**PRONTO User’s Manual (v 8.5)**

1. Introduction 3

2. Overview of PRONTO Framework 4

i. Script overview 4

3. Running Pipeline Optimization 6

i. Requirements 6

ii. Creating Pipeline Input Text Files 7

File #1: {subject input}.txt 7

File #2: {pipeline list}.txt 9

iii. Specifying your Analysis 10

iv. Running Individual Pipelines 13

EXAMPLES 20

QUALITY CONTROL 21

5. Group-Level Pipeline Optimization 22

6. References 22

Appendix A: Tips for Installing AFNI and FSL packages 24

Appendix B: Individual Pipeline Optimization Script Details 26

Appendix C: Spatial Normalization Script Details 28

Appendix D: Split\_info structure 28

Appendix E: Sample Quality Control Figures 29

i. Quality Control-1 29

ii. Quality Control-2 31

Appendix F: Sample Quality Control Figures 34

i. Determining the Optimal Fixed Pipeline (FIX) 35

ii. Determining the Optimal Individual Pipeline 36

# 1. Introduction

This manual provides instructions for running **PRONTO** (**PR**eprocessing **O**ptimizatio**N** **TO**olkit), which does fast optimization of preprocessing pipelines for BOLD fMRI (Blood Oxygenation Level Dependent functional MRI). This software package identifies the set of preprocessing steps (“pipeline”) specific to each dataset, which optimizes quality metrics of Prediction and Reproducibility for post-processing analysis results (Strother et al., 2002; see Appendix E for more information about these metrics). This procedure has been shown to significantly improve signal detection, reliability of brain activations, and sensitivity to brain-behaviour correlations (Churchill et al., 2012a, 2012b). The pipeline software can also be used for simple automated batch-processing of fMRI datasets, if no appropriate analysis model is currently available to do optimization (e.g. some resting-state connectivity studies).

What can PRONTO do for you?

Clean up your data: it will automatically output data that has been optimally processed to control for noise and artifact (e.g. due to motion, physiology, scanner noise), which you can then use for further analysis. Automated de-noising is done based on replicable, statistical criteria; no more tedious batch scripting, or hand-selection of ICA components!

Analyze your data: as part of optimization, PRONTO creates Z-scored maps of brain activity for each dataset. You can directly report these results, or take individual activation maps and do standard group-level analysis. PRONTO can perform univariate or multivariate analysis of brain activity. This can be done for a variety of different task designs, including event-related and block-design tasks. Recent additions also include seed-based connectivity and component modelling, i.e., selecting an optimal PCA subspace.

Run batched pipelines: if you cannot analyze your data (or PRONTO does not have the appropriate analysis tools), you can still process it using PRONTO without doing optimization. Our scripts make it straightforward to choose the pipeline steps that you want, and perform automatic batch processing of large datasets.

Preprocessing and analysis scripts are coded in Matlab/Octave, and functions are called and managed using Python scripts. The code tests all possible combinations of a set of 12 different preprocessing steps, to identify the optimal pipeline for each fMRI dataset that is being tested. Current preprocessing options include AFNI utilities (Analysis of Functional NeuroImaging; Cox, 1996), along with a set of functions developed in-house. All steps are 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, 2012b). Pipeline optimization is analysis-driven: it evaluates the quality of analysis results for each pipeline, via Prediction and Reproducibility metrics, and selects the pipeline that gives highest-quality outputs. Analysis techniques are “modular” – you choose the task design and analysis model you wish to optimize, from a list of available models. The pipeline software also includes a procedure for automated spatial normalization of subjects to an anatomical template, using FSL utilities. This enables users to run group-level analysis of preprocessed results across subjects and task runs.

# 2. Overview of PRONTO Framework

## i. Script overview

The pipelines are run from a single wrapper script:

**Run\_Pipelines.py**

This wrapper script calls four scripts (corresponding to 4 main steps), which run in the following order:

1. **pipeline\_wrapper.py:** runs all selected pipelines, computes metrics of pipeline quality

2. **optimization\_wrapper.py:** finds optimal pipelines, creates brain maps & processed data

3. **spatial\_normalization\_wrapper.py:** registers preprocessed data to a structural template

4**. mask\_tissue\_wrapper.py:** generates a group mask

Each script calls a set of Matlab/Octave functions, along with pre-compiled AFNI and FSL functions.

This pipeline automatically tests a set of different preprocessing algorithms in a fixed order, 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. 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); if you are running pipelines locally, make sure the packages are in your path. Both are freely available online; refer to **Appendix A** for installation tips.

**Table 1. list of currently available pipeline steps**

|  |  |  |  |
| --- | --- | --- | --- |
| **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 motion1. | AFNI’s 3dvolreg | OFF/ON |
| Censoring outlier volumes; also does more aggressive component-based filtering (**CENSOR**) | Basic censoring: identifies significant outlier volumes in fMRI time series, which are discarded and replaced with interpolated values from neighbouring volumes. Aggressive censoring: uses PCA2 (default) or ICA2 to remove components with significant temporal spiking, or significant edge artefact (To be published). | Octave script developed in-house (Campbell et al.,2013)3 | OFF / BASIC / AGGRESSIVE (PCA or ICA) |
| 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 ***must be acquired during the scanning session.*** | AFNI’s 3dretroicor | 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 |
| Subject-specific non-neuronal tissue mask | Generates a data-driven, subject-specific mask of predominantly nonneuronal tissue voxels (vasculature, sinuses and ventricles), which should be excluded from subsequent processing. Otherwise, these voxels produce false-positive activations, and biased estimates of spatial reproducibility. | Octave script developed in-house  (Churchill and Strother, 2013) | Fixed ON |
| 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 motion parameter variance. This corrects for residual motion artifact. | Octave script developed in-house | OFF/ON |
| Task Covariate regression (**TASK**) | Includes task design as a covariate when regressing out DETREND, MOTREG and GSPC1 (see below). This step protects against over-regression of task-related BOLD | 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 (Carbonell et al., 2011). | Octave script developed in-house | OFF/ON |
| Physio. correction; data-driven  (**PHYPLUS**) | With subject-specific, non-neuronal tissue masking already performed 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 |
| Regress out mean time-series in a user defined mask.  (**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 |
| Low-pass filtering (**LOWPASS**) | Uses a linear filter to suppress BOLD frequencies above 0.10 Hz, which contain a relatively large amount of physiological noise power4. | Octave script developed in-house | OFF / ON |

1The reference volume for alignment is automatically chosen by identifying the volume with estimated minimum overall head displacement in the scanning run, defined as the volume with minimum Euclidean distance from the median coordinates in Principal Component (PCA) space of the 4D data set.

2Full pipeline optimization including PCA censoring is quite slow, even using high performance computing resources with a Sun Grid Engine; using ICA instead of PCA is even slower. See section 3(ii) for details on how to specify censoring options.

3 Only includes description of basic despiking in this paper (CENSOR=1).

4 This is primarily used for resting-state data, to minimize potential physio. noise (as functional connectivity measures may be more sensitive than task-based analysis). We recommend applying with caution as it is a very conservative step, and there is some evidence of BOLD response power >0.1 Hz that may be relevant to cognitive function.

This software package allows users to test all possible combinations of choices for these pipeline steps. For every tested pipeline, the dataset of interest is preprocessed and analyzed (using your chosen analysis model and task design), which produces performance metrics of (Prediction, Reproducibility). The script then compares across all tested pipelines, and identifies the pipeline that optimizes these performance metrics (see Appendix E for further details on these metrics).

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

Standard, conservative pipeline (CON): this approach applies a single “conservative” set of preprocessing steps to each fMRI dataset, designed to minimize potential noise confounds without concern for signal optimization; this is a (somewhat enhanced) version of standard processing options in fMRI literature, included for comparison purposes. The steps included are: motion correction with optimal reference volume selection, basic outlier censoring, RETROICOR, slice-timing correction, spatial smoothing (if multiple kernels are tested, it selects the one closest to 6 mm FWHM), subject-specific non-neuronal tissue masking, motion parameter regression, and linear detrending (if multiple orders are tested, it selects the one closest to AFNI's heuristic criterion; afni.nimh.nih.gov/pub/dist/doc/ program\_help/3dDeconvolve.html).

Fixed optimization (FIX): selects the single fixed set of pipeline 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. In cases where there are multiple task runs or contrasts of interest, the user may choose the pipeline that gives best average performance (this is explained in further detail in Section 5 below).

# 3. Running Pipeline Optimization

This section lists requirements for runing PRONTO. This includes fMRI data, optional T1 anatomical and physiological data, and some user-supplied text files, which specify the data and analysis of interest. We also provide tutorial instructions of how to create input files and submit jobs to PRONTO (with examples).

## i. Requirements

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

(this is optional –only required if you want to normalize all data to a common template space)

1. Cardiac and Respiratory data, converted into .1D design files; must be formatted as {name\*}.puls.1D and {name}.resp.1D file extensions (respectively).

(this is optional –only required if you want to run RETROICOR step in pipelines)

1. A set of formatted textfiles, specifying (a) the name and location of your data, (b) pipeline steps you want to test, and (c) task information, condition onsets and duration:
   1. File #1: {subject input}.txt
   2. File #2: {pipeline list}.txt
   3. File(s) #3: {task information}.txt

You will have to write these yourself, but they have a simple format with step-by-step instructions below (Sections **3.ii** and **3.iii**). Once written, you can also re-use them for different analyses and pipeline choices.

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

## ii. Creating Pipeline Input Text Files

### 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 separated by spaces and described below. ***Note that only fields 1-3 are mandatory*** (and field 4 is required to do analysis & optimization). The rest depend on if you want to test specific options.

1. **IN**={ (path)/(input nifti file) }
   1. Name and location of unprocessed fMRI data you wish to optimize
2. **OUT**={ (path)/(output prefix) }
   1. Name and location for final processed & optimized outputs
3. **DROP**=[{# scans to drop at start},{# scans to drop at end}]
   1. Discards non-equilibrium and instructional scan volumes at the ends of the run

**ex**. DROP=[2,2] discards the first two volumes and last two volumes of the time series

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

1. **TASK**={(path)/(task information)}
   1. Name and location of formatted text file, describing the experimental paradigm for this dataset, including task onsets and duration. You must create this file yourself, with step-by-step instructions below.
   2. In the {task information file}, onsets are 1-relative (i.e. first volume=1), and indexed relative to the first scan volume before any volumes are dropped.

**\* TASK** field can be omitted if you are not doing analysis or pipeline optimization

**\* If datasets have the same task paradigm (e.g. fixed onsets and durations), you can re-use the same (task information) file for all of these datasets.**

1. **PHYSIO**={ (path)/(physiological data prefix) }
   1. Name and location of Cardiac and Respiratory data, converted into .1D design files. must be formatted as (path)/(physiological data prefix).puls.1D and (path)/(physiological data prefix).resp.1D, respectively

**\* PHYSIO** field can be omitted if RETROICOR is not being tested in pipeline

**\* If RETROICOR is being tested and some subjects are missing physio files, those subjects must be processed in a separate ‘{subject input}.txt’ file**

1. **STRUCT**={ (path)/(T1 anatomical nifti file) }
   1. Name and location of subject’s 3D structural brain image, which is used to co-register data to a common MRI template space, if desired.

**\* STRUCT** field can be omitted if you do not warp subject data to common template

1. **CUSTOMREG**={(path)/(nifti file)}
   1. Name and location of binary mask, where 1=brain regions containing noise confounds (e.g. white matter). The mean time-series of this ROI is computed and regressed out of all voxels in the brain by the CUSTOMREG pipeline step.

**\*** This volume must have the same dimensions as the user’s **STRUCT** file. If you want to perform CUSTOMREG, you will need to create the binary ROI mask yourself, e.g. using one of the neuroimaging packages that allows you to manually edit images (e.g. fslview, mricron, afni). For advanced users only!

**\* CUSTOMREG** field can be omitted if you do not want perform this step

Tips for {subject input}.txt file:

* 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** mandatory fields must be included in every line (i.e. for every dataset being optimized). Optional fields should be consistently included/excluded from all lines in the textfile.
* You choose how you want to analyze your data (e.g. task onsets and analysis model). See section ***2(iii)*** below for more information on task design and analysis models

Example Text for File#1:

For 1 run from each of 5 different subject datasets, you would produce a textfile with the 5 lines below. Note that “CUSTOMREG” field is omitted, meaning that this pipeline step will be fixed OFF.

|  |
| --- |
| IN=my\_dir/subj1\_task.nii OUT=new\_dir/subj1 DROP=[2,1] TASK=taskInfo\_subj1.txt PHYSIO=my\_dir/physio/subj1 STRUCT=my\_dir/subj1\_T1.nii  IN=my\_dir/subj2\_task.nii OUT=new\_dir/subj2 DROP=[2,1] TASK=taskInfo\_subj2.txt PHYSIO=my\_dir/physio/subj2 STRUCT=my\_dir/subj2\_T1.nii  IN=my\_dir/subj3\_task.nii OUT=new\_dir/subj3 DROP=[2,1] TASK=taskInfo\_subj3.txt PHYSIO=my\_dir/physio/subj3 STRUCT=my\_dir/subj3\_T1.nii  IN=my\_dir/subj4\_task.nii OUT=new\_dir/subj4 DROP=[2,1] TASK=taskInfo\_subj4.txt PHYSIO=my\_dir/physio/subj4 STRUCT=my\_dir/subj4\_T1.nii  IN=my\_dir/subj5\_task.nii OUT=new\_dir/subj5 DROP=[2,1] TASK=taskInfo\_subj5.txt PHYSIO=my\_dir/physio/subj5 STRUCT=my\_dir/subj5\_T1.nii |

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

This is a text file listing the choices for each of the 12 pipeline steps that you want to test during pipeline optimization. Individual fields are formatted as {pipeline step}=[{options}], where {options} is a comma-separated list of choices that you are testing.

**Table 2. list of pipeline input options**

|  |  |  |
| --- | --- | --- |
| **Entry in File** | **Pipeline Step** | **Options** |
| MOTCOR=[{options}] | Rigid-body motion correction | “0”,”1” or “0,1” (0=OFF, 1=ON) |
| CENSOR=[{options}] | Censoring outlier spikes in data | Comma-separated list of 0,1,2,3 options  (0=OFF, 1=BASIC, 2=AGGRESSIVE PCA, 3=AGGRESSIVE ICA) |
| RETROICOR=[{options}] | Physiological noise correction #1 | “0”,”1” or “0,1” (0=OFF, 1=ON) |
| TIMECOR=[{options}] | Slice-timing correction | “0”,”1” or “0,1” (0=OFF, 1=ON) |
| 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}] | Motion parameter regression | “0”,”1” or “0,1” (0=OFF, 1=ON) |
| TASK=[{options}] | Include task design covariate | “0”,”1” or “0,1” (0=OFF, 1=ON) |
| GSPC1=[{options}] | Global signal corrected with PC#1 | “0”,”1” or “0,1” (0=OFF, 1=ON) |
| PHYPLUS=[{options}] | Physiological noise correction #2 | “0”,”1” or “0,1” (0=OFF, 1=ON) |
| CUSTOMREG=[{options}] | Regression of noise ROI | “0”,”1” or “0,1” (0=OFF, 1=ON) |
| LOWPASS=[{options}] | Low-pass filtering | “0”,”1” or “0,1” (0=OFF, 1=ON) |

Tips for {pipeline\_list}.txt file:

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

Ex. MOTCOR=[0,1]

* Comma-separated list specifies which options are being tested. For example with MOTCOR:
  + MOTCOR = [0] 🡪 do not apply to any datasets (forced off)
  + MOTCOR = [1] 🡪 apply to all datasets (forced on)
  + MOTCOR = [0,1] 🡪 do not apply and apply to all datasets (test on AND off)
* Note that DETREND, MOTREG and GSPC1 are regressed as nuisance covariates in a general linear model (GLM). If TASK=1, we include the task design as an additional GLM covariate, to protect against over-regression of the task signal.
* **We recommend testing as many pipeline options as possible, as they may all have a significant impact on analysis results, with the caveat that fully optimized testing of all steps will likely be very slow (i.e., days) without significant high performance computing resources.**

Example Text for File #2

If you wanted to test MOTCOR, CENSOR, LOWPASS and DETREND (orders 0 to 3), and leave all other steps OFF (with a fixed 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=[0]  LOWPASS=[0,1] |

### File #3: {task information}.txt

This is a textfile (or set of textfiles) specifying details of your fMRI task paradigm. For each line of the {subject inputs} file described above, the field **TASK**=(…) points to the corresponding {task information} file for that dataset, so that PRONTO knows how to analyze it. If all of your datasets have the same task design (e.g. a fixed block paradigm), you will only need to create a single split-info file. However if onsets are randomized across subjects, different files must be created.

As with File #2, this file will have a series of entries {task parameter}=[{options}], that describe important details about thsse task. You will need to include all of the following “general” inputs (except the “SEED” parameter, which is only required if you are doing seed-based connectivity):

**Table 3. list of “general” task parameters (only need to specify once in file)**

|  |  |  |
| --- | --- | --- |
| **Entry in File** | **Entry** | **Options** |
| UNIT=[{options}] | Unit in which task onset/duration is measured | “TR”, “sec” or “msec”  (#scan volumes, seconds or milliseconds) |
| TR\_MSEC=[{options}] | Time between scan volumes (repetition time) | Integer value, in milliseconds |
| TYPE=[{options}] | Type of task paradigm being processed/analyzed | “block”, “event” or “nocontrast”  (see the next section for discussion of task types, if you are unsure) |
| SEED=[{options}] | Name of ROI brain mask, for seed-based connectivity analysis.  \*only required for SCONN analysis (details in next section) | Should give {path/filename} of 3D binary brain volume (1=seed ROI/0=non-ROI). Volume should be a NIFTI file of same dimensions as the input fMRI data. |

Now, for each task condition in the fMRI run that you want to include in your analyses, you must specify the following three fields:

**Table 4. list of parameters for each condition (need to specify for each condition being analyzed)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Entry in File** | **Entry** | | **Options** |
| NAME=[{options}] | | Name of this task condition | Anything you like, but avoid special characters e.g. “+-[]”. The name "baseline" is reserved for the scans in which the brain is in the "REST" condition (See below for details). |
| ONSETS =[{options}] | | List of onset times, **in time-units specified by UNIT field** | Must be a comma-separated list of numbers, e.g. “ONSETS=[1,10,30]” |
| DURATION=[{options}] | | List of condition durations corresponding to the above onsets, **in time-units specified by UNIT field** | Must be a comma-separated list of numbers, e.g. “ONSETS=[9,10,8]” |

Tips for creating your {task information} files:

* Defining conditions: for every new condition, you will need to specify a NAME, ONSETS and DURATION field. Within each task condition, the number of ONSET entries must match the number of DURATION entries.
* Defining onsets: onset times are relative to the start of the raw data run, i.e. before any scans have been dropped using the DROP field in the {subject input} file. Also, it is assumed that scanning time starts at TR=1, and time sec=0 (and msec=0).
* Multiple conditions: You can specify as many conditions as you like, prior to optimization. Block design requires a minimum of 2 conditions; event-related analysis only requires 1 condition
* Event-related analysis: if data is event-related, all “DURATION” fields should be 0.
* Baseline condition: The baseline condition has to be defined if the two following conditions are met: 1) the data is a block design (i.e. TYPE=[BLOCK]); 2) the preprocessing step TASK is set to ON [1], or ON AND OFF [0,1], or the analysis model is GLM (See the next subsection for the available analysis models). In this case, the users have to specify the onset and duration of those blocks during which the brain is in the resting-state condition (i.e. the participants did not perform any particular task). The name of this condition has to be "baseline" (NAME=[baseline]). This condition will be treated differently in the code; there will not be a regressor associated with this condition in the GLM model or when TASK is ON. It is better to define the baseline condition as the last condition in the task information file (See below for the example). Note that for other cases (the remaining analayis models and TASK=[0]) defining the baseline condition is not necessary and does not have any effect on the final results.
* IMPORTANT NOTE: If 1) the data is a block design (i.e. TYPE = [BLOCK]); AND 2) the preprocessing step TASK is set to ON [1], or ON AND OFF [0,1], or the analysis model is GLM, all conditions in the paradigm MUST be specified, even if they are not going to be analyzed. In addition, one condition must be defined as a ‘baseline’ condition (see above). For example, if your dataset has for conditions A,B,C,D and you want to contrast A vs. C, conditions B and D still need to be defined in the {task information}.txt file ( i.e. the file will contain onset information for Conditions A,B,C,D ), with one condition defined as the ‘baseline’ condition. For example, the conditions can be defined as ‘A’,’B’,’C’, ‘baseline’.

Example Text for File #3

In the example below, we have a block-design task with a TR of 2000 ms. It has three task conditions presented sequentially (A-B-C-A-B-C), each 10 TR in length. In this example, we discard the first 2 TR of each block (hence DURATION values of 8); due to the sluggish nature of the BOLD hemodynamic response, there is a 4-6 second delay before reaching peak activation.

|  |
| --- |
| UNIT=[TR]  TR\_MSEC=[2000]  TYPE=[BLOCK]    NAME=[task\_A]  ONSETS=[13,43]  DURATION=[8,8]    NAME=[task\_B]  ONSETS=[23,53]  DURATION=[8,8]  NAME=[task\_C]  ONSETS=[33,63]  DURATION=[8,8]  NAME=[baseline]  ONSETS=[1,71]  DURATION=[12,12] |

If we saved this file as “design.txt”, in the directory “mydir”, then in File #1, you would point to this file by typing **TASK=mydir/design.txt**.

## iii. Specifying your Analysis

Pipeline optimization is done to maximize the Prediction and Reproducibility of your analysis results, so it is important to decide what sort of analysis you are conducting, see Appendix E for more detail. You have control over your analyses in two ways: (1) what task conditions you want to analyze (specified inside of a {task information} file, described in the last section), and (2) what kind of analysis model you want to use (which is specified when you submit your jobs to PRONTO). This section provides some information to help guide your choice of analysis model. Once you have decided the analysis model you want to use from Table 3, you can go to the next section and start running pipelines!

First, you need to decide how the fMRI data will be analyzed. This depends on your experimental design. Broadly speaking, are three types of fMRI task design:

Block design: a fixed stimulus is presented for an extended duration, usually longer than the time-to-peak for a standard HRF (>6-10 sec.). A block design analysis looks for difference in “mean” signal between different experimental stimulus blocks (for example, task and baseline). If you have runs with multiple conditions, make sure to optimize pipelines specifically for the contrast you are interested in studying. In general, different task contrasts in the same fMRI run require different pipeline choices for optimal signal detection (Churchill et al., 2012a).

Event-Related design: a brief stimulus event is presented, 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 information about the shape of the HRF; this comes at the expense of decreased estimation power, and a more complex, potentially unstable signal space.

Single-Condition design: studies which only expose subjects to a single stimulus type. The most common example is "resting-state" (i.e. 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. **Although models are available in PRONTO, Single-Condition optimization properties are less well understood and the subject of ongoing research – proceed at your own risk!**

**\* NB**: if you are analyzing blocked or event-related designs and have multiple task contrasts of interest, you can find the optimal pipelines that gives best average (Prediction, Reproducibility) for all contrasts at once – refer to Table 6 ( “--contrast” option) and Section 6 for further details.

The table below tells you which analysis models are recommended for pipeline optimization. It depends on which of the above 3 categories your task design falls into (Block, Event-Related, Single-Condition), and whether you want to conduct univariate or multivariate analysis. Univariate models measure regional BOLD activations and tend to be less sensitive to pipeline choices (good and bad); multivariate models account for functional connectivity between brain regions and tend to be more sensitive to pipeline choices (good and bad). Neither approach is “right” or “wrong”, it just depends on what you are interested in measuring.

**Table 5. available analysis models in the PRONTO framework**

|  |  |  |
| --- | --- | --- |
|  | Univariate, Independent Voxels | Multivariate, Covarying Voxels |
| **Block Design**  (TYPE=”block”) | **GNB** (Gaussian Naïve Bayes)  predictive general linear model of difference between 2 conditions | **LDA** (Linear Discriminant Analysis)  predictive multivariate model of differences between 2 conditions |
| **Event-Related**  (TYPE=”event”) | **erGNB** (Event-Related GNB)  predictive general linear model, treating each time-point in HRF as a different condition | **erCVA** (Event-Related CVA)  predictive multivariate model, treating each time-point in an HRF window as a different condition |
| **Single Condition**  (TYPE=”nocontrast”) | **SCONN** (Seed-based Connectivity)  measures pairwise correlations of all voxels, relative to average signal in a seed Region of Interest (ROI) | **gPCA** (Generalized PCA)  Principal Component Analysis decomposition of fMRI dataset, identifying most consistent subspace |

\* The PRONTO code also has standard General Linear Model (GLM) and event-related GLM (erGLM) models available, but we do not recommend them: they can only be used to measure Reproducibility, whereas Prediction accuracy cannot be computed. If you want to analyze your data assuming independent voxels, as in GLM, we recommend using **GNB** or **erGNB**, which are the predictive versions of these models.

\*NB: The baseline condition has to be defined, when the GLM analysis is used (See the previous subsection).

## iv. Running Individual Pipelines

Once you have located the raw fMRI data, created your input files and chosen your analysis model, you are ready to begin pipeline optimization.

PRONTO is designed to run using the high-performance Sun Grid Engine (SGE). This is a distributed resource manager, which allows datasets to be submitted as individual jobs to a computing cluster, greatly accelerating computational speed. This is all executed by running the wrapper “**Run\_Pipelines.py**”. The command line structure is defined below, along with important outputs (see Appendix for further details). You can still run PRONTO without SGE in (almost) any unix-like environment, but it will be much slower. This may be a significant problem with modest computational resources and anything but small numbers of subjects with small numbers of scans.

**Preparatory details**:

* Users should contact Rotman IT for username and password access to the SGE system
* Open a terminal and log onto the “gateway” server, by typing: “**ssh {username}@172.24.4.65**”, and then “**{password}**”
* Access the “headnode” server by typing “**ssh headnode**”
* Check that both AFNI and FSL software packages are in your path
* Make sure PRONTO folders are in the current working directory along with {subject input} and {pipelines} files
* Confirm raw fMRI data, physio data, and {task information} files are in the directories listed in your {subject input} file

When the scripts are run on SGE system, each line in the {subject input}.txt file is submitted as a job via “qsub” command. Every job has an ID number, used to track progress of the 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. If you see an error (or data appears to be missing), you can check these log files to see what happened. Typing “qstat –f” in the command line will show jobs that are currently being processed as well as pending jobs, identified by job number.

**Input commands**:

Once you have created your {subject input}.txt and {pipeline list}.txt files, along with the necessary “split\_info”.mat files for each dataset, type the following:

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

We have already reviewed how to write the {subject input}.txt and {pipeline list}.txt files in section 2(iii). You will want to specify additional flags, the most important being **–a** and **–r**, but refer to Tables 5 & 6 for special options that may be useful if you want to do advanced analysis

The flag **“-a {analysis model}”** specifies the chosen analysis method (e.g. “**-a LDA**” performs linear discriminant analysis). Options are described in detail in the previous section, and include **GNB**, **LDA**, **erGNB**, **erCVA**, **SCONN** and **gPCA**. If you don’t want to (or can’t) do analysis, type “**-a NONE**”; this will just output all processing pipeline combinations specified in {pipeline\_list.txt}. **WARNING: if you choose this option, it will create a preprocessed dataset for every pipeline combination. Make sure it is a short list of options, as this will quickly eat up all disk space!**

The flag “**-r {reference\_image.nii}**” specifies the path+name of a reference template volume (e.g. MNI, ICBM or Talairach atlas); all optimized results will be spatially normalize to match this volume. **If you omit this flag, or don’t include “STRUCT=…” fields in the {subject\_input.txt} file, spatial normalization step will be skipped.**

The **[other arguments]** allows you set other optional flags for PRONTO, giving you greater control over pipeline inputs/outputs. Refer to the tables 5 & 6 below to see what is available.

**Output results**:

After completing pipeline optimization, you will get a set of folders containing intermediate pipeline results, and final optimized outputs. If your {subject\_input.txt} has output fields **OUT**={outdir}/{outname}, ***you will see the following output structure in directory “outdir”***:

intermediate\_processed 🡪 contains semi-processed data plus other selected outputs

afni\_processed 🡪 data processed using AFNI algorithms

diagnostic 🡪 diagnostic metrics used to identify outlier volumes

masks 🡪 functional brain masks

mpe 🡪 head motion parameter estimates

spat\_norm 🡪 parameters required to do spatial normalization

split\_info 🡪 stores copies of split\_info data, describing task structure

intermediate\_metrics 🡪 contains metrics of data quality and analysis results

res0\_params 🡪 misc. parameters, e.g. analysis model type, artifact measures

res1\_spms 🡪 activation maps for all pipelines (stored in 2D matrix)

res2\_temp 🡪 BOLD timeseries of activation maps (in 2D matrix)

res3\_stats 🡪 statistics of pipeline quality (Prediction, Reproducibility)

regressors 🡪 task and nuisance covariates for each pipeline model

optimization\_results 🡪 contains optimally-processed fMRI data and brain maps

processed 🡪 processed 4D fMRI runs (for STD, FIX and IND pipelines)

spms 🡪 optimal activation maps (STD, FIX and IND concatenated)

matfiles 🡪 activation maps and quality metrics, stored in .mat files

All important outputs are highlighted in red; the rest are just intermediate results that are used by the PRONTO code during pipeline optimization. If you are pressed for disk space after optimization, you can delete the “intermediate\_processed” and “intermediate\_metrics” folders.

The results of optimization are created as follows. If, for example, you have a dataset with the following output field in {subject input}.txt : “**OUT**=new\_dir/subj\_1”, you get the following output:

new\_dir/optimization\_results/processed

Proc\_ subj\_1\_STD.nii

Proc\_ subj\_1\_FIX.nii

Proc\_ subj\_1\_IND.nii

Proc\_ subj\_1\_STD\_sNorm.nii \*

Proc\_ subj\_1\_FIX\_sNorm.nii \*

Proc\_ subj\_1\_IND\_sNorm.nii \*

new\_dir/optimization\_results/spms

rSPM\_ subj\_1\_STD\_FIX\_IND.nii

rSPM\_ subj\_1\_STD\_FIX\_IND\_sNorm.nii \*

Where “Proc…” represents the preprocessed 4D fMRI datasets, under STD (standard), FIX (fixed) and IND (individually optimized) pipelines. The “rSPM…” is a set of reproducible Z-scored SPMs produced by STD, FIX and IND pipelines, concatenated along the time (4th) dimension.

Outputs with a ‘\*’ are only produced if spatial normalization is included. Datasets without “sNorm” are in native subject space, while “sNorm” indicates the corresponding dataset, warped into common template space.

If you run spatial normalization, a set of group-level masks are also created in a “GroupMasks” folder; it is placed in the output path of the first dataset in {subject input}.txt. This includes:

group\_consensus\_mask.nii 🡪 a binary mask of brain tissue voxels

group\_mean\_NN\_WM.nii 🡪 concatenated probabilistic tissue masks: non-neuronal (NN), white matter (WM)The Tables below displays all arguments available when calling run\_pipeline.py. This includes some options that can be applied to all task types (Table 6), and options that may only apply to specific analysis models (Table 7). We also have a few options that are available for group-level analysis (Table 8). **Only arguments in grey (Table 6) are critical for running pipelines – all others are strictly optional, and can be ignored unless you have specialized analysis plans.**

**Table 6**. **general** **arguments available when running pipelines**

|  |  |
| --- | --- |
| **Switch** | **Description** |
| **-h, --help** | **show help message and exit** |
| **-i FILE.txt,**  **(--inputdata=FILE.txt)** | **Path and name of the {subject input}.txt file** |
| **-c PIPELINE.txt,**  **(--pipeline PIPELINE.txt)** | **Path and name of PIPELINE.txt, with list of pipeline steps being tested** |
| **-a ANALYSIS,**  **(--analysis=ANALYSIS)** | **Chosen analysis model (LDA, GNB, GLM, erCVA, erGNB, erGLM, SCONN, NONE)** |
| **-r REFERENCE,**  **(--reference=REFERENCE)** | **Path and name of reference volume used in (optional) spatial normalization step** |
| **-p PART**  **(--part PART)** | **Only do specific parts of full optimization pipeline.**  **1= run all pipelines and produce metrics**  **2= select optimal pipelines based on metrics (must have already run part-1)**  **3= do spatial normalization of optimized results (must have already run parts-1,2)**  **0= do all three steps (this is the default)** |
| **--convolve VALUE** | **VALUE=Binary value, determines whether user-provided design matrix (in split\_info) should be convolved with the canonical HRF modeled by two gamma functions (AFNI’s “SPM1” function), for TASK pipeline step and GLM analysis. 0 = do not convolve and 1 = perform convolution (default =0)** |
| **-m METRIC,**  **(--metric=METRIC)** | **Metric used to choose optimal pipelines standard options include**  **‘R’= reproducibility ‘P’= prediction**  **‘dPR= combined prediction & reproducibility (this is the default)** |
| **--dospnormfirst** | **First normalize the data to a reference (specified by switch -r), then perform the preprocessing optimization (not recommended unless you have good reason to do this; greatly increases compute time).** |
| **--contrast=CONTRAST** | **Task contrast being analyzed and optimized for. Necessary when more than two conditions 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)** |
| **-n NUMCORES,**  **(--numcores=NUMCORES)** | **Number of threads used for each job (not effective for all SGE systems)** |
| **-q QUEUE,**  **( --queue=QUEUE)** | **SGE queue name, default is bigmem\_16.q (For HPCVL SGE, used –q abaqus.q).** |
| **-k KEEPMEAN,**  **(--keepmean=KEEPMEAN)** | **Determines whether the output nifti files have temporal means re-added to each voxel after processing (default KEEPMEAN=0, removes the mean scan)** |
| **-v VOXELSIZE,**  **(--voxelsize=VOXELSIZE)** | **Determines the output voxel size of nifti file.**  **Syntax: -v “[3.0 3.0 5.0]” gives 3x3x5 mm voxels as final results. Default is to keep output voxels the same size as input.** |
| **-e ENVIRONMENT** | **Determine which software to use to run the code: matlab or octave(default)** |
| **--DEOBLIQUE** | **Corrects for raw data that are at an oblique angle relative to the cardinal scanning axes, using AFNI’s 3dWarp program. May improve the quality of registration** |
| **--TPATTERN** | **Defines axial slices acquisition order for slice-timing correction, if this information is not available in the header. Options include:**  **‘altplus’ (alternating, + direction – standard interleaved protocol)**  **‘altminus’ (alternating, negative direction)**  **‘seqplus’ (sequential, positive direction)**  **‘seqminus’ (sequential, negative direction)** |

**Table 7. optional arguments that are specific to individual analysis models.**

|  |  |  |
| --- | --- | --- |
|  | **Switch** | **Descriptions** |
| **GLM** |  | **This model does not have any specific switch** |
| **GNB** | **--decision\_model MODEL** | **MODEL is a string specifying type of decision boundary. Either: ‘linear’ for a pooled covariance model or ‘nonlinear’ for class-specific covariances (default is ‘linear’)** |
| **LDA** | **--drf FRACTION** | **FRACTION is a scalar value of range (0,1), indicating the fraction of full-date PCA subspace to keep during PCA-LDA analysis (default is 0.5)** |
| **erGLM** |  | **This model does not have any specific switch** |
| **erGNB** | **--Nblock NUMBER** | **NUMBER is the number of equal-sized blocks to break the data into, to perform time-locked averaging. Must be at least 4 (the default) to obtain robust covariance estimates.** |
| **--WIND SIZE** | **SIZE is the window size to average on, in TR (usually in range 6-10 TR). Default is 10.** |
| **erCVA** | **--Nblock NUMBER** | **NUMBER is the number of equal-sized blocks to break the data into, to perform time-locked averaging. Must be at least 4 (the default) to obtain robust covariance estimates.** |
| **--WIND SIZE** | **SIZE is the window size to average on, in TR (usually in range 6-10 TR). Default is 10.** |
| **--drf FRACTION** | **FRACTION is a scalar value of range (0,1), indicating the fraction of full-date PCA subspace to keep during PCA-LDA analysis (default is 0.5)** |
| **--subspace COMP** | **COMP is a string specifying either: 'onecomp' = only optimize on CV#1 or 'multicomp' = optimize on full multidimensional subspace. Default is ‘onecomp’** |
| **SCONN** | **--spm FORMAT** | **FORMAT is a string specifying format of output SPM. Options include ‘corr’ (map of voxelwise seed correlations) or ‘zscore’ (Z-scored map of reproducible correlation values). Default is ‘zscore’** |

**Table 8. optional arguments that are available for group-level analysis**

|  |  |
| --- | --- |
| **--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 re-sampling splits to perform (and average over). A minimum of 100 resamples is recommended for stable group-level estimates** |

### EXAMPLES OF PIPELINE JOBS

1. Lets 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 PMT, 2) DMS vs RT. The syntax for submitting all three parts is as follows:

**./Run\_Pipelines.py –i /example/input.txt –c /example/param.txt –a LDA –r /example/MNI152\_1mm.nii –v 4.0 --contrast ‘ATT+DMS-PMT,DMS-RT’**

The pipeline will optimize for both ATT and DMS vs PMT, and DMS vs RT task contrasts. Note that we used a special option of “-v 4.0” (Table 6) which forces the spatially normalized fMRI data to have voxel resolution of 4x4x4mm. This would be omitted if we wanted to keep the same voxel size as in the raw data.

2. Let assume the goal is to optimize the preprocessing of an event-related dataset using the erCVA analysis models, and we do not want to perform spatial normalization. The syntax is as follows:

**./Run\_Pipelines.py –i /example/input.txt –c /example/param.txt –a erCVA --WIND 6**

Unlike in example #1, we did not specify the reference image using –r. Then the program skips this step, and only produces optimized data in the subjects’ native space. In addition, we did not specify --contrast. Consequently the program optimizes for the condition 1 vs. condition 2.

Finally, note that we again used a special option, of “--WIND 6” (Table 7). This changes the window size of the estimated HRF in erCVA analysis, to 6TR. If we omitted this option, it would give the default of 10TR.

### 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 sample QC output.

**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 )

Output is saved to “QC1\_results” folder, in output path of first dataset in {subject input}.txt

**Quality Control-2**: to be run after completing PART-2 ***and*** spatial normalization (see **Appendix C 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; (5) group-level PCA results for each pipeline.

Syntax:

**Pipeline\_QC2**({subject input}.txt, [], numpcs)

Where fields are:

1. **{subject input}.txt:** is the name of the input file for pipeline optimization
2. **[]:** this field can define the name of group mask, if you have a specialized one. Leaving it empty uses the default one created by the pipeline
3. **numpcs:** subject SPMs are examined using Principal Component Analysis. This field is an integer specifying how many PCs to keep for further analysis. . Default (and recommended minimum) is numpcs=2. This allows you to see the “mean SPM pattern” across subjects in PC#1, if it exists, and the largest source of pattern variability in PC#2.

Output is saved to “QC2\_results” folder, in output path of first dataset in {subject input}.txt

# 5. Group-Level Pipeline Optimization

It is possible to perform the pipeline optimization at the group-level (e.g. across multiple task runs for a single subject, or across multiple subjects). This should be used if you are interested in directly optimizing analyses that involve multiple fMRI runs. It requires no special modifications of your {subject input}, {pipelines} and {task information} files.

Simply use the --autodetect switch (Table 8) when executing Run\_Pipeline.py. Instead of analyzing individual fMRI runs (i.e. lines of {subject input} textfile) separately, the PRONTO software automatically combines them into a group analysis, for each pipeline being tested. In this case, the NPAIRS resampling is conducted across runs. The switch --N\_resample (Table 8) must be used to specify the number of re-sampling splits.

# 6. Optimizing for Multiple Task Contrasts

Independently Optimizing for each task contrast increases computational time, and the required storage space. In addition, it results in several sets of preprocessed data, which make it difficult for researcher to generate a consensus dataset. PRONTO provides the capability of optimizing for several task contrasts simultaneously. To use this feature, all desired the task contrasts have to defined in the command line using the "--contrast" option.

The pipelinegives best average (Prediction, Reproducibility) over all contrasts, using the “--contrast" option, in Table 6). For example, if you have 3 conditions CON1,CON2,CON3 and you just wanted to optimize CON1 vs. CON2, you would use the option:

“--contrast CON1-CON2”

If you wanted to optimize for all possible paired contrasts (CON1 vs CON2, CON1 vs CON3, CON2 vs CON3), you would instead use the option:

“--contrast CON1-CON2,CON1-CON3,CON2-CON3”

which would gives a single preprocessed dataset, along with optimized SPMs for each of the 3 different contrasts in your outputs.

# 7. 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

Schmah T, et al. (2010): Comparing classification methods for longitudinal fMRI studies. Neural Comput, 22:2729-62.

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.

Strother S et al. (2014): Stability and Reproducibility in fMRI Analysis, in Practical Applications of Sparse Modeling, I. Rish, G. A. Cecchi, A. Lozano, and A. Niculescu-Mizil, Eds., pp. 99-121, Boston: MIT Press.

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

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 key scripts that are called during pipeline optimization:

* + 1. Pipeline\_PART1\_afni\_steps.m
    2. Pipeline\_PART1.m
    3. Pipeline\_PART2.m

The following sections describe in more detail how each underlying script functions.

**Pipeline\_PART1\_afni\_steps.m:** this script is able to perform the following steps:

**Fixed steps**

1. Remove non-equilibrium/instructional scans, specified in {subject input}.txt
2. Generate coarse brain mask on unregistered data via3dAutomask (AFNI)
3. Identify minimum-displacement fMRI volume using PCA (i.e. the volume with minimum average Euclidean distance from all others in the run), then run MOTCOR using 3dvolreg (AFNI), with this volume chosen as the reference. This produces motion-corrected fMRI data and Motion Parameter Estimates (MPEs)
4. Run diagnostic script on fMRI data (with and without MOTCOR), producing .mat files containing diagnostic information, including indices of potential head motion spikes

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

1. Correct for oblique scan-volumes using 3dWarp (AFNI)
2. Retain data, with and without MOTCOR applied
3. Remove head motion spikes, including full volumes, and component-based (CENSOR)
4. Perform RETROICOR using 3dRetroicor (AFNI)
5. Perform TIMECOR using 3dTshift (AFNI)
6. Perform SMOOTH for specified kernel(s) using 3dmerge (AFNI)

**Pipeline\_PART1.m:** this script is able to perform the following steps:

**Fixed Steps**

1. Obtain maps of brain edges, used to detect motion artifact
2. Generate mask of non-neuronal (vascular/ventricle regions) and downweight them

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

1. Run specified DETREND orders
2. Perform MOTREG using the MPE estimates from “Pipeline\_PART1\_afni\_steps.m”
3. Include task design as part of the GLM model (TASK)
4. Estimate and remove Principal Component #1, a potential global signal confound (GSPC1)
5. Physiological regression using PHYCAA+ (PHYPLUS)
6. Low-pass filtering of BOLD time-series (LOWPASS)

**Fixed Final Step:** run split-half NPAIRS analysis on preprocessed data, generating brain maps and performance metrics

**Pipeline\_PART2.m:** this script performs the following steps:

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

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 (CON/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 optimally preprocessed 4D data

Note that this script does optimization based on the metric you choose to optimize, which means you can choose from all fields in the “METRIC\_set” output. For example, ‘LDA’ analysis model has options of {‘P’,’R’,dPR}, where ‘dPR’ is the default option (see Table ?).

In addition, pipeline optimization is able to automatically exclude pipelines with excess motion artifact or excess global signal confound. The current default is to exclude motion artifact, but **not** control for global signal, as it is unclear whether this should be removed in all cases. This is controled using the 2D binary input **mot\_gs\_control** (see Table ?), defined as:

* 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**

# Appendix C: Spatial Normalization Script Details

This section describes how to use the **spatial\_normalization**.m script used to co-register all subject data into a common anatomical space. Inter-subject alignment is performed using a 2-stage transformation process:

1. Non-rigid (12-parameter affine) warp of subject anatomical volumes (T1) to an anatomical reference template (e.g. Talairach, MNI or ICBM)
2. Rigid-body within-subject alignment of functional (EPI) to their anatomical (T1) data. This is used for group-level analysis across SPMs

Spatial normalization should only run on the optimally processed datasets, as it is a highly compute-intensive process.

**Spatial normalization steps**:

1. Mask subject T1 brain volumes using bet (brain extraction tool; FSL)
2. Compute 12-parameter affine transformation of each T1 volume to the user-specified group template using flirt algorithm (FSL), producing affine transformation matrix
3. Compute the 7-parameter affine (rigid-body + scaling) alignment of fMRI data to the subject’s corresponding T1 volume, producing affine transformation matrix
4. Compute the net transformation matrix from the above two steps (fMRI 🡪 T1 🡪 template), by multiplying these matrices
5. Spatially transform 4D fMRI dataset with basic preprocessing (called “baseproc”) using transform. matrix of Step #4, in order to build group-level functional brain masks
6. Spatially transform optimized SPMs, and optimally processed fMRI datasets, with transform. matrix computed in Step #4
7. Produce downsampled structural T1, to use as reference.

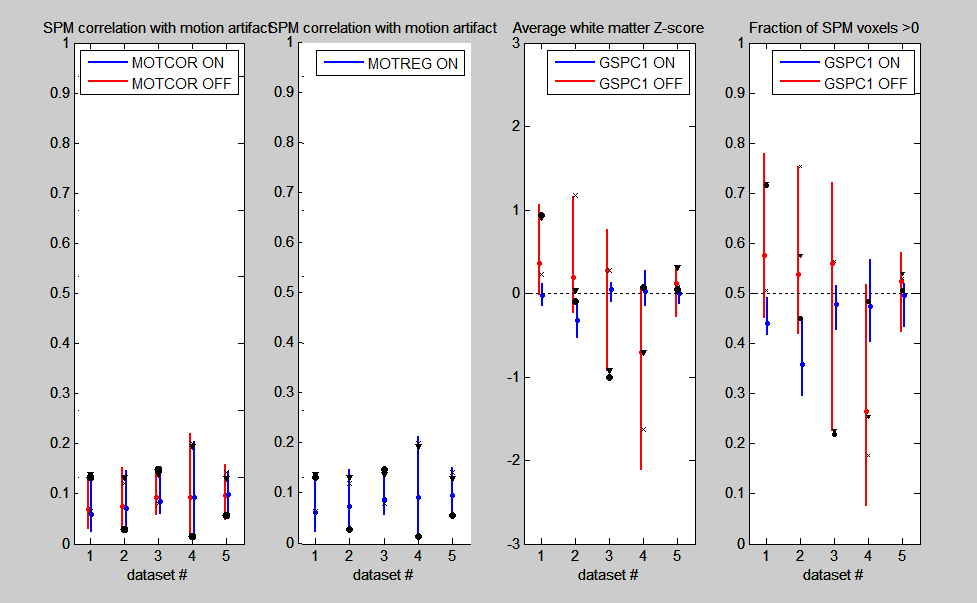
# Appendix D: Sample Quality Control Figures

## i. Quality Control-1

## 

**FIG1\_motion\_statistics:** This plot provides some statistics on head motion, for each dataset being optimized (columns correspond to lines in your {subject input} file. 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.

\* If you notice pipelines with consistently high motion amplitude (e.g. >0.05 mm/deg) and strong task correlations (e.g. >0.3), or lots of head motion spikes (e.g. in both MPE and fmri, for >10% of timepoints) examine their data carefully to make sure there are no persistent motion artifacts.

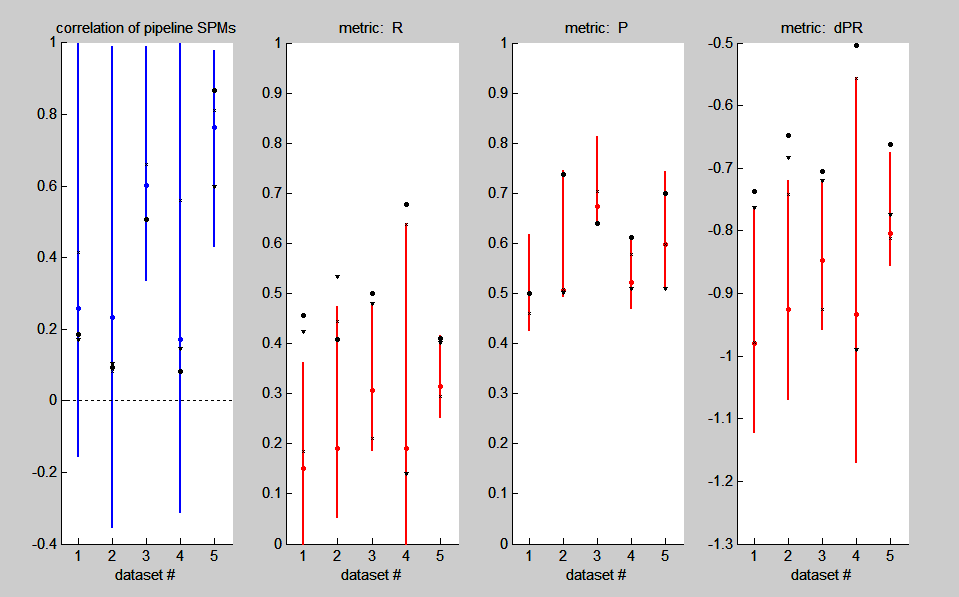


**FIG2\_SPM\_artifact:** This plot shows potential artifact present in SPMs, for each dataset being optimized. Vertical bars show distribution across all tested pipelines, per dataset. Going left to right, Panel-1 shows correlation with motion artifact, estimated by spatial gradient maps (for pipelines with/without MOTCOR), and Panel-2 shows the same (for pipelines with/without MOTREG). Panel-3 plots fraction of signal >0 (“globalness” of signal), and Panel-4 plots average Z-score in white matter tissues. The latter two panels are shown for pipelines with/without GSPC1 which can sometimes control for these confounds.

\* note that if you leave MOTCOR, MOTREG or GSPC1 fixed (on or off) you will only see a single set of bars. In this example, we fixed MOTREG on, which you can see in Panel-2.

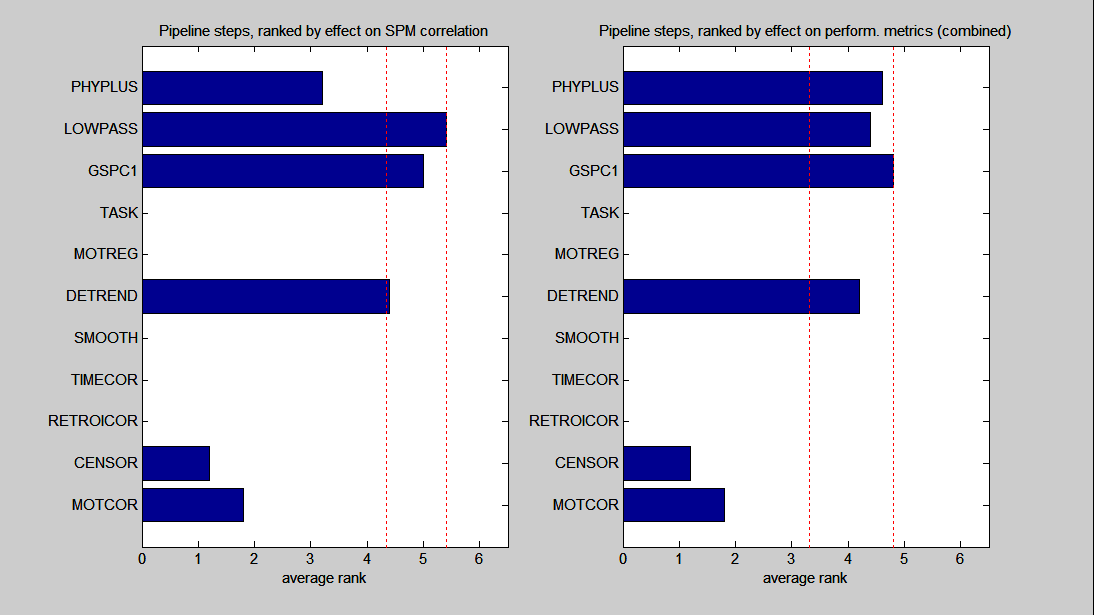
\* Panels 1,2 should give you an idea of how much motion artifact is present in your SPMs, and whether motion correction has a significant impact on the SPM patterns of your data.

\* Panels 3,4 will help to determine if there is persistent global/white matter signal in your data, and whether the current pipelines can fully correct it



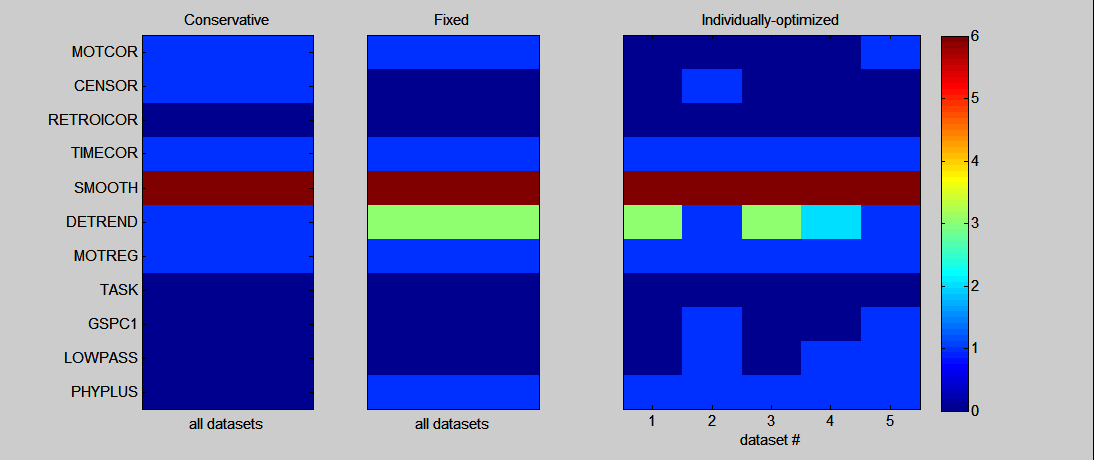
**FIG3\_pipeline\_similarity\_by\_dataset:** This plot shows how much results are altered by different pipeline choices, for each dataset being optimized. Vertical bars show the 95% CI distribution across all tested pipelines, per dataset. Left: pair-wise correlation between all pipeline SPMs. This indicates how similar the SPMs are across pipelines. For example, SPMs of dataset#4 are highly sensitive to pipeline choice (wide range of correlation values, many are negative), whereas SPMs of dataset#5 are relatively insensitive to pipeline choice (consistently high correlations).

All other panels show distribution of NPAIRS metrics across pipelines, with black icons indicating CON (‘X’), FIX (triangle) and IND (circle).

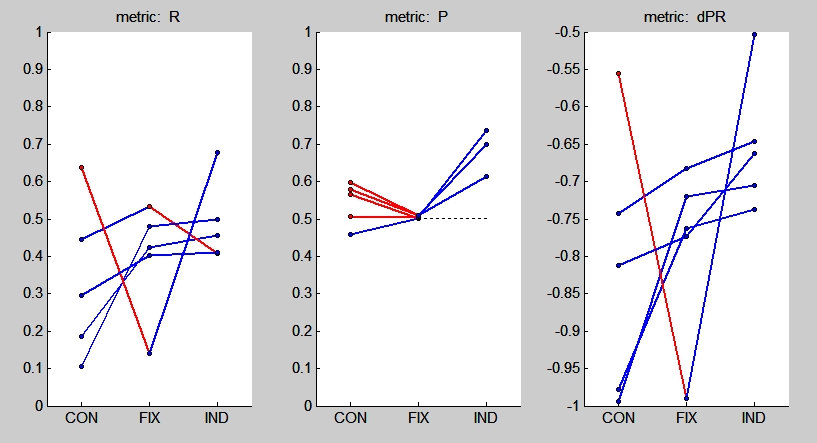


**FIG4\_effects\_of\_pipeline\_steps:** This bar chart depicts the relative influence of individual pipeline steps on results, with higher rank indicating a more influential step; bars that fall between the dashed red lines are the most influential steps for this dataset (Friedman rank test). Panel-1 indicates effect on SPM pattern (based on correlation distance between pipelines with/without each step). Panel-2 indicates effect on performance metrics (absolute change in (Prediction, Reproducibility) between pipelines with/without each step).

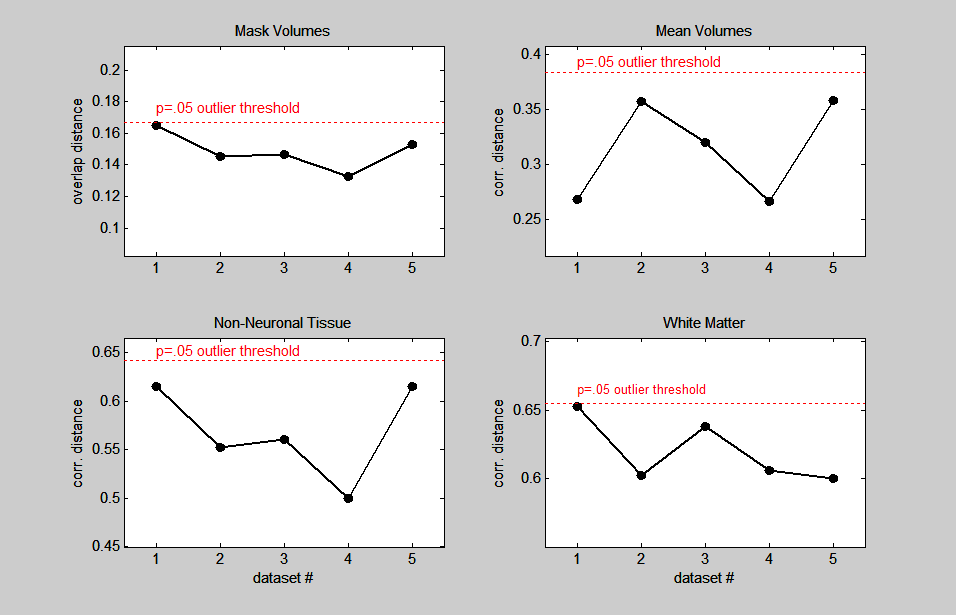
## ii. Quality Control-2



**FIG1\_optimized\_pipeline\_steps:** List of pipeline choices for each optimization strategy; for the IND pipelines (right panel), pipeline choices are listed for each dataset. For 8/11 pipeline steps, 1=ON 0=OFF. Current exceptions are CENSOR (0=none, 1=basic, 2=aggressive(PCA), 3=aggressive(ICA)), 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.



**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).

\* Some further details:

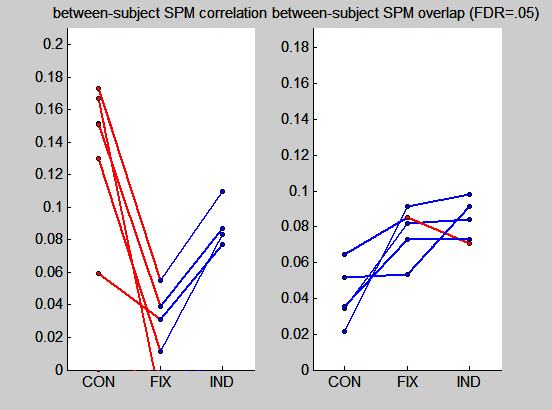
-for generating masks and spatial transformation matrices, I use a single basic pre-processing pipeline (motion correction, slice timing, 6mm smoothing), with is produced as part of the intermediate pipeline outputs (has a "baseproc" suffix).

-we generate masks by applying 3dAutomask to the transformed EPI "baseproc" data from each subject, and mean volumes" are subjects' masked "baseproc" pipelines.

- “overlap distance” is the average (1 – [jaccard overlap]) of each subject map relative to all others

- “correlation distance” is the average (1 – [correlation]) of each subject map relative to all others

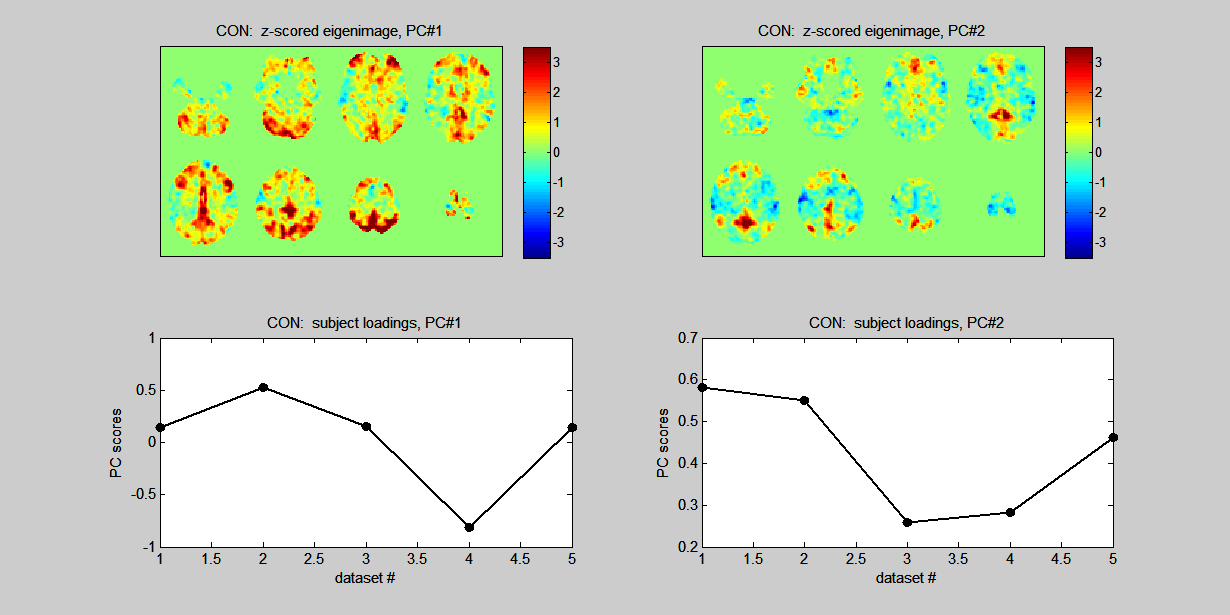
- significance is estimated by fitting a maximum-likelihood Gamma distribution to the data, and identifying and subjects with *p*<.05. This is going to be a bit conservative, since the distribution is fitted to the subjects themselves, but if there is an outlier, you can be more confident that there is a potential abnormal normalization result that should be checked out

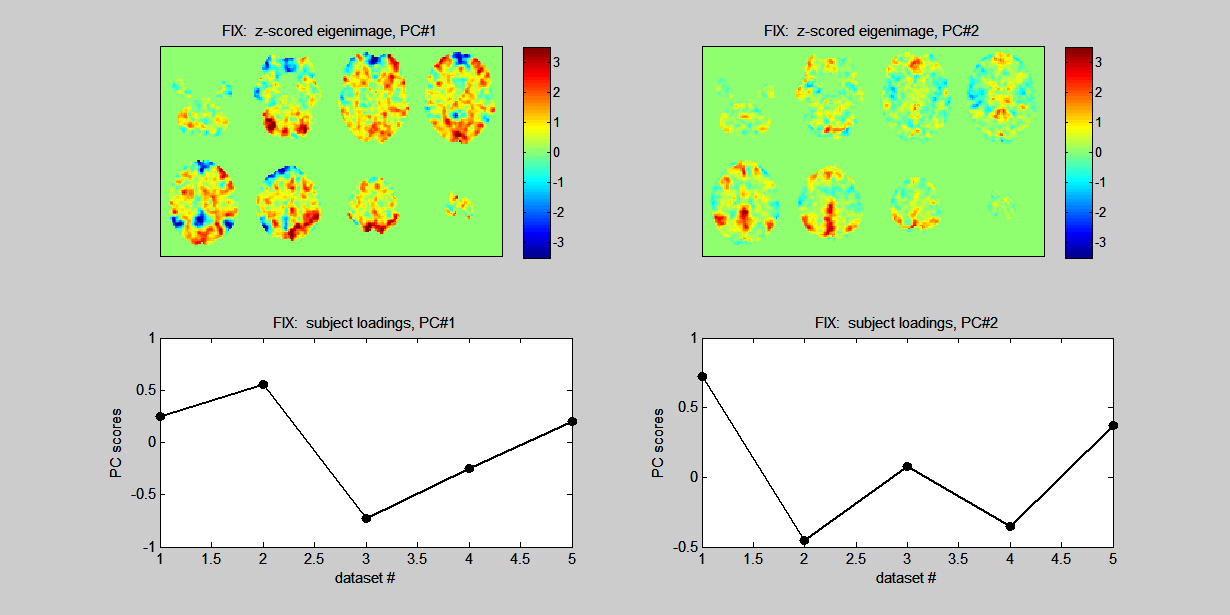


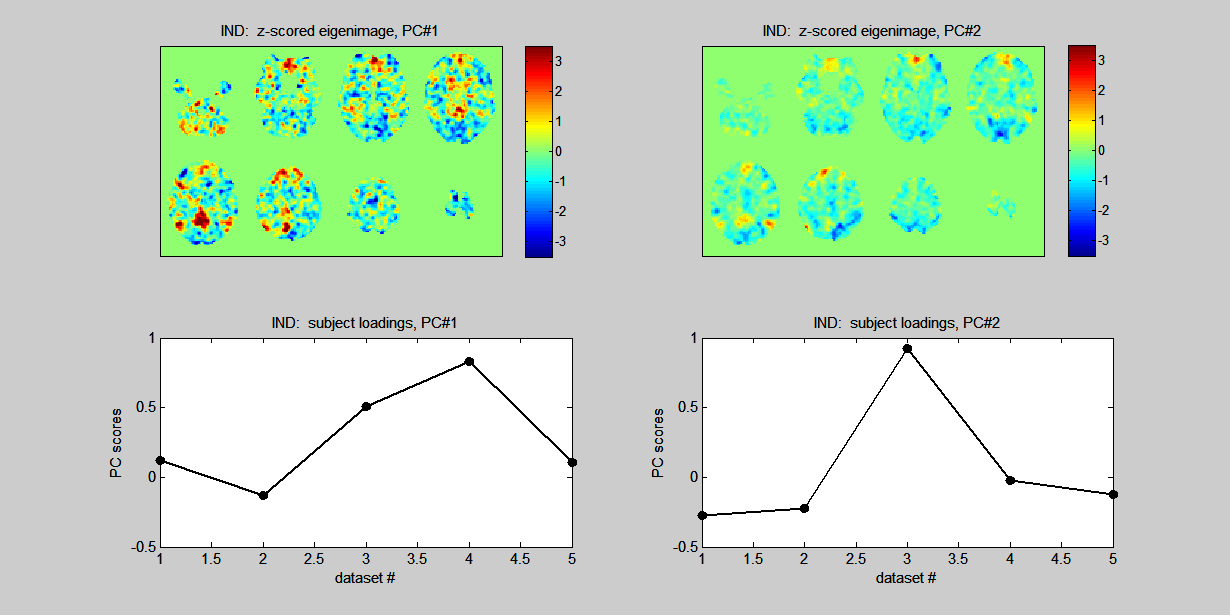
**FIG4\_inter\_subject\_SPM\_similarity:** inter-subject reliability metrics of (left) SPM correlation and (right) overlap of thresholded SPMs, as a function of pipeline optimization method, plotted for each dataset. SPMs are thresholded at False-Discovery Rate FDR=.05

**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.

\* The PCA results show “networks” of brain regions that tend to be found in subject SPMs. Subject loadings indicate how strongly this network is present in each subject.







# Appendix E: Details of Split-half Pipeline Optimization

Every single pipeline for all subjects and all runs are put through the NPAIRS (Nonparametric Prediction, Activation, Influence, and Reproducibility reSampling) framework (Strother et al., 2002), using the user-selected analysis model. NPAIRS uses a method called split-half resampling, where the scans from one half of a run are compared to its other half. The splits are independent of each other; one is the ‘training’ set and the other is the ‘test’ set. The user-chosen analysis method is applied to each split to generate two spatially z-scored SPMs.

NPAIRS calculates two quantitative statistics that reflect the consistency of the data across splits without having to set an SPM detection threshold, see Fig. E-1. The first is reproducibility, which is a measure of similarity between two SPMs. Reproducibility (R) in NPAIRS is obtained by measuring the correlation between all pairs of unthresholded brain voxel values between the z-scored SPMs of the two split groups. Prediction (P) is calculated as the median rate of using the first split’s analysis model to correctly predict the class of each scan in the second split with posterior Bayesian probability, and vice versa. P ranges from 0.5🡪1 and R ranges from 0🡪1, with perfect P and R both at 1.

|  |
| --- |
| Figure E-1 Reproducibility and Prediction in NPAIRS  Illustration of how reproducibility and prediction metrics are calculated by split-half resampling in the NPAIRS framework. |

### Determining the Optimal Fixed Pipeline (FIX)

An optimal fixed pipeline is the “best” fixed pipeline on average across all subjects. Following NPAIRS analysis, the R and P values are obtained and used for determining the optimal fixed pipeline. The optimal fixed pipeline is chosen using the procedure outlined by Churchill et al. (2011), see Fig. E-2:

1. Each subject’s pipeline performance is assessed via D, quantifying how close performance is to having the perfect model (R=1, P=1), see Fig. E-2A.
2. Pipelines per subject are ranked by arranging the D’s in a matrix and assigning the smallest D (closest to perfect) as the highest rank, which forms a matrix of ranks, see Fig. 2.5B.
3. The median of each subject’s ranking matrix is calculated for each pipeline across subjects. This was the ‘median rank metric’, see Fig. 2.5C.
4. Significance of the optimal fixed pipeline is tested using a Friedman test.

|  |
| --- |
| Figure E-2 Median Rank Metric  The procedure for identifying the optimal fixed pipeline across subjects in a 5 pipeline example. Euclidean distance, D, is obtained per pipeline to measure the pipeline closest to perfect R and P (A), ranked from least optimal (lower score) to most optimal (higher score) (B) for all subjects, and then median rank is obtained for each pipeline across all subjects, producing a median-rank profile (C). Adapted from Churchill et al. (2011) (Churchill et al., 2011). Copyright © 2011, John Wiley and Sons. The adaptation is by permission of the copyright holder. |

### Determining the Optimal Individual Pipeline

Pipelines that are optimal for each individual subject usually have increased R and/or P relative to most of the other pipelines, and the preprocessing steps included generally differ from the fixed pipeline (Churchill et al., 2011). The optimal individual pipeline is chosen by calculating D (mentioned above) per pipeline per subject and selecting the pipeline per subject, without reproducible spatial artifact, that produces the lowest D-value. ). Individual subjects may be prone to task-coupled motion, which generates artifact with high spatial Reproducibility. For this reason, IND optimization includes an addition step (established in Chuchill et al. (2012a)) to automatically reject pipelines corrupted with motion artifact before choosing the optimal one.