). Because there is no structured temporal design, these analyses focus on coherent brain dynamics. We only have two models thus far.

Available Models:

1. SCONN (seed-based connectivity)
2. gPCA (principal component generalization)

## iv. Considerations for Model Selection

**Univariate vs. Multivariate:**

This involves a trade-off in model complexity. Univariate models are highly constrained, and can potentially have greater consensus under strong activations. Multivariate models are flexible, requiring model optimization; but they have the potential to incorporate voxel covariance, which improves signal detection sensitivity.

Examples: LDA (multivariate) vs. GNB (univariate)

erCVA (multivariate) vs. erGNB (univariate)

**Performance metrics:**

As part of the analysis process, we obtain performance metrics, which measure the "quality" of preprocessed analysis results. Although all models are run in a split-half framework, some have different metrics. Nearly all models can measure spatial reproducibility (R) of activation patterns. However, only predictive models measure model generalizability, by estimating prediction accuracy (P). These are both important metrics, reflecting bias-variance tradeoffs;

optimized R -> increases model stability (decreased variance), but is at risk for model bias, which it cannot measure

optimized P -> increases model accuracy (decreased bias), but tends to inflate variance in model parameters.

Examples: GNB (P+R) vs. GLM (R only)

erGNB (P+R) vs. erGLM (R only)

We recommend incorporating both prediction AND reproducibility into pipeline optimization, to get a more robust model!

\* For "Single-condition" tasks (e.g., resting state), prediction is usually not an option, as there is no structured task design to predict on. In this case, exercise caution when reviewing results, as the most reproducible brain patterns may reflect structured noise (e.g., motion and physiology). If you are unsure, we recommend using our metrics of global signal, white matter, and head motion artifact as guidelines.

# 5. Group-Level Pipeline Optimization

It is possible to perform the pipeline optimization at the group-level. These should be used if you are interested in directly optimizing multi-run analyses, i.e. analyses across all subjects or runs.

By using --autodetect switch when running Parts 1, 2 of the pipeline, the software automatically detects different runs of subjects, and performs a multi-run analysis. The data are split across runs. The switch --N\_resample has to be used to specify the number of resampling.

# 6. References

Campbell K et al. (2013). Age Differences in the Intrinsic Functional Connectivity of Default Network Subsystems. Frontiers in Human Neuroscience. 5:73

Churchill NW et al. (2012a): Optimizing Preprocessing and Analysis Pipelines for Single-Subject fMRI: 2. Interactions with ICA, PCA, Task Contrast and Inter-Subject Heterogeneity. PLoS One. 7(2):e31147

Churchill NW et al. (2012b): Optimizing Preprocessing and Analysis Pipelines for Single-Subject FMRI. I. Standard Temporal Motion and Physiological Noise Correction Methods. Human Brain Mapping 33:609–627

Churchill NW, Strother SC (2013): PHYCAA+: an optimized, adaptive procedure for measuring and controlling physiological noise in BOLD fMRI. NeuroImage 82:306-325.

Cox RW (1996): AFNI: Software for analysis and visualization of functional magnetic resonance neuroimages.Computers and Biomedical Research, an International Journal, 29(3): 162-173.

Glover GH, et al. (2000): Image-based method for retrospective correction of physiological motion effects in fMRI: RETROICOR. Magnetic Resonance in Medicine: Official Journal of the Society of Magnetic Resonance in Medicine / Society of Magnetic Resonance in Medicine. 44(1): 162-167.

LaConte S et al. (2003): The Evaluation of Preprocessing Choices in Single-Subject BOLD fMRI Using NPAIRS Performance Metrics. NeuroImage, 18(1):10-27

Shaw ME et al. (2003): Evaluating subject specific preprocessing choices in multisubject fMRI data sets using data-driven performance metrics. NeuroImage. 19(3):988-1001.

Strother SC, Anderson J, Hansen LK, Kjems U, Kustra R et al. (2002): The Quantitative Evaluation of Functional Neuroimaging Experiments: The NPAIRS Data Analysis Framework. NeuroImage 15:747–771

Strother S et al. (2004): Optimizing the fMRI data-processing pipeline using prediction and reproducibility performance metrics: I. A preliminary group analysis. NeuroImage. 23:S196-S207

Tegeler C et al. (1999): Reproducibility of BOLD-based functional MRI obtained at 4 T. Hum. Brain Mapp. 7:267–283

Zhang J et al. (2009). Evaluation and optimization of fMRI single-subject processing pipelines with NPAIRS and second-level CVA. Magn. Reson. Imag. 27:264–278

# Appendix A: Tips for Installing AFNI and FSL packages

**AFNI**: you will need to install this package as it is required for multiple preprocessing steps, e.g. motion correction, slice-timing correction, spatial smoothing. This can be obtained from the installation website, which gives helpful step-by-step instructions:

afni.nimh.nih.gov/pub/dist/HOWTO/howto/ht00\_inst/html/linux\_inst\_basic.html

Although you can download and compile AFNI locally, it is often easier to just get the pre-compiled binaries. This can be done as follows:

* + - 1. First determine what OS you are running, as different AFNI binaries are compiled for different OSes. Use command "uname -m" in the unix terminal where you will be running pipelines. You'll get something like:

"Linux [...] x86\_64 GNU/Linux”

this tells you it is a 64-bit Linux OS (x86\_64 = 64-bit, anything else is usually 32-bit, e.g. i386)

* + - 1. now you have to identify and download the correct package from afni.nimh.nih.gov/pub/dist/tgz/. For example we want the "linux\_xorg7\_64.tgz" package (standard 64-bit linux one)

type:

wget http://afni.nimh.gov/pub/dist/tgz/linux\_xorg7\_64.tgz

and wait for download to terminate

* + - 1. Now unzip/un-tar the downloaded file, and move into directory of choice. Make sure it is in path. e.g. if we have moved it to folder "abin" type

export PATH=$PATH:~/abin

this must be but into ~/.bashrc file, so that it loads path on for every job

**FSL**: you will need to install this package in order to use the nonlinear registration tools, to spatially normalize functional data to a common template. They also have helpful instructions at:

fsl.fmrib.ox.ac.uk/fsldownloads/fsldownloadmain.html

You can either (1) download their “installer” Python script, or download the full FSL package (quite large, ~1.7 Gb in size). Option (1) is rapid and should automatically configure your environment to run FSL. However it requires administrator privileges to install and configure, and may throw unexpected errors. If you go by option (2), use the “download” tab to choose the appropriate build (e.g. Linux CentOS5 64-bit works for DTU servers).

As with AFNI, if you choose (2) you will have to unzip the file, and move to your chosen directory. You will also have to set your terminal environment so that all FSL software is in your path. For example in bash shell, you will add the following to a ~./bash\_profile script (you may have to create it):

FSLDIR=/usr/local/fsl

. ${FSLDIR}/etc/fslconf/fsl.sh

PATH=${FSLDIR}/bin:${PATH}

export FSLDIR PATH

where you would replace /usr/local/fsl with the directory your have installed FSL into. Refer to fsl.fmrib.ox.ac.uk/fsl/fslwiki/FslInstallation/ShellSetup for further instructions and details for other OSes.

# Appendix B: Individual Pipeline Optimization Script Details

There are three scripts underlying the “wrappers”. These scripts can be run off the grid, via command line, and should be run in the following order:

* + 1. Pipeline\_STEP1.m
    2. Pipeline\_STEP2.m
    3. Pipeline\_STEP3.m

Make sure all files and folders in the pipeline package are installed in your working directory.

The following sections describe how each underlying script functions and how to run them via command line.

### Pipeline Scripts #1 and #2

These scripts are “Part-1” of running the individual optimization on the grid.

**Pipeline Script #1 Documentation: Pipeline\_STEP1.m**

This script has the capability to perform the following steps:

**Fixed steps**

1. Removes non-equilibrium/instructional scans, specified in {subject input}.txt
2. Generates brain masks
3. Identifies minimum-displacement fMRI volume for reference, and then runs MOTCOR. This produces corrected 4D data and Motion Parameter Estimates (MPEs)
4. Runs diagnostic script on data (with and without MOTCOR), producing plots and matfiles with diagnostic information, including a list of potential motion spikes

**Optional steps, specified in {pipeline list}.txt**

1. Retains data with and without MOTCOR
2. Remove spikes and interpolates (CENSOR)
3. Performs RETROICOR
4. Performs TIMECOR
5. Performs SMOOTH for specified kernel(s)

Requirements:

This script requires formatted {subject input}.txt and {pipeline list}.txt files

How to run command line:

1. To execute the script:
   1. In Matlab and Octave: Pipeline\_STEP1(‘{subject input}.txt’, ‘{pipeline list}.txt mat’);

**Pipeline Script#2 Documentation: Pipeline\_STEP2\_MM.m**

This script has the capability to perform the following steps:  
**Fixed Steps**

1. Obtains maps of brain edges, to detect motion artifact later
2. Generates mask of vascular/CSF regions and downweights them
3. Generates mask of white matter tissue to test global signal bias

**Optional steps (specified in {pipeline list}.txt**

1. Runs specified DETREND orders
2. Performs MOTREG using the MPE estimates from script #1
3. May include task design as part of the GLM model (TASK)
4. Estimates and removes Principal Comonent #1, a potential global signal confound (GSPC1)
5. Physiological regression using PHYCAA+ (PHYPLUS)

**Fixed Final Step**

1. Runs split-half NPAIRS analysis on preprocessed data, generating brain maps and performance metrics

Requirements:

This script requires formatted {subject input}.txt and {pipeline list}.txt files.

How to run command line:

1. At Matlab or Octave command line – execute as function:

“Pipeline\_STEP2 (‘{subject input}.txt’, ‘{pipeline list}.txt’, analysis\_model, niiout, contrast)

* 1. **analysis\_model**: a string that specifies the type of analysis model
     1. ‘LDA’: Liner discriminant
     2. ‘GNB’: Gaussian Naive Bayes
     3. ‘erCVA’: Event-related Canonical Variates Analysis
  2. **niiout**: Default option is in ‘matfile’ format, but this flag specifies whether it is also output to a nifti file
     1. 0: matfile (‘.mat’) output only
     2. 1: nifti files (‘.nii’) in addition to ‘.mat’ files
  3. **contrast**: Default option is ‘1-2’. But you can specify more complex contrasts, or multiple contrasts. For example, ‘1+2+3-baseline’

### Pipeline Script #3: Pipeline\_STEP3.m

This script produces optimized analysis results, under three different optimization strategies. Two pipelines are fixed (i.e. the same pipeline across all subjects), and one pipeline is individually optimized:

1. **std:** “standard” fixed pipeline; all steps except PHYCAA+ GSPC1, TASK turned on; detrending order is set using AFNI’s heuristic function; smoothing at 6mm FWHM
2. **fix:** highest-ranked fixed pipeline
3. **ind:** individually optimized subject pipelines

This step is designed to optimize across multiple subjects and pipelines

* if you have less than five subjects in your input file, optimization results may be unreliable
* if you have less than three subjects in your input file, the program immediately terminates
* if you are only testing 1 pipeline, the program immediately terminates

This script performs the following steps:

1. Loads all subjects “METRIC\_set” data, and concatenated values
2. Performs pipeline optimization, selecting the three different optimal pipeline strategies mentioned above (std/fix/ind)
3. Acquires optimal pipeline SPMs and time-series data
4. Performs spatial similarity testing of selected pipeline statistical parametric maps (SPMs)
5. Re-generates the optimially preprocessed 4D data

Requirements:

This script requires formatted {subject input}.txt

How to run command line:

1. At Matlab or Octave command line – execute as function:

Pipeline\_STEP3( '{subject input}.txt', optimize\_metric, mot\_gs\_control, dataoutfix, process\_out )

Where:

1. **optimize\_metric:** A metric from analysis in Step2 that is used to compare and optimize pipeline choice. This needs to be a field in the “METRIC\_set” output. For example, ‘LDA’ analysis model has options of {‘P’,’R’,’Dneg’}
2. **mot\_gs\_control:** 2D binary vector, indicating whether we control for motion artifact and global bias using spatial priors:
   1. **[0 0] = no artifact control**
   2. **[1 0] = control motion artifact only**
   3. **[0 1] = control global signal bias only**
   4. **[1 1] = control both**
3. **dataoutfix:** string specifying prefix for saved results (in Matlab format)
4. **process\_out:** binary value specifying whether optimally preprocessed 4D nifti data is output(includes 3 fixed pipelines; 1 individually optimized)
   1. **0 = no**
   2. **1 = yes**

# Appendix C: Spatial Normalization Script Details

This section describes how to use the spatial\_normalization.m script underlying the part 3.

* + - 1. Inter-subject alignment: between-subject alignment of functional data, using a 2-stage transformation process

1. Non-rigid (affine) warping of subject anatomical volumes (T1) to template
2. Rigid alignment of functional (EPI) to anatomical (T1) data. This is used for group-level analysis across SPMs

Alignment procedures should ideally be run after “Pipeline\_STEP3.m”, as it automatically normalizes the optimally preprocessed data.

## i. Inter-Subject Alignment

The spatial\_normalization.m to generate subject alignments to a group template.

* + - 1. Mask\_GroupT1template.out
      2. Prep\_Transform.out
      3. Run\_Transform.out

Given an initial reference template, the first script uses this as a reference to generate a robust group-average T1 map (adaptive group-specific template). The second script prepares for spatial normalization by generating preparatory information (e.g. fMRI volume dimensions). The third script generates a net EPI to group template transformation matrix. This script then applies the transformation to the optimized pipeline output obtained by the “Run\_Pipelines.py” script (part 2).

This procedure has one new requirement: a T1 reference anatomical template (e.g. an MNI or Talairach atlas), used as a reference for the spatial normalization.

### Inter-Subject Alignment Script #1: Mask\_GroupT1template.out

This script performs the following steps:

1. For each T1 volume, it computes the non-rigid transformation to reference template
2. Takes the average of the transformed T1s as the new reference
3. Repeats the process (iterative) (i.e. each subjects original T1 is aligned to the new reference, and then averaged to create a second new reference. This is repeated as many times as specified by the user)

Requirements

This step should be applied after running “Run\_Pipelines.py” (part 2). A reference {reference T1 anatomical}.nii should be chosen for this step.

How to run command line

1. If uncompiled, type ”g++ -o MakeTransform\_IntraSubject .out MakeTransform\_IntraSubject.cpp” to compile script
2. Execute with command “./MakeTransform\_IntraSubject.out {subject input}.txt {reference T1 anatomical}.nii {prefix of new anatomical average} {number of iters}

**Where:**

1. {prefix of new anatomical average}: name of new reference made from subject T1s
2. {number of iters}: the number of iterations – i.e. the number of times the process is repeated. Usually “3” is sufficient for stable results

Note: This process uses non-rigid warping to match structural volumes to the reference volume

### Inter-Subject Alignment Script #2: Prep\_Transform.out

This script performs the following steps:

1. Records the dimensions of fMRI images (in “niidims.txt”)
2. Creates blank fMRI volume, for reference transformations (“blankvol.nii.gz”)
3. Creates an Identity matrix, for T1 down-sampling (“eye.mat”)

Requirements

Requires {subject input}.txt and {reference T1 anatomical}.nii inputs

How to run command line

1. If uncompiled, type “g++ -o MakeTransform\_InterSubject.out MakeTransform\_InterSubject.cpp” to compile script
2. Execute with command “./Run\_Transform.out {subject inputs.txt} {reference T1 anatomical }.nii {optimization metric}

### Inter-Subject Alignment Script #3: Run\_Transform.out

This script performs the following steps:

1. Masks the T1 brain volume
2. Computes the non-rigid transformation of each T1 volume to the group template from script #1
3. Computes the affine (rigid-body + scaling) alignment of EPI data to the subject’s corresponding T1 volume, producing the transformation matrix “Transmat\_EPItoREF\_{prefix}.mat”
4. Computes the net transformation matrix from the above two steps (EPI to T1 volume to group template)
5. Spatially transforms 4D data with basic preprocessing (called “baseproc”) with transformation matrix computed in above Step #4, this is used to build group-level brain masks
6. Spatially transforms optimized SPMs with transformation matrix computed in above Step #4
7. Spatially transforms optimally preprocessed data runs in above Step #4
8. Produces downsampled structural T1, to use as reference. It is named “spat\_norm/{struct prefix}\_T1toREF\_downsamp.nii

Requirements

Requires {subject input}.txt and {reference T1 anatomical}.nii inputs

How to run command line

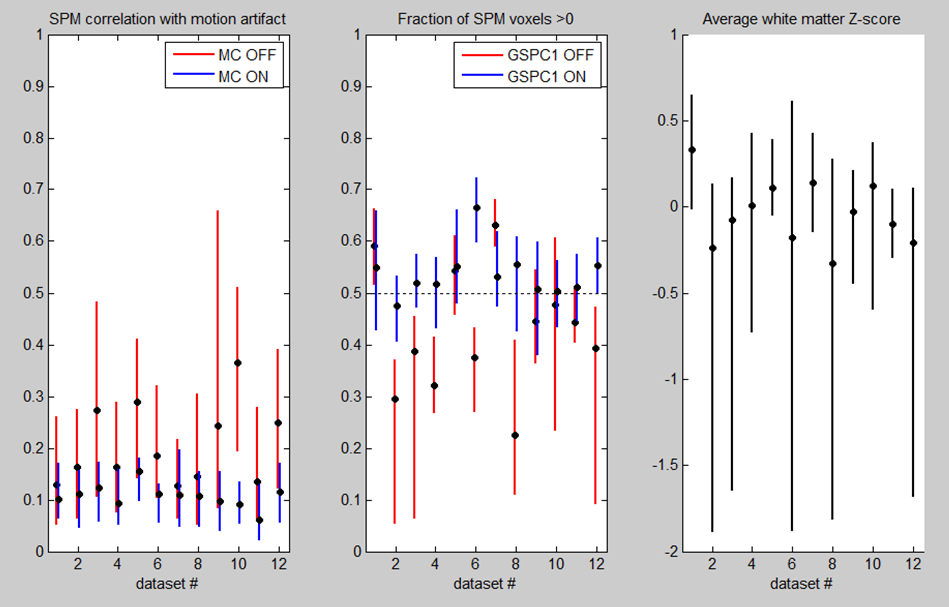
1. If uncompiled, type “g++ -o MakeTransform\_InterSubject.out MakeTransform\_InterSubject.cpp” to compile script
2. Execute with command “./Run\_Transform.out {subject inputs.txt} {reference T1 anatomical }.nii {optimization metric}

# Appendix D: Sample Quality Control Figures

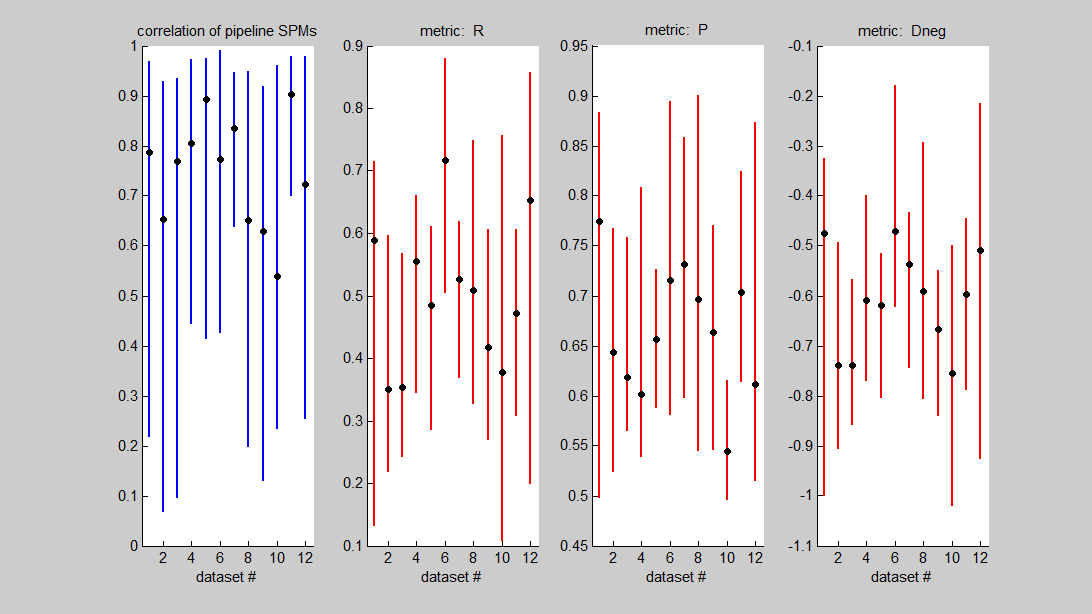
## i. Quality Control-1

## C:\Users\Nathan\Documents\MATLAB\2_Code_Development\1_pipelines\temp2\QC1_images\qc1_FIG1_motion_statistics.png

**FIG1\_motion\_statistics:** This plot provides some statistics on head motion, for each dataset being optimized. (top) average displacement on the 6 rigid-body motion parameter estimates. (middle) the number of volumes identified as head motion spikes during CENSOR pipeline step for MPE (motion parameter estimates), fMRI data (with/without motion correction (MC)), and number of volumes that are spikes in both MPE and fMRI data. (bottom) correlation between task paradigm and first PC of motion parameters – determines if motion is task-coupled.

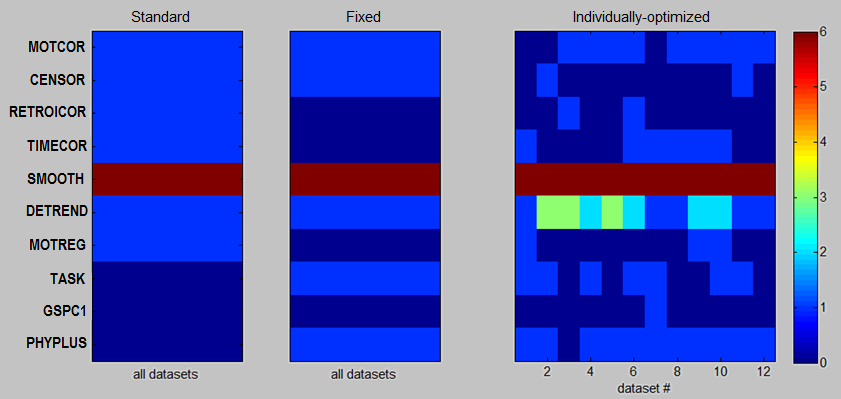


**FIG2\_SPM\_artifact:** This plot shows potential artifact present in SPMs, for each dataset being optimized. Vertical bars show distribution across all tested pipelines, per dataset. (left) correlation with motion artifact, estimated by spatial gradient maps. (middle) fraction of signal >0 (“globalness” of signal). (right) average Z-score in white matter tissues.

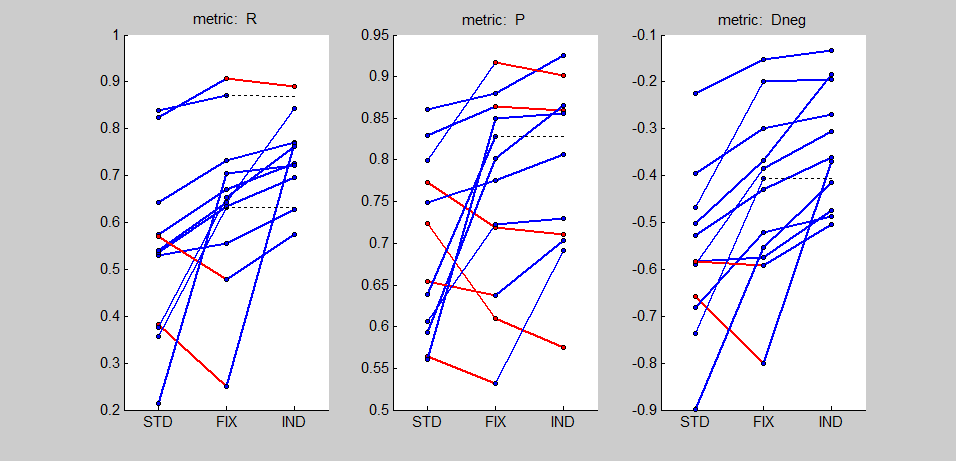


**FIG3\_pipeline\_similarity\_by\_dataset:** This plot shows how much results are altered by different pipeline choices, for each dataset being optimized. Vertical bars show distribution across all tested pipelines, per dataset. (left) pairwise correlation between all pipeline SPMs. All other plots show distribution of NPAIRS metrics across pipelines.

## ii. Quality Control-2



**FIG1\_optimized\_pipeline\_steps:** List of pipeline choices for each optimization strategy. For 8/10 steps, 1=ON 0=OFF. The only exceptions are SMOOTH (=kernel FWHM) and DETREND (=maximum polynomial order *k*).



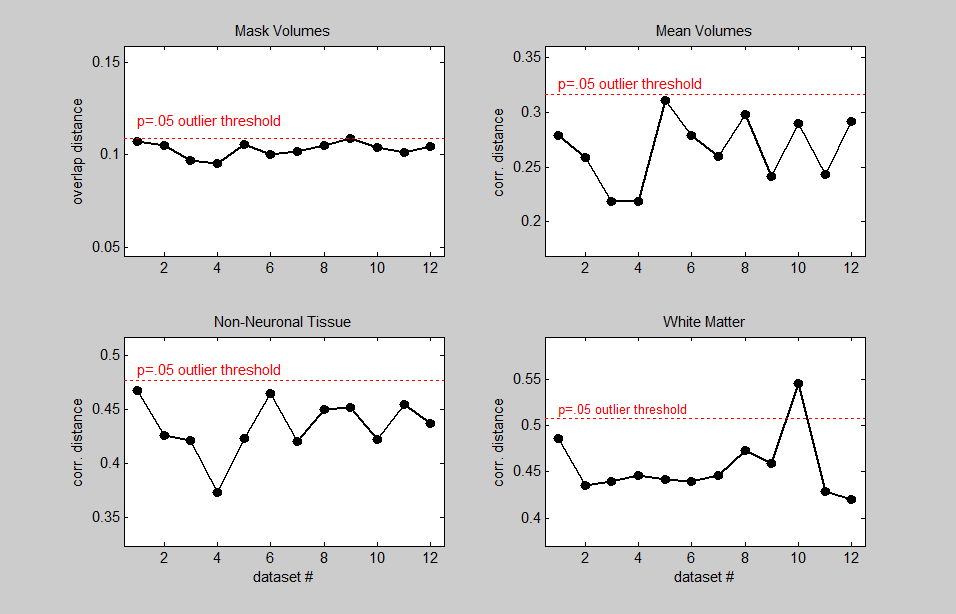
**FIG2\_optimized\_performance\_metrics:** Performance metrics as a function of pipeline optimization method (STD, FIX, IND), plotted for each dataset. Blue bars indicate improved performance with increasing pipeline flexibility; red bars indicate the converse.

22/05/2014 E-mail: On the spatial-norm statistics:

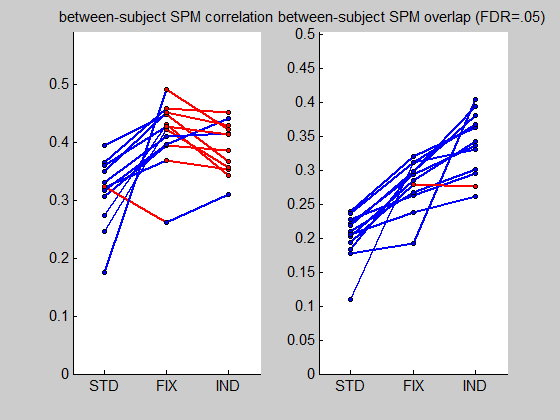
-for generating masks and spatial transformation matrices, I use a single basic pre-processing pipeline (motion correction, slice timing, 6mm smoothing), with is produced as part of the intermediate pipeline outputs (has a "baseproc" suffix).

-we generate masks by applying 3dAutomask to the transformed EPI "baseproc" data from each subject

-similarly, "mean volumes" is the correlation of mean functional volumes, for subjects' "baseproc" pipelines.



**FIG3\_spatial\_norm\_statistics**: Metrics reflecting the quality of spatial normalization across subject datasets. Plots indicate mean overlap/correlation distance of each dataset, relative to all others. Dashed red line indicates threshold for significant outliers (p<.05 significance).



**FIG4\_inter\_subject\_SPM\_similarity:** inter-subject reliability metrics of (left) SPM correlation and (right) overlap of thresholded SPMs, as a function of pipeline optimization method, plotted for each dataset. SPMs are thresholded at False-Discovery Rate FDR=.05.

**FIG5\_group\_pca\_{STD/FIX/IND}\_pipeline**: plots showing Principal Components (PCs) #1,2 on subject SPMs, as a function of pipeline optimization methods. Brain maps are spatially Z-scored, and subject loadings are plotted below.

