oxasl Documentation

Release 0.0.1

Martin Craig

Contents:

OXASL is a package for performing Bayesian analysis of Arterial Spin Labelling MRI data. Features of the toolbox include:

- Support for single or multi delay (inversion time) pCASL or PASL data including acquisitions with variable repeats and/or labelling durations

- Handles label-control or subtracted input data in various ordering conventions

- Calibration using the reference region or voxelwise methods

- Structural registration and output in structural space

- GM/WM Partial volume correction

- HTML summary report

Plugins are also available for:

- Vessel-encoded ASL data

- Multiphase ASL data

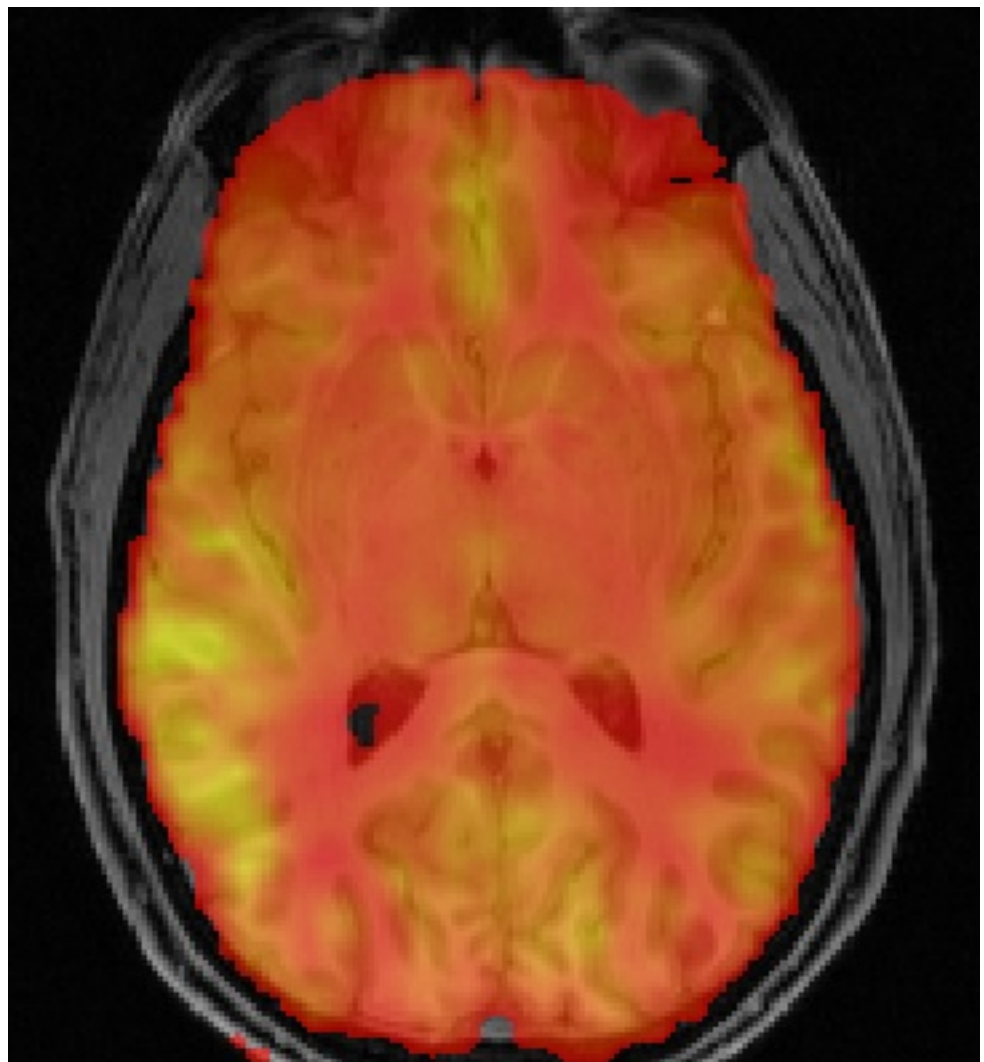OXASL works within an FSL environment which must be installed.

Contents:

CHAPTER 1

Bayesian Inference for Arterial Spin Labelling MRI

Arterial Spin Labeling (ASL) MRI is a non-invasive method for the quantification of perfusion. Analysis of ASL data typically requires the inversion of a kinetic model of labeled blood-water inflow along with a separate calculation of the equilibrium magnetization of arterial blood.

The OXASL toolbox uses the FSL FABBER ASL package which performs the kinetic modelling using Bayesian inference principles. The package was orginally developed for multi delay (inversion time) data where it can be used to greatest effect, but is also sufficiently flexible to deal with the widely used single delay form of acquisition.

Note: If you want to perform analysis of a functional experiment with ASL data, i.e. one where you want to use a GLM, then you should consult the perfusion section of FEAT, or if you have dual-echo (combined BOLD and ASL) data then consult FABBER.

For single delay ASL data kinetic model inversion is relatively trivial and solutions to the standard model have been described in the literature. However, there are various advantages to aquiring ASL data at multiple times post-inversion and fitting the resultant data to a kinetic model. This permits problems in perfusion estimation associated with variable bolus arrival time to be avoided, since this becomes a parameter of the model whose value is determined from the data. Commonly the model fitting will be performed with a least squares technique providing parameter estimates, e.g. perfusion and bolus arrival time. In contrast to this BASIL uses a (fast) Bayesian inference method for the model inversion, this provides a number of advantages:

- Voxel-wise estimation of perfusion and bolus arrival time along with parameter variance (allowing confidence intervals to be calculated).

- Incorporation of natural varaibility of other model parameters, e.g. values of T1, T1b and labeling/bolus duration.

- Spatial regularization of the estimated perfusion image.

- Correction for partial volume effects (where the appropriate segmentation information is available).

While the first two apply specfically to the case of mulitple delay data, the latter are also applicable to single delay ASL and are only available using the Bayesian technique employed by OXASL.

Getting the OXASL software

To use OXASL you will need FSL - version 6.0 or later is strongly recommended. See FSL installation for installation instructions.

2.1 I have FSL 6.0 or later

To install into the fslpython environment use:

```
fslpython -m pip install oxasl oxasl_ve oxasl_mp --user
```

This installs the main package and the vessel-encoding plugin. To check it is working, try running the main executable:

```
oxasl --version
```

2.2 I have an older version of FSL

You will need to download a pre-built bundle package containing the OXASL code and also the required updated FSL dependencies. This can be found on our GitHub release page:

https://github.com/physimals/oxasl/releases

OXASL walk through tutorial - command line

This tutorial demonstrates some of the common options available in the OXASL command line tool.

We will be working with multi-PLD data from the FSL tutorial on Arterial Spin Labelling. You will need to download this data before following the tutorial.

The tutorial has been written so that we start with the most basic analysis and gradually add options and show the effect they have on the output, as well as how they are reported in the command log and the summary report. However this is not a complete description of all available options - for that see the *OXASL command reference*.

3.1 The data

The data is from a multi-PLD pCASL acquisition with a bolus duration of 1.4s and post-labelling delays (PLDs) PLDs every 0.25s up to 1.5s. The acquisition was 2D with an increase in the PLD per slice of 0.0452s.

3.2 Basic run without calibration or structural data

In this case we only need specify the structure and acquisition parameters for our ASL data:

```
oxasl -i mpld_asltc --casl --iaf=tc --ibf=tis --slicedt=0.0452 \
      --plds=0.25,0.5,0.75,1.0,1.25,1.5 --bolus=1.4 \
      -o oxasl_out
```

Here the `--iaf=tc` option indicates that the data contains tag-control pairs. `--ibf=tis` indicates that the TC pairs are stored in blocks corresponding to each TI/PLD, so the first two volumes are tag/control for PLD=0.25, while the next two volumes are repeats at the *same PLD*.

If the data instead consisted of label-control pairs for *all* the PLDs followed by another block of repeated images at *all* the PLDs, this would be indicated using `--ibf=rpt` (blocks of repeats).

`-o oxasl_out` sets the output directory.

3.2.1 The log output

The command line outputs (hopefully) helpful information on it's progress - to redirect this to a file you could add >output_log to the command line above. In this case it appears as follows:

```
OXASL version: 0.0.7.dev12
- Found plugin: oxasl_ve (version 0.0.7.dev2)
- Found plugin: oxasl_deblur (version 0.0.1)
- Found plugin: oxasl_enable (version 0.0.1.dev6)

Input ASL data: asldata
Data shape                   : (64, 64, 24, 96)
Label type                   : Label-control pairs
Labelling                    : CASL/pCASL
PLDs (s)                     : [0.25, 0.5, 0.75, 1.0, 1.25, 1.5]
Repeats at each TI           : [8, 8, 8, 8, 8, 8]
Bolus durations (s)          : [1.4, 1.4, 1.4, 1.4, 1.4, 1.4]
Time per slice (s)           : 0.0452
```

This is the program reporting its current version and some useful information about the ASL data you have given:

```
Initialising structural data
- No structural data supplied - output will be ASL space only

Applying preprocessing corrections
- Data transformations
- No corrections to apply

No motion correction

Applying preprocessing corrections
- Data transformations
- No corrections to apply

No fieldmap images for distortion correction

No CBLIP images provided for distortion correction

Calculating Sensitivity correction
- No source of sensitivity correction was found

Applying preprocessing corrections
- Data transformations
- No corrections to apply
```

Next the preprocessing step is performed which involves performing any required corrections. In this case there is nothing to do, but it goes through the motions anyway:

```
Getting the ASL image to use for registration)
- Registration reference is mean ASL signal (brain extracted)

Generated ASL data mask
- Mask generated from brain extracted registration ASL image
```

Now the mask is generated - it first checks to see if there is structural data but there isn't so the mask comes from the mean ASL signal:

```
Running BASIL Bayesian modelling on ASL data
- Doing initial fit on mean at each TI

BASIL v0.0.7.dev12
Data shape                    : (64, 64, 24, 6)
Label type                    : Already differenced
Labelling                     : CASL/pCASL
PLDs (s)                      : [0.25, 0.5, 0.75, 1.0, 1.25, 1.5]
Repeats at each TI            : [1, 1, 1, 1, 1, 1]
Bolus durations (s)           : [1.4, 1.4, 1.4, 1.4, 1.4, 1.4]
Time per slice (s)            : 0.0452
Model (in fabber) is : aslrest
Dispersion model option is none
Compartment exchange model option is mix
Step 1 of 3: VB -  Tissue  100%
Step 2 of 3: VB -  Tissue  Arterial  - Initialise with step 1 100%
Step 3 of 3: Spatial VB -  Tissue  Arterial  - Initialise with step 2 100%

End

- Doing fit on full ASL data

BASIL v0.0.7.dev12
Data shape                    : (64, 64, 24, 96)
Label type                    : Label-control pairs
Labelling                     : CASL/pCASL
PLDs (s)                      : [0.25, 0.5, 0.75, 1.0, 1.25, 1.5]
Repeats at each TI            : [8, 8, 8, 8, 8, 8]
Bolus durations (s)           : [1.4, 1.4, 1.4, 1.4, 1.4, 1.4]
Time per slice (s)            : 0.0452
Model (in fabber) is : aslrest
Dispersion model option is none
Compartment exchange model option is mix
Step 1 of 3: VB -  Tissue  100%
Step 2 of 3: VB -  Tissue  Arterial  - Initialise with step 1 100%
Step 3 of 3: Spatial VB -  Tissue  Arterial  - Initialise with step 2 100%

End
```

This section is doing the actual modelling to determine the perfusion and arrival maps. The modelling is initially done on data that has been averaged at each PLD. The results of this are used to initialize a second run with all the repeats available separately. The fitting process proceeeds in three steps - the first to fit tissue parameters (perfusion and arrival), the second adds the arterial component, and the final step performs spatial regularization:

```
Generating HTML report
- Report generated in /home/ibmeuser/data/asl/fsl_course/ASL/oxasl_out/report

Output is /home/ibmeuser/data/asl/fsl_course/ASL/oxasl_out
OXASL - done
```

Finally an HTML report is generated which will be described below. This may not occur if you do not have the `sphinx-build` program installed.

3.2.2 Output images

Output images can be found in the oxasl_out/output/native directory and should contain the following files:

- `perfusion.nii.gz` - This is the relative perfusion image
- `arrival.nii.gz` - This is the inferred bolus arrival time image
- `aCBV.nii.gz` - This is the inferred macrovascular signal image containing arterial volume fraction as a percentage
- `mask.nii.gz` - This is the binary brain mask used in the analysis

The perfusion map should be viewed to ensure it looks like a perfusion image - it should show good WM/GM contrast, for example see the image below:
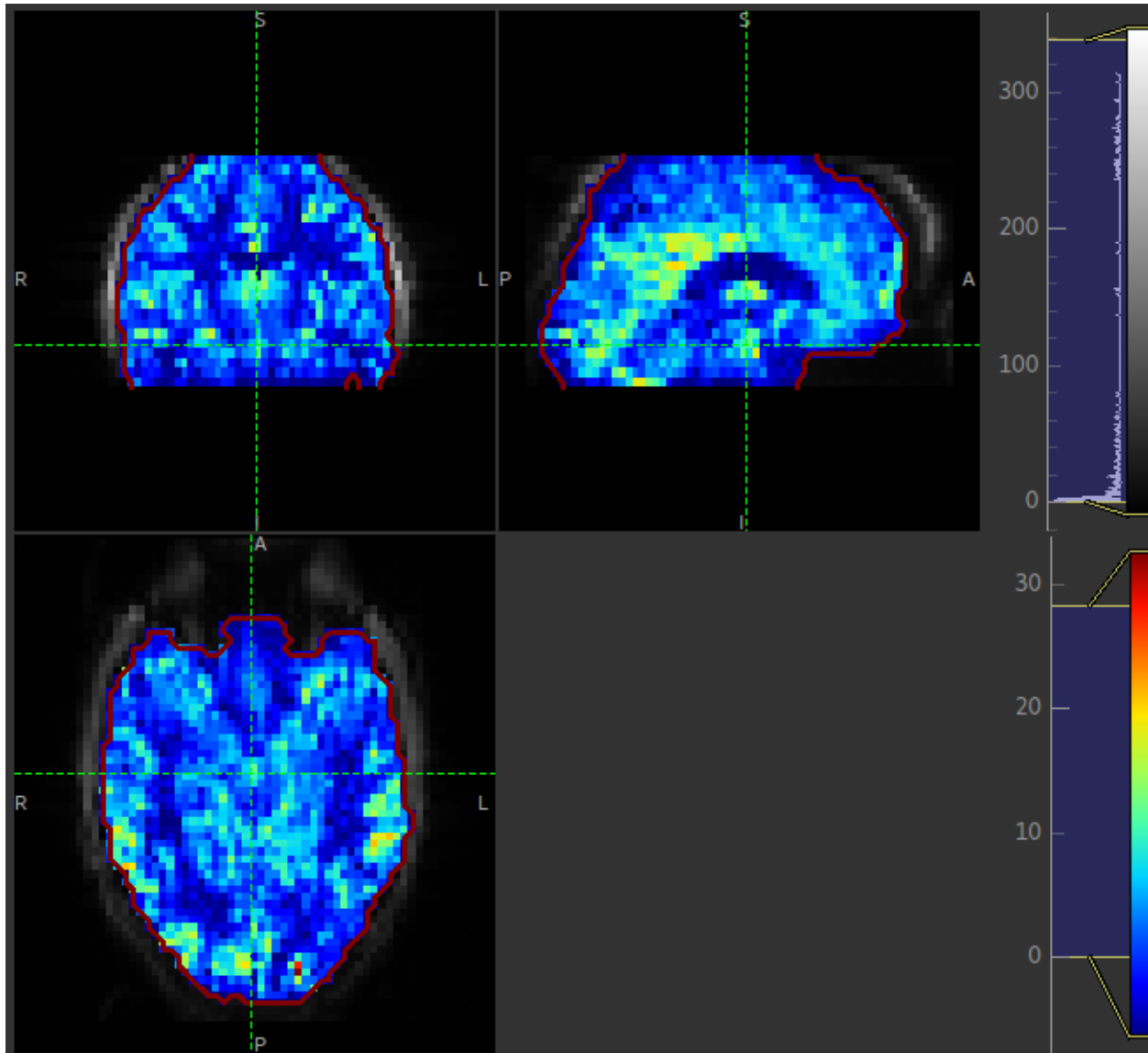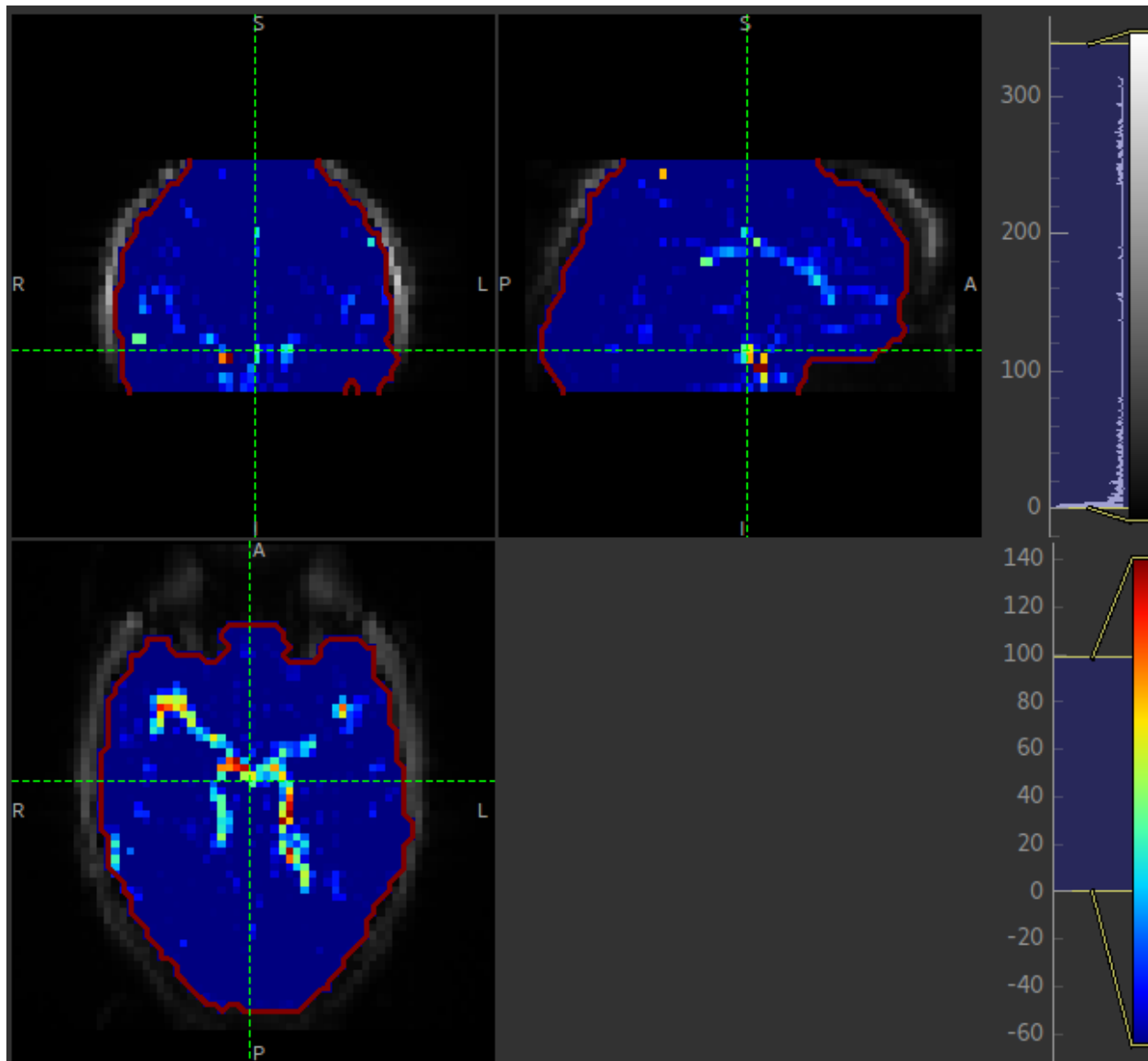

The arrival image is generally more uniform but may show delayed arrival at the posterior and superior regions:
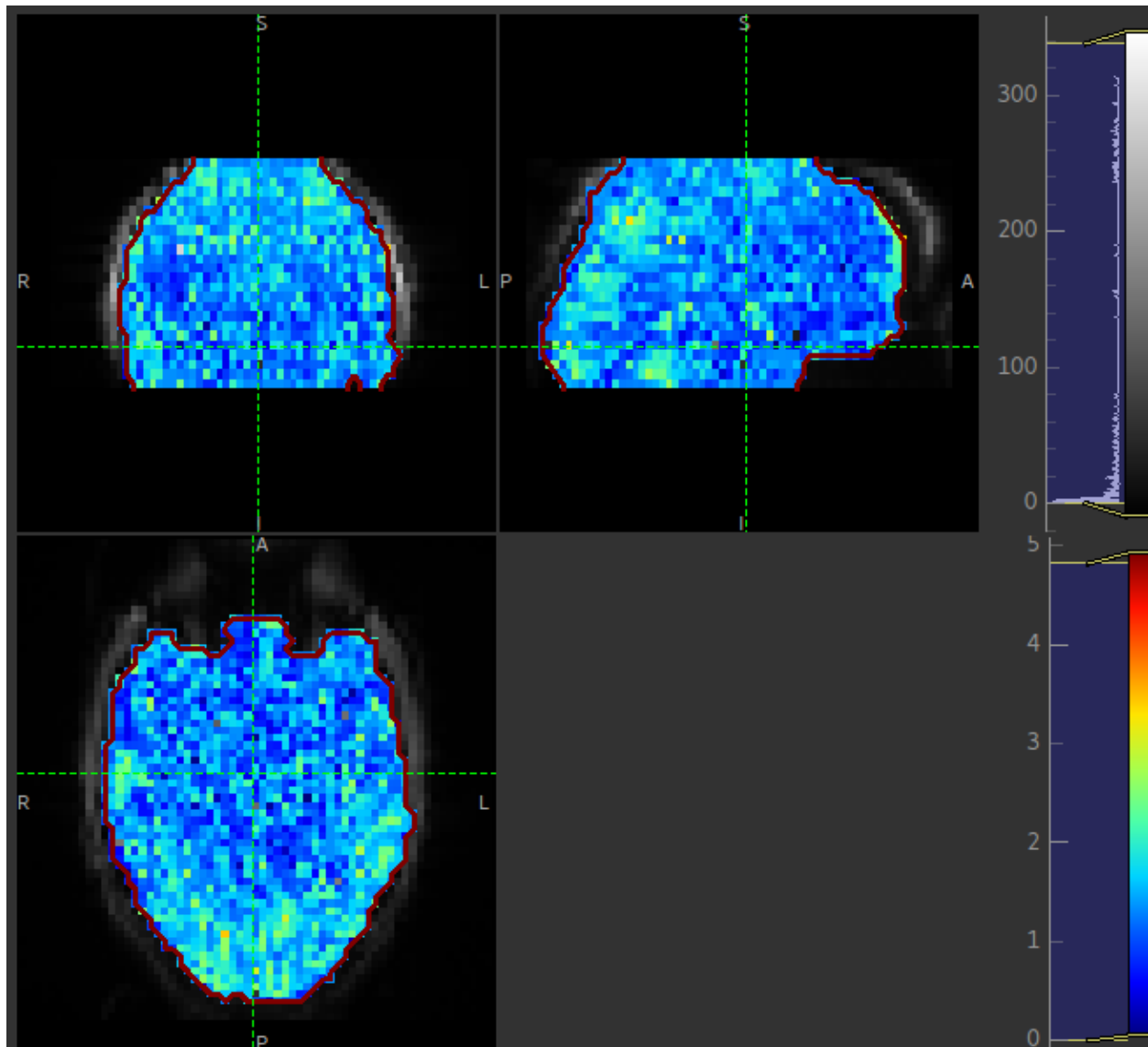
The macrovascular component should be limited to major arteries. To view a good image set the display range in your viewing software to 0-100 and view a slice through the circle of Willis:
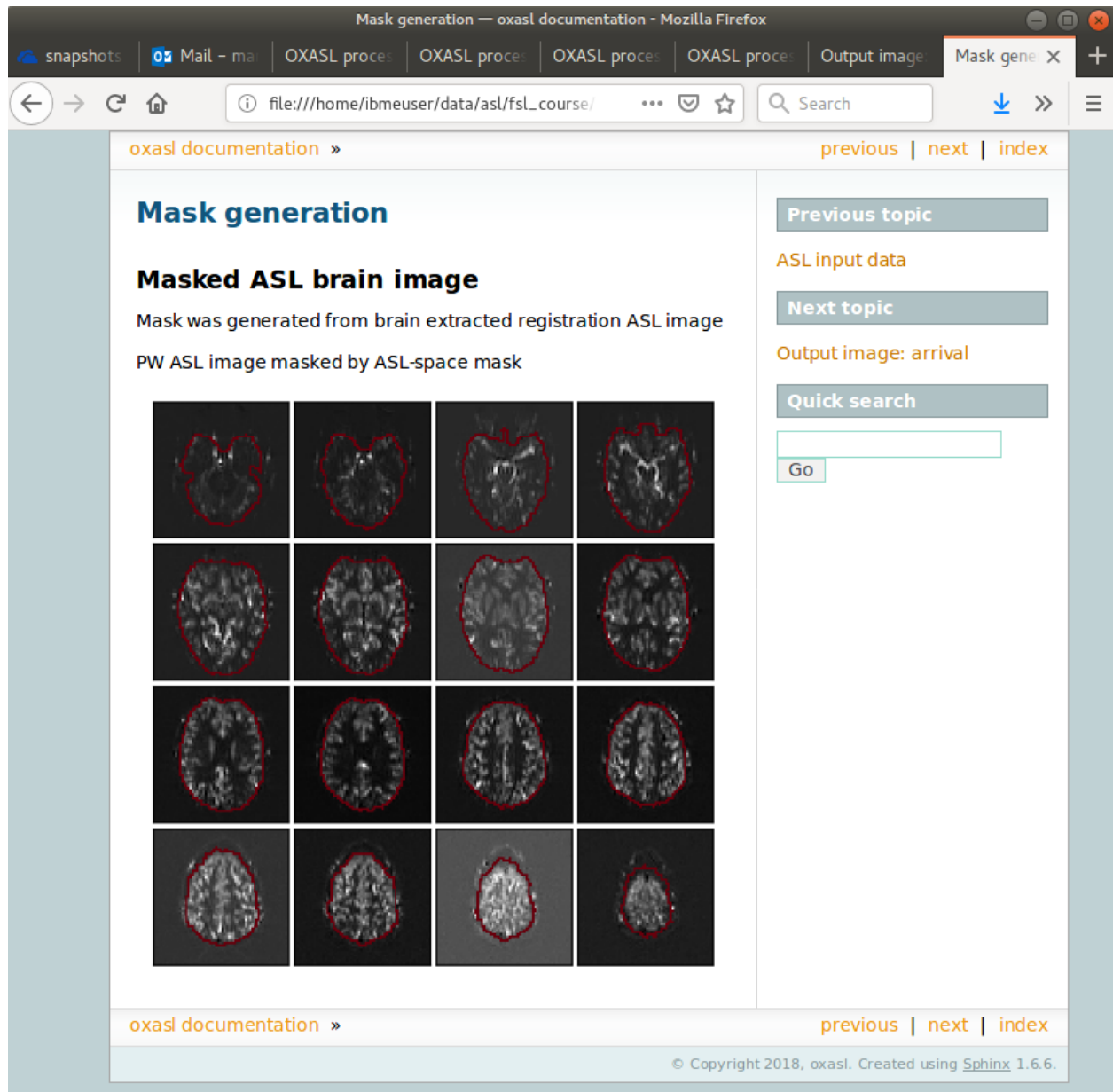
3.2.3 Summary report

If a summary report was generated, it will be stored in the `report` subdirectory. Open the file `index.html` in a Web browser to see the report:
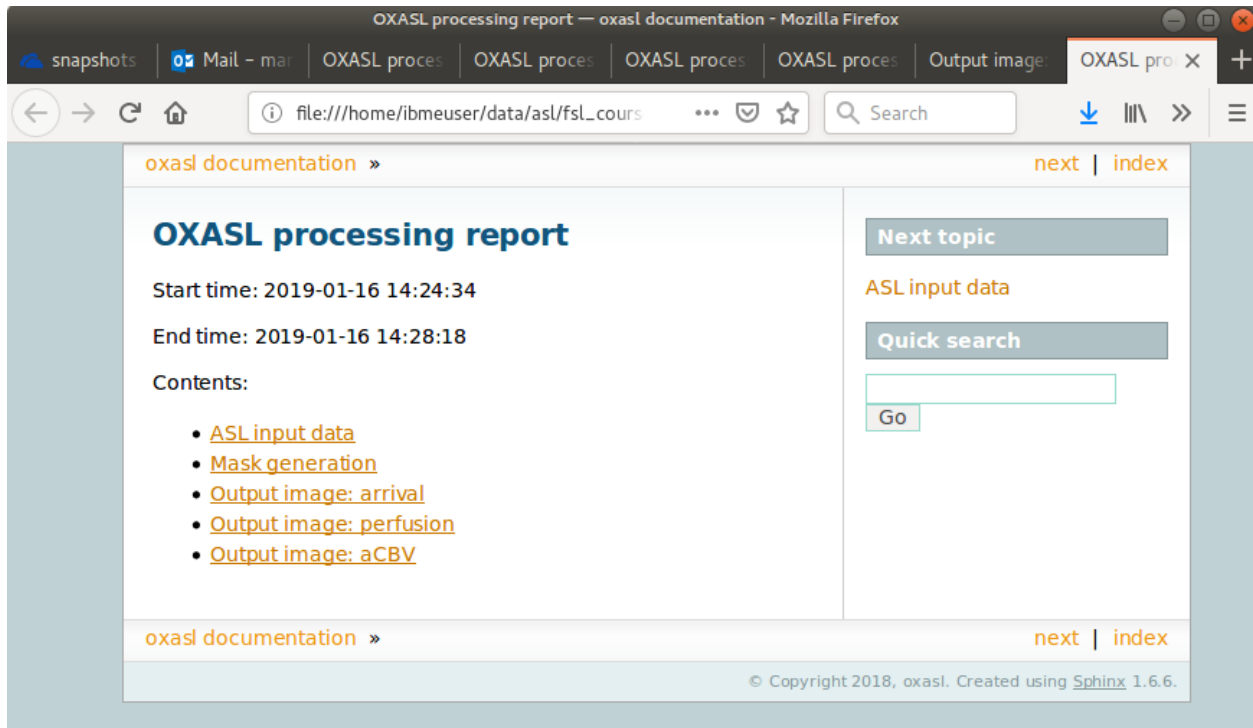
Each link provides some summary or visual representation of that part of the processing. For example we can see how good the brain mask generated was:

In this example the other pages in the report are not that interesting, but some of the more complex examples below generate useful information in the report.

3.3 Adding structural information

By providing structural information we get the following benefits:

- Better brain extraction

- Output in both ASL (native) space and also in structural space for overlaying onto structural image

- Possibility of automatic reference-region calibration (with calibration image, see below)

- Possibility of partial volume correction (see below)

Structural data may be supplied as a T1 weighted image or, better, an output folder from the FSL_ANAT command run on a structural image. This is preferred because the structural image is already segmented and bias-corrected so these steps do not need to be done by OXASL. If a structural image is supplied directly the FSL FAST segmentation tool will be used to do a structural segmentation which can be a slow process. Here we use FSL_ANAT output:

```
oxasl -i mpld_asltc --casl --iaf=tc --ibf=tis --slicedt=0.0452 \
      --plds=0.25,0.5,0.75,1.0,1.25,1.5 --bolus=1.4 \
      --fslanat T1.anat --senscorr \
      -o oxasl_out --overwrite
```

`--senscorr` indicates that the bias-correction field from the FSL_ANAT should be used. We have also added the `--overwrite` option - otherwise OXASL will refuse to run since the output directory already exists from our previous run.

3.3.1 Log output

The log output contains a few additional pieces of information. We will just highlight the differences rather than showing the entire log:

Firstly, we are using the structural image as the basis of our brain mask, and registering the ASL and structural images to each other:

```
Getting the ASL image to use for registration)
  - Registration reference is mean ASL signal (brain extracted)

Registering ASL data to structural data
  - Registering image: regfrom using FLIRT
  - ASL->Structural transform
[[ 9.99993443e-01 -3.06986241e-03 -1.90982874e-03 -1.71159280e+01]
 [ 3.05030371e-03  9.99943733e-01 -1.01611035e-02 -6.20556631e+00]
 [ 1.94091448e-03  1.01552118e-02  9.99946535e-01  3.53589818e+01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  1.00000000e+00]]
  - Structural->ASL transform
[[ 9.99993485e-01  3.05030364e-03  1.94091453e-03  1.70661166e+01]
 [-3.06986253e-03  9.99943711e-01  1.01552116e-02  5.79359551e+00]
 [-1.90982884e-03 -1.01611039e-02  9.99946567e-01 -3.54528364e+01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  1.00000000e+00]]

Generated ASL data mask
  - Mask generated from brain extracting structural image and registering to ASL space
```

We are also performing a sensitiviy correction using the bias field from the FSL_ANAT output:

```
Calculating Sensitivity correction
  - Sensitivity image calculated from bias field
```

Finally, after the modelling steps are complete, the ASL->Structural registration is improved using BBR (Boundary Based Registration) which uses the output perfusion map because this has good WM/GM contrast. This means output in structural space will be better aligned to the structural image:

```
Registering ASL data to structural data
  - BBR registration using epi_reg
  - ASL->Structural transform
[[ 9.99985245e-01 -3.27267408e-03  4.33140408e-03 -1.71764269e+01]
 [ 3.23107063e-03  9.99948863e-01  9.57749342e-03 -6.65767001e+00]
 [-4.36252543e-03 -9.56335410e-03  9.99944806e-01  4.26785518e+01]
```

```
[ 0.00000000e+00   0.00000000e+00   0.00000000e+00   1.00000000e+00]]
   - Structural->ASL transform
[[ 9.99985284e-01   3.23107076e-03  -4.36252668e-03   1.73838719e+01]
[-3.27267442e-03   9.99948967e-01  -9.56335721e-03   7.00926763e+00]
[ 4.33140255e-03   9.57749029e-03   9.99944701e-01  -4.25380300e+01]
[ 0.00000000e+00   0.00000000e+00   0.00000000e+00   1.00000000e+00]]
```

3.3.2 Output images

The ASL space (native) output should be much the same as the previous example (possibly with a slightly different brain mask). However we now also have output in structural space in the `output/struct` subdirectory. These images are transformed into the same space as the structural image so they can easily be overlaid onto the structural image. e.g. this perfusion image:
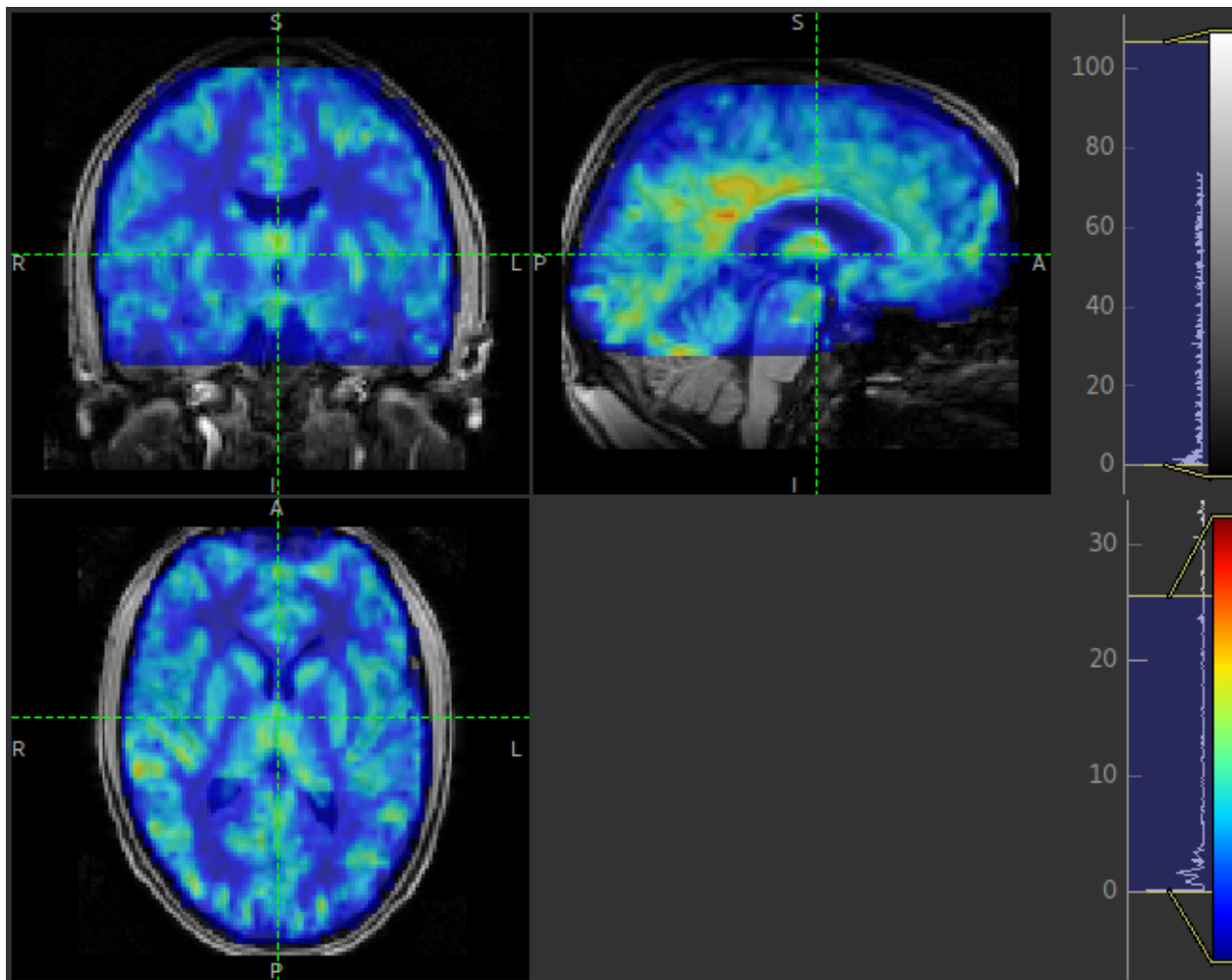
3.3.3 Summary report

The initial and final ASL->Structural registrations are presented in the report as a matrix, summary transformation parameters and an overlay of GM/WM segmentations onto the original ASL data. These should align pretty well, particularly the final registration.
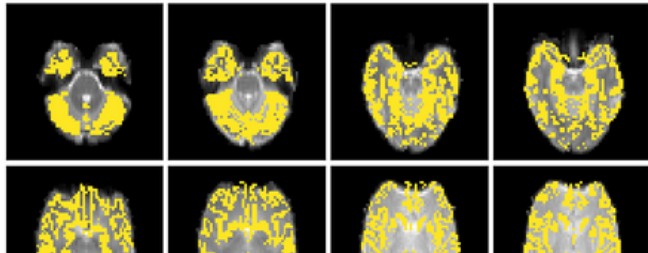
The report also includes a page showing the segmentation of the structural image into WM, GM and CSF. This may be important to check if you are using partial volume correction.

3.4 Adding calibration

Calibration enables the output of perfusion maps in physical units, enabling cross-subject and cross-session comparisons:

```
oxasl -i mpld_asltc --casl --iaf=tc --ibf=tis --slicedt=0.0452 \
      --plds=0.25,0.5,0.75,1.0,1.25,1.5 --bolus=1.4 \
      --fslanat T1.anat --senscorr \
      -c aslcalib --tr=4.8 --cmethod=single  \
      -o oxasl_out --overwrite
```

The calibration image is specified using `-c aslcalib`. `--cmethod=single` indicates that we want to use a single M0 value for calibration, derived from a reference region. By default OXASL uses CSF from the ventricles, identified by registering the structural image to a standard brain image and using this to mask the ventricles from the CSF segmentation output from either FAST or (in this case) FSL_ANAT. `--tr=4.8` allows a correction to be made for differing T1 value in the tissue and reference. TE can also be similarly provided to correct for differing T2 values but we are not doing this for this example.

3.4.1 Log output

The first part of calibration consists in calculating the tissue `M0` magnetisation value. This occurs before the modelling step as it depends only on the calibration image:

```
Calibration - calculating M0
 - Doing reference region calibration
 - Acquisition: TE=0.000000, TR=4.800000, Readout time (TAQ)=0.000000
 - Using tissue reference type: csf
 - T1r: 4.300000; T2r: 750.000000; T2b: 150.000000; Part co-eff: 1.150000
 - Doing automatic ventricle selection using standard atlas
 - Masking FAST output with standard space derived ventricle mask
 - Transforming tissue reference mask into ASL space
 - Thresholding reference mask
 - Number of voxels in tissue reference mask: 224
 - MODE: longtr
 - Calibration gain: 1.000000
 - mean signal in reference tissue: 1116.398541
 - T1 correction factor: 1.486980
 - T2 correction factor: 1.000000
 - M0: 1443.532699
```

The T1 correction factor is based on our supplied `--tr` value. The T2 correction factor is 1 because we did not supply a `--te` value.

After modelling has been done the output perfusion maps can then be scaled using this M0 value. There is also a presumed value for the inversion efficiency which differs between PASL and pCASL, and a fixed multiplier to convert the answer into physical units - for perfusion this is `ml/100g/min`

Calibrating perfusion data: perfusion

- Using inversion efficiency correction: 0.850000

- Using multiplier for physical units: 6000.000000

- Applying sensitivity correction

Calibrating perfusion data: aCBV

- Using inversion efficiency correction: 0.850000

- Using multiplier for physical units: 100.000000

3.4.2 Output images

Calibrated images are stored with the suffix `__calib`, e.g. `aCBV_calib` and `perfusion_calib`.

Since reference region calibration scales the output perfusion map by a constant M0 value, the `perfusion_calib` image looks identical to the `perfusion` image but the value range is different and should be comparable between different calibrated scans.
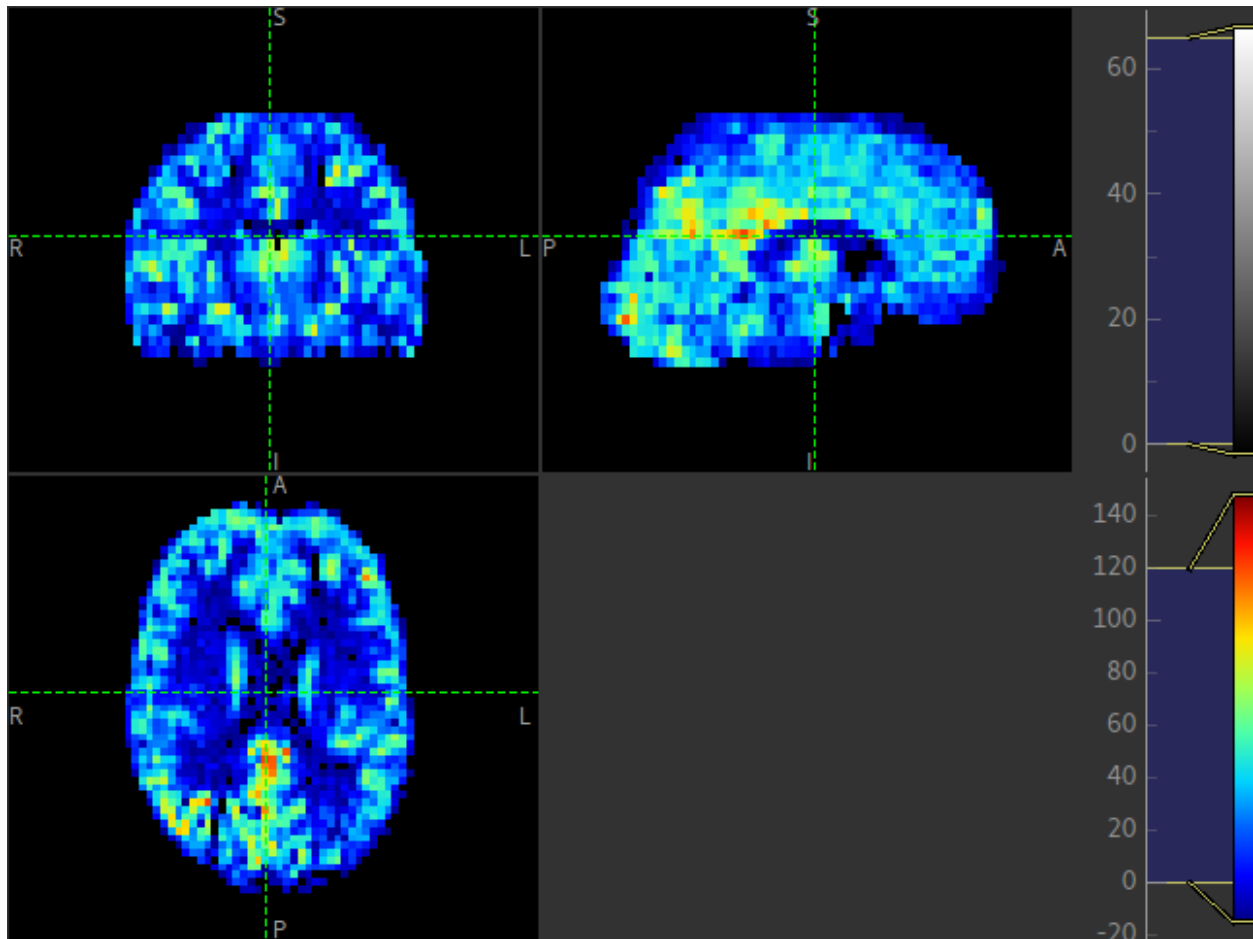
3.4.3 Summary report

Reference region calibration involves isolation of a particular tissue type in the calibration image - usually CSF from the ventricles. The report presents the steps taken to identify this region which should be checked to ensure that what it thinks are the ventricles really are. For example in this case this is the final reference mask:
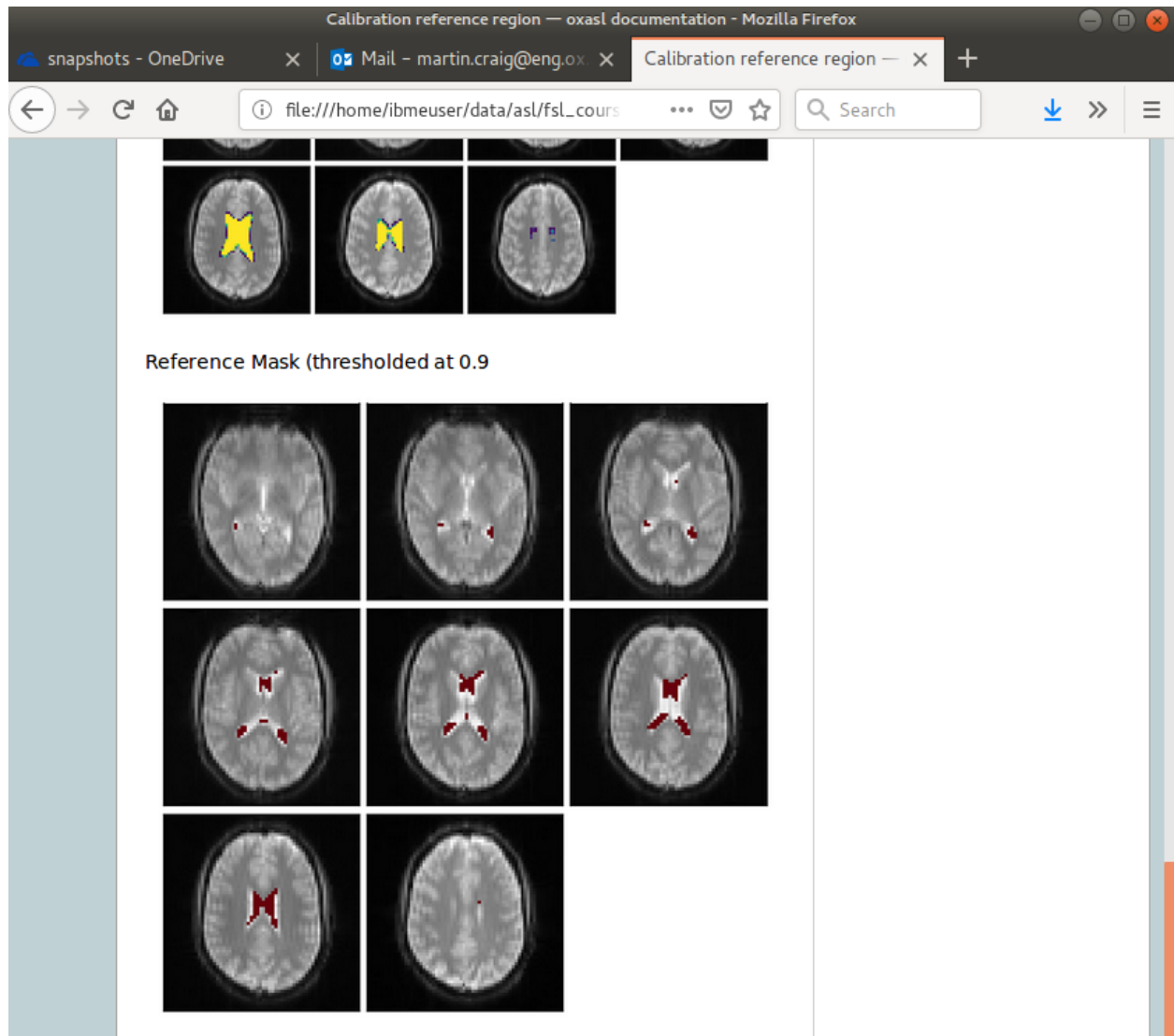
Note that this process is intended to identify voxels which are close to 100% CSF. It is not intended to identify the whole of the ventricles and the number of voxels selected may be quite small.

The report also presents average perfusion values in GM and WM with the normal ranges, so we can check things are roughly as we'd expect:

3.5 Distortion correction

In this case a phase-encoding reversed calibration image (*Blipped* image) was obtained which can be used to apply distortion correction using the FSL TOPUP tool:

```
oxasl -i mpld_asltc --casl --iaf=tc --ibf=tis --slicedt=0.0452 \
      --plds=0.25,0.5,0.75,1.0,1.25,1.5 --bolus=1.4 \
      --fslanat T1.anat --senscorr \
      -c aslcalib --tr=4.8 --cmethod=single \
      --cblip=aslcalib_PA --echospacing=0.00952 --pedir=y \
      -o oxasl_out --overwrite
```

The echo spacing (also known as the dwell time) is given in seconds and the phase encoding direction must also be given `--pedir=y`. Normally this corresponds to scanner co-ordinates, however it is important to view the results of distortion correction to make sure it is as expected.

3.5.1 Log output

Distortion correction is performed as part of the preprocessing steps. Note that this is a multi-step process and distortion correction happens at the end:

```
Calculating distortion Correction using TOPUP

Calculating Sensitivity correction
 - Sensitivity image calculated from bias field

Applying preprocessing corrections
 - Pre-processing image: calib
 - Pre-processing image: cblip
 - Data transformations
 - No corrections to apply
 - Adding TOPUP distortion correction
 - Applying sensitivity correction
```

3.5.2 Output images

The effect of distortion correction can be subtle. The image below show a slice from the perfusion map with distortion correction enabled (right image) and disabled (left image). The largest difference is at the anterior end which corresponds to `--pedir=y`.
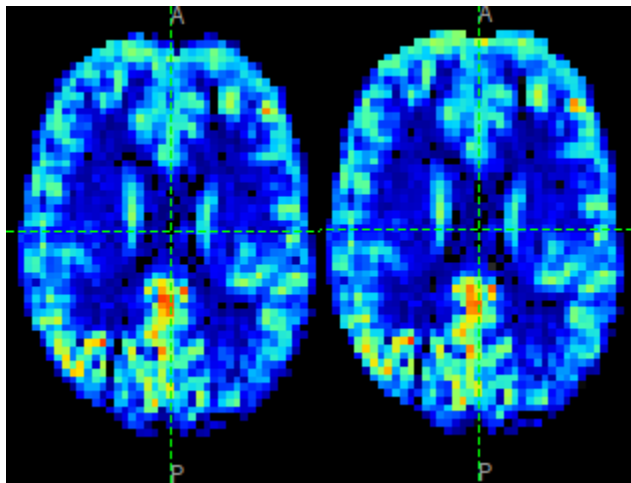
3.5.3 Summary report

The summary report includes a page presenting distortion correction images however these are not currently easy to interpret so we will not present them here.

3.6 Partial volume correction

> **Warning:** Partial volume correction adds considerably to the run time of OXASL!

Partial volume correction is enabled using the `--pvcorr` option. It uses the GM/WM segmentation from the structural data to model the GM and WM contributions separately, weighted according to the tissue proportions in each voxel:

```
oxasl -i mpld_asltc --casl --iaf=tc --ibf=tis --slicedt=0.0452 \
      --plds=0.25,0.5,0.75,1.0,1.25,1.5 --bolus=1.4 \
      --fslanat T1.anat --senscorr \
      -c aslcalib --tr=4.8 --cmethod=single  \
      --cblip=aslcalib_PA --echospacing=0.00952 --pedir=y \
      --pvcorr \
      -o oxasl_out --overwrite
```

3.6.1 Log output

Partial volume correction is not currently very well described in the log. It occurs after the main model fit has been performed, and the final stage registration (using the perfusion weighted image) has occurred. This is important - accurate PVC requires a good registration to the structural image which provides the GM and WM partial volumes.

If the data mask was generated from the structural image, it is first recreated to account for the final ASL->Structural registration. The model fitting is then run again with PVC enabled in the final step, and with an initialization step for the PVC parameters which uses the structural segmentation:

```
Generated ASL data mask
  - Mask generated from brain extracting structural image and registering to ASL space

Running BASIL Bayesian modelling on ASL data

- Doing fit on full ASL data

BASIL v0.0.7.dev19
Data shape                   : (64, 64, 24, 96)
Label type                   : Label-control pairs
Labelling                    : CASL/pCASL
PLDs (s)                     : [0.25, 0.5, 0.75, 1.0, 1.25, 1.5]
Repeats at each TI           : [8, 8, 8, 8, 8, 8]
Bolus durations (s)          : [1.4, 1.4, 1.4, 1.4, 1.4, 1.4]
Time per slice (s)           : 0.0452
Model (in fabber) is : aslrest
Dispersion model option is none
Compartment exchange model option is mix
Step 1 of 4: VB -  Tissue  100%
Step 2 of 4: VB -  Tissue  Arterial  - Initialise with step 1 100%
Step 3 of 4: PVC initialisation - Initialise with step 2    Initialising partial␣
↪volume correction...
DONE
Step 4 of 4: Spatial VB -  Tissue  Arterial  PVE - Initialise with step 3 100%
```

Note the PVC initialisation in Step 3, and the PVE component in Step 4.

3.6.2 Output images

The main differenec is that the perfusion image is split between GM (`perfusion_calib`) and WM (`perfusion_wm_calib`). Both should only be interpreted within the corresponding segmentation - outside these regions (e.g. in WM regions when viewing the GM perfusion map), the image will be smooth and lacking in detail -

reflecting the lack of information in the data for this region. This is visible in the images below (Top: GM, Bottom: WM).
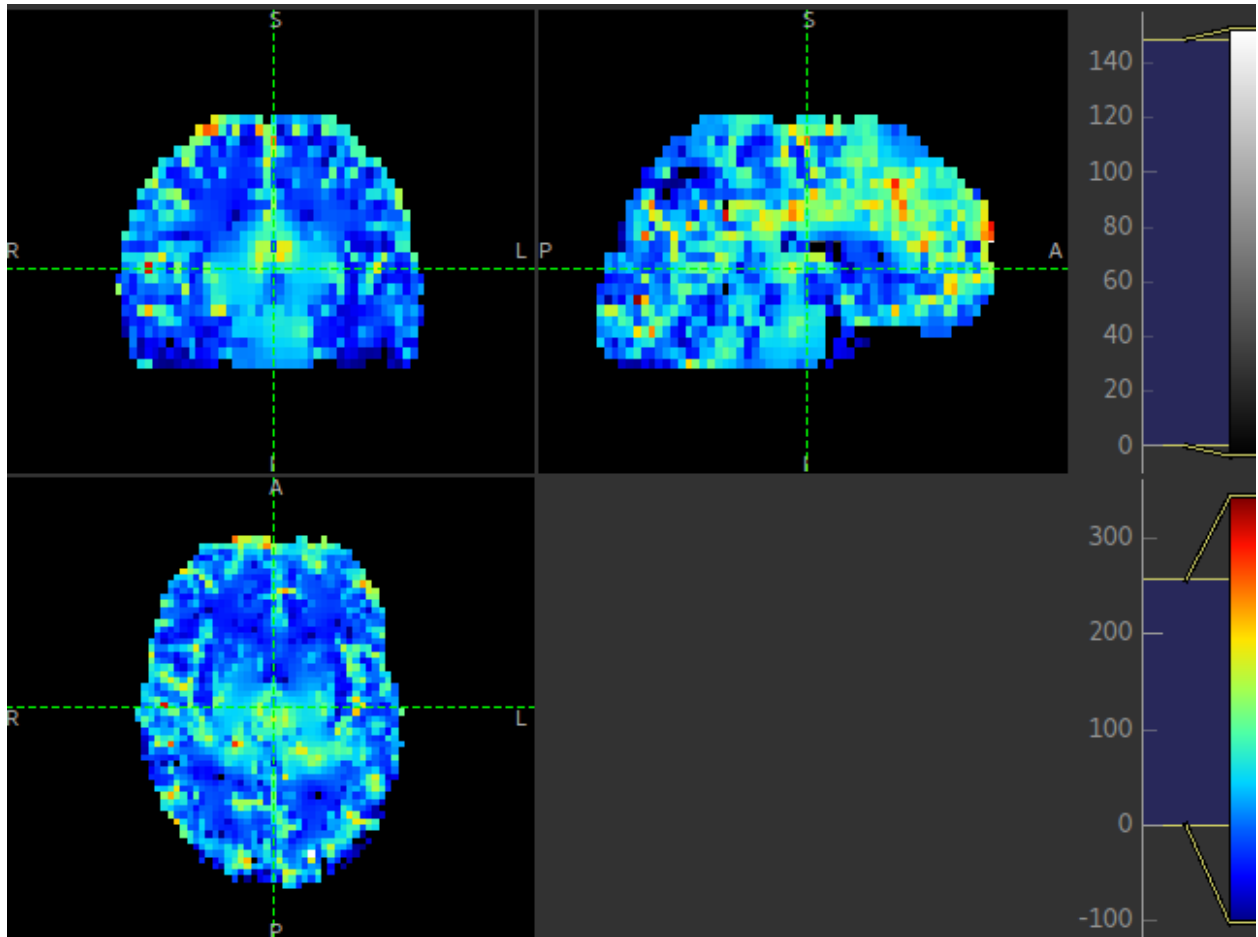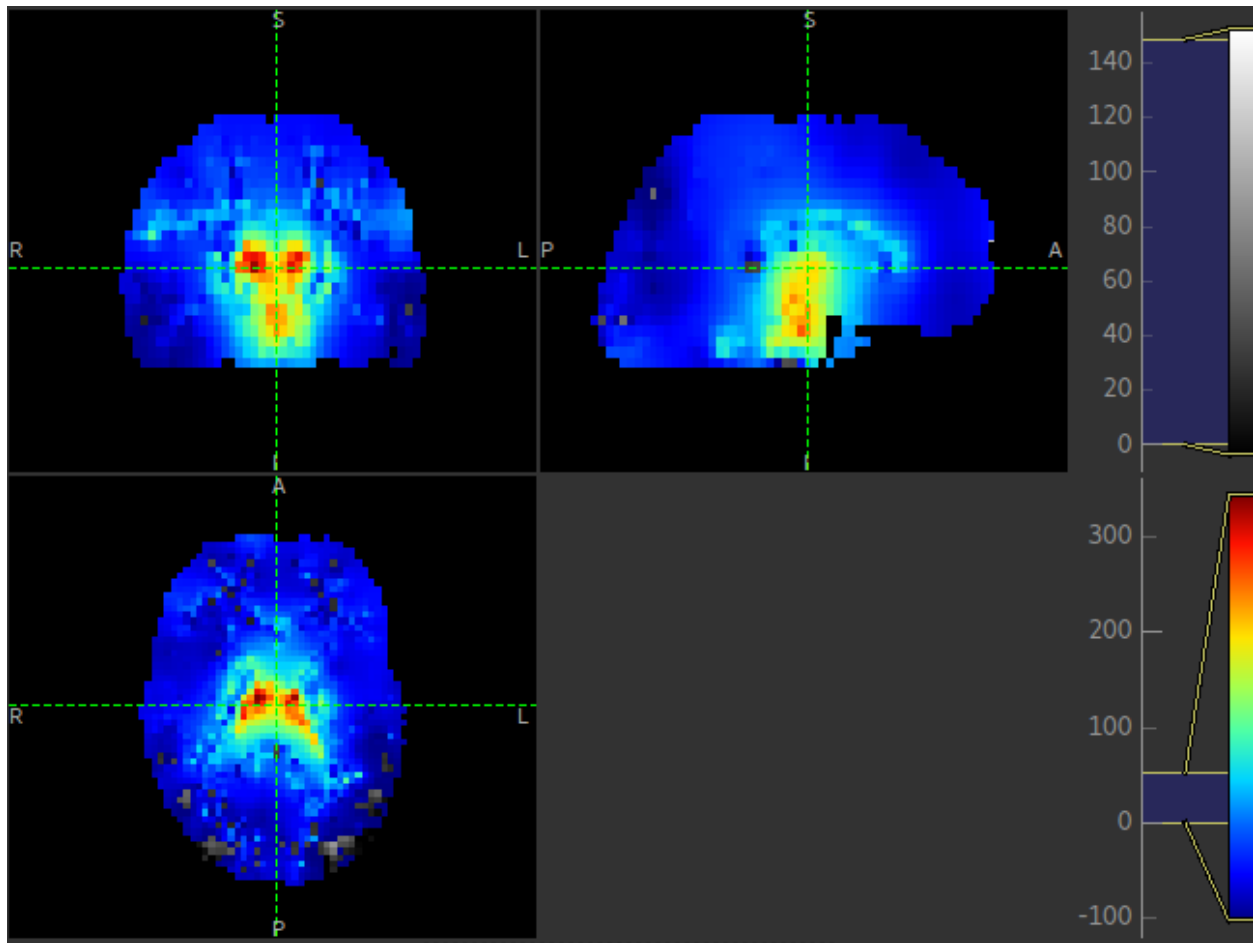
3.6.3 Summary report

In the summary report, it is important to disregard the WM averages in the GM perfusion map, and vice versa:

Output image: perfusion_calib — oxasl documentation - Mozilla Firefox

| snapshots - OneDrive ✕ | Mail – martin.craig@eng ✕ | TOPUP distortion correctio ✕ | Output image: perfusion_ca ✕ | + |

file:///home/ibmeuser/data/asl/fsl_cours   Search

oxasl documentation »   previous | next | index

Output image: perfusion_calib

Calibration

Image was calibrated using supplied M0 image
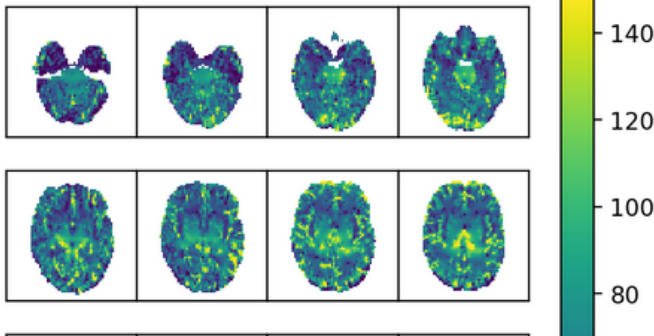
Inversion efficiency: 0.850000

Multiplier for physical units: 6000.000000

Metrics

Metric	Value	Typical
Mean within mask	67.36 ml/100g/min	
GM mean	58.82 ml/100g/min	30-50
Pure GM mean	46.46 ml/100g/min	30-50
WM mean	59.13 ml/100g/min	10-20
Pure WM mean	60.29 ml/100g/min	10-20

Image

Previous topic

Output image: arrival

Next topic

Output image: aCBV_calib

Quick search

Go

OXASL walk through tutorial - GUI

This tutorial demonstrates some of the common options available in the OXASL GUI.

We will be working with single and multi-PLD data from the FSL tutorial on Arterial Spin Labelling. You will need to download this data before following the tutorial.

The tutorial has been written so that we start with the most basic analysis and gradually add options and show the effect they have on the output, as well as how they are reported in the command log and the summary report. However this is not a complete description of all available options in the GUI or the command line - for that see the *OXASL walk through tutorial - command line* or *OXASL command reference*.

Contents

4.1 Perfusion quantification using Single PLD pcASL

The aim of this exercise is to perform perfusion quantification with one of the most widely recommened variants of ASL. Single PLD pcASL is now regarded as sufficiently simple and reliable, both for acquisition and analysis, that it is the first option most people should consider when using ASL for the first time. Although more can be done with other ASL variants, particularly when acquisition time allows.

4.1.1 The data

This dataset used **pcASL labeling** and we are going to start with data collected using a **single post-label delay**. This dataset follows as closely as possible the recommendations of the ASL Consensus Paper (commonly called the 'White Paper') on a good general purpose ASL acquisition, although we have chosen to use a 2D mutli-slice readout rather than a full-volume 3D readout.

The files you will need to begin with are:

- `spld_asltc.nii.gz` - the label-control ASL series containing 60 volumes. That is 30 label and 30 control, in pairs of alternating images with label first.

- `aslcalib.nii.gz` - the calibration image, a (largely) proton-density weighted image with the same readout (resolution etc) as the main ASL data. The TR for this data is 4.8 seconds, which means there will be some T1 weighting.

- `aslcalib_PA.nii.gz` - another calibration image, identical to `aslcalib.nii.gz` apart from the use of posterior-anterior phase encoding (anterior-posterior was used in the rest of the ASL data). This is provided for distortion correction.

- `T1.nii.gz` - the T1-weigthed anatomical of the same subject.

To launch the GUI at the command line you will need to type `oxasl_gui`. Note that if you have downloaded the 'pre-release' yourself, you may need to provide a path to the installed version of the GUI, e.g. `/Users/{blah}/Downloads/oxasl/oxasl_gui`

Once it has launched you will find yourself on the 'Input Data' tab, you should:

- Load the ASL data `spld_asltc.nii.gz` as the 'Input Image'.

- Set the 'Number of PLDs', which in this case is 1, this is already done by default.

- Click the 'Update' button beneath the 'Data Preview' pane on the right.

At this point the GUI should look like the screen shot below and a perfusion weighted image will have appeared in the 'Data Preview' pane. This this is reassuring, if we didn't see something that looks roughly like this, we might check if the data order that the GUI is expecting matches that in the data. We could alter the 'Data order' settings if needed and update the preview again.
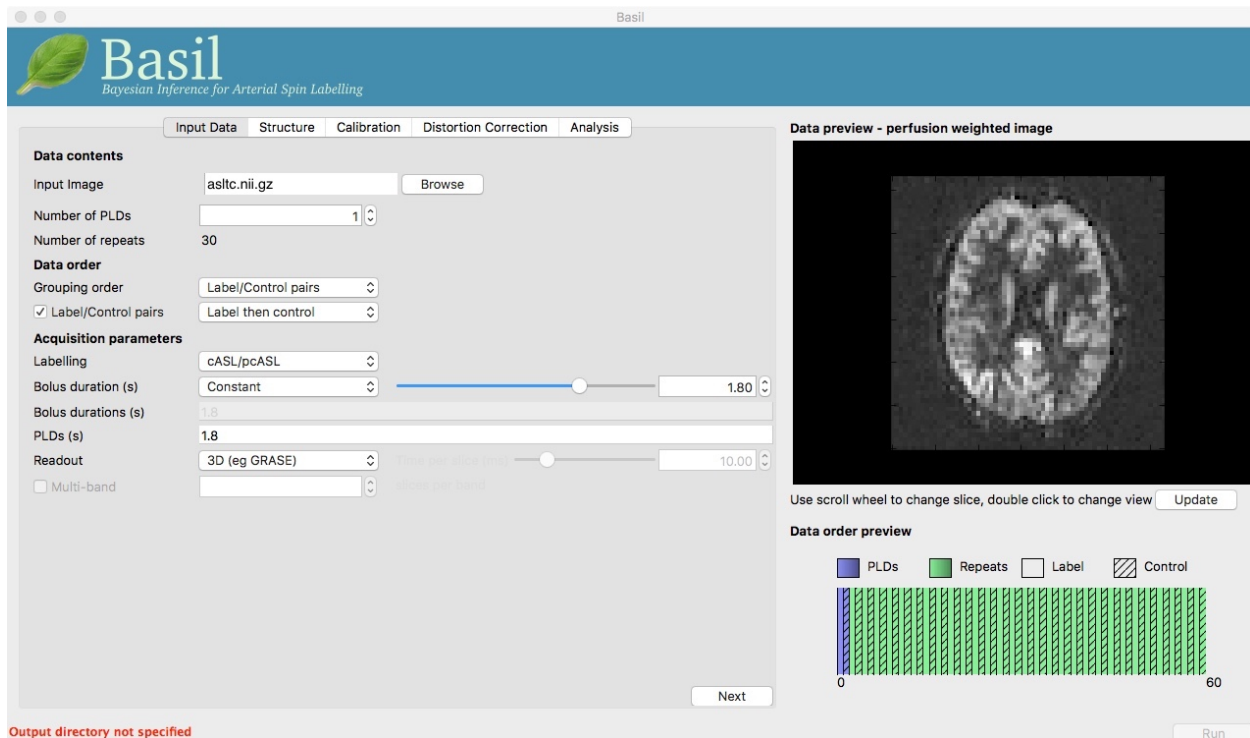
Note also, beneath the 'Data Preview', that there is a 'Data order preview'. The idea of this graphic is to help visually to confirm that the way that the GUI is intepreting the ordering of volumes in the data matches what you are expecting. In this case we have a single PLD repeated 30 times with the label and control images paired in the data (this is pretty common). What the 'Data order preview' shows is the first instance of the PLD in purple, showing both the label and control (hatched) volume. Each subsequent repeat of the same PLD is coloured green, again showing that we have a label follwed by control (hatched) volume.

You can try a different 'Data order' option to see what happens. Change 'Label/Control pairs' from 'Label then control' to 'Control then label'. This switches the expected order of label and control images within the pair. If you then udpate the preview you will find that the contrast reverses, the perfusion now has the wrong 'sign'.

4.1.2 (Simple) Perfusion Quantification

We have checked the PWI, thus we can proceed to final quantification of perfusion, inverting the kinetics of the ASL label delivery and using the calibration image to get values in the units of ml/100g/min.

To do this we need to tell the BASIL GUI some information about the data and the analysis we want to perform.

On the 'Input Data' tab we need to sepcify the 'Acquisition parameters':

- Labelling - cASL/pcASL (the deafult option).

- Bolus duration (s) - 1.8 (default).

- PLDs (s) - 1.8 (default).

- Readout - 2D multi-slice (you will need to set this).

- Time per slice (ms) - 45.2 (only appears when you change the Readout option).

You can now hit 'Next' and you will be taken to the next tab. For this (simple) analysis we do not want to use a structural image, so we can move on by clicking 'Next' again. Or we could skip stright to the 'Calibration' tab using the menu across the top.

On the 'Calibration' tab, 'Enable Calibration' first, then load the calibration image `aslcalib.nii.gz`. Change the 'Calibration mode' to 'voxelwise', and set the 'Sequence TR (s)' to be 4.8.
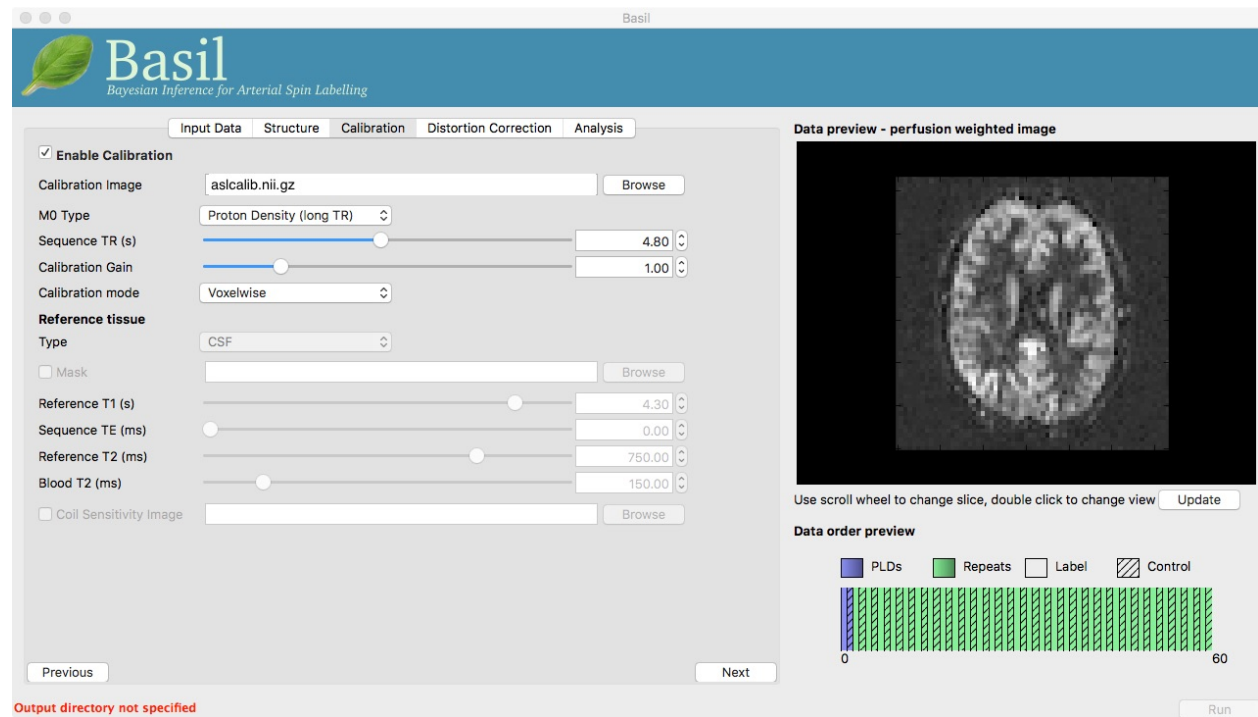
Finally, we need to set the analysis options: either skip to the 'Analysis' tab or click 'Next' twice.

On the 'Analysis' tab, choose an output directory name, e.g., `oxasl`. And, select 'Analysis which conforms to White Paper', so that we know the analysis is using the same default parameter values proposed in the 'ASL White Paper' quantification formula. Note that in the lower left corner the GUI is now telling us that we are 'Ready to Go'. At this point you can click 'Run' in the lower right corner.
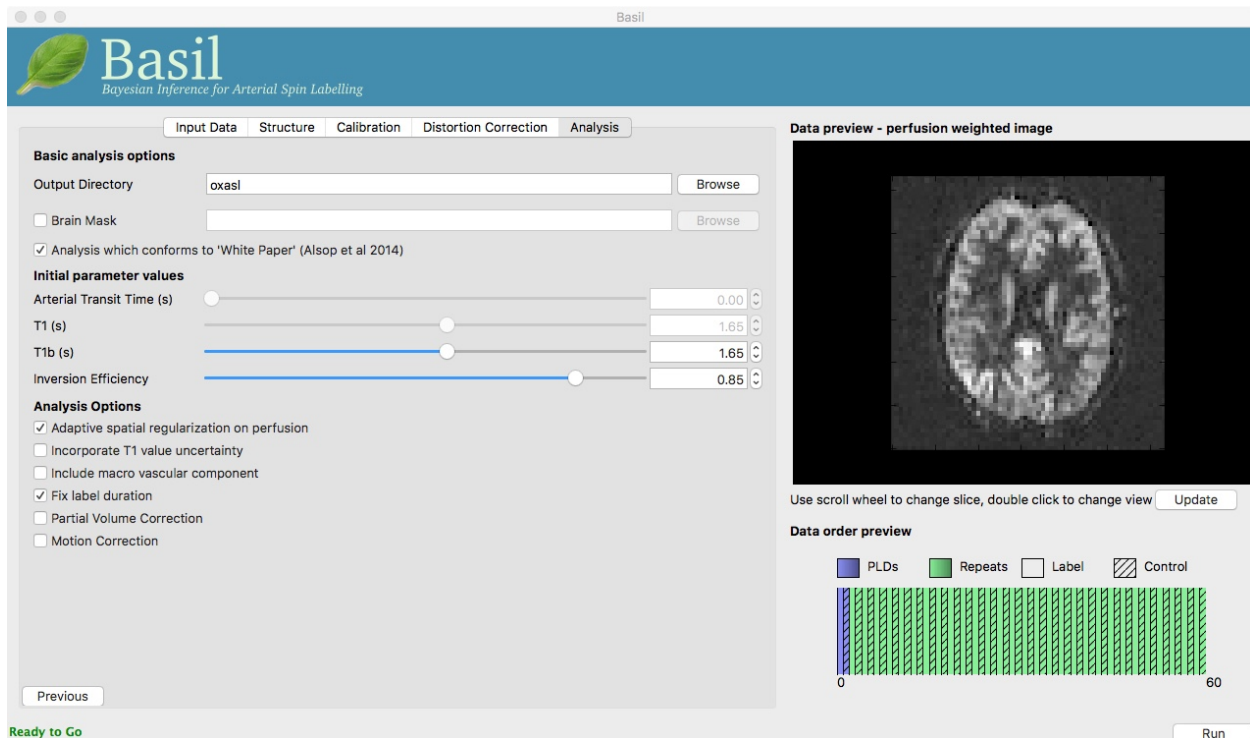
The output of the oxasl command line tool is shown in a pop-up window. You can ignore any `erfc underflow error` messages - they are harmless and occur because we haven't provided any structural data

This analysis should only take a few minutes, but while you are waiting you can read ahead and even start changing the options in the GUI ready for the next analysis that we want to run.

Once the analysis had completed, view the final result:

```
fsleyes oxasl/output/native/calib_voxelwise/perfusion.nii.gz
```

Note that if you just supply a name for the output directory (not a full path), as we have here, this will be placed in the 'working directory', i.e. whichever directory you were in when you launched the GUI.

You will find something that looks very similar to the PWI we viewed before, but now the values at every voxel are in ml/100g/min.

You will also find a PWI saved as `oxasl/output/native/perfusion`. This is very similar to the PWI displayed in the preview pane, except that the kinetic model inversion has been applied to it, this is the image pre-calibration.

4.2 Improving the Perfusion Images from single PLD pcASL

The purpose of this practical is essentially to do a better job of the analysis we did above, exploring more of the features of the GUI including things like motion and distortion correction.

4.2.1 Motion and Distortion correction

Go back to the GUI which should still be setup from the last analysis you did (if you have closed it follow the steps above to repeat the setup - but do not click run).

On the 'Distortion Correction' tab, select 'Apply distortion correction'. Load the 'Phase-encode-reveresed calibration image' `aslcalib_PA.nii.gz`. Set the 'Effective EPI echo spacing' (also known as the dwell time) to 0.95ms and the 'Phase encoding direction' to 'y'.
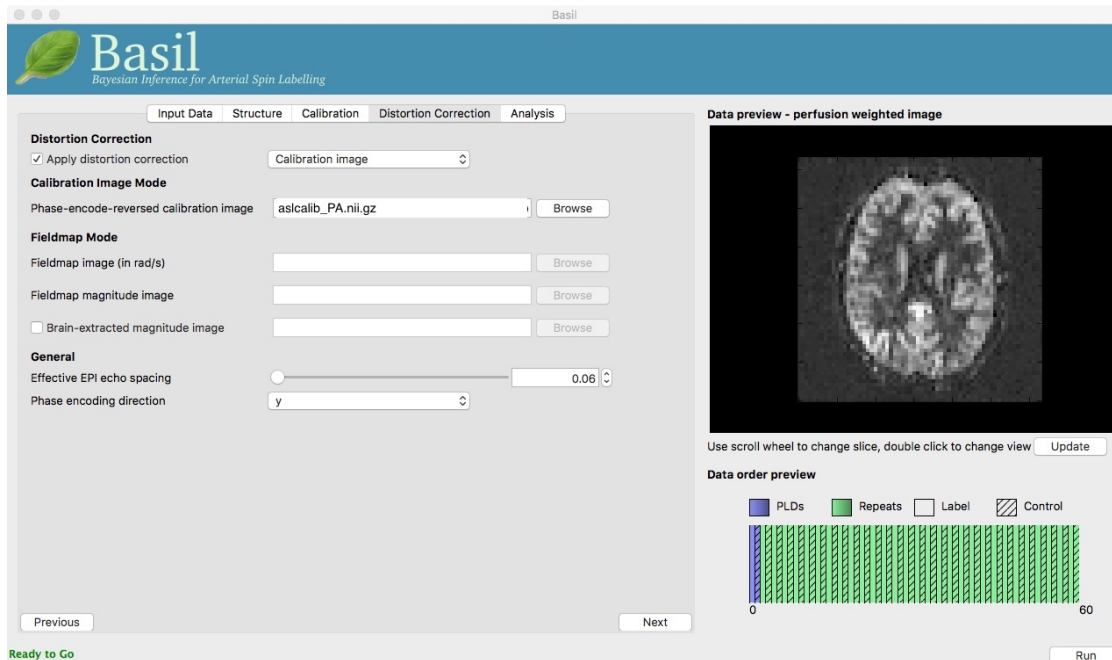
On the 'Analysis' tab, select 'Motion Correction'. Make sure you have 'Adaptive spatial regularisation on perfusion' selected (it is by default). This will reduce the appearance of noise in the final perfusion image using the minimum amount of smoothing appropriate for the data.

You might like the change the name of the output directory at this point, so that you can comapre to the previous analysis.

Now click 'Run'.
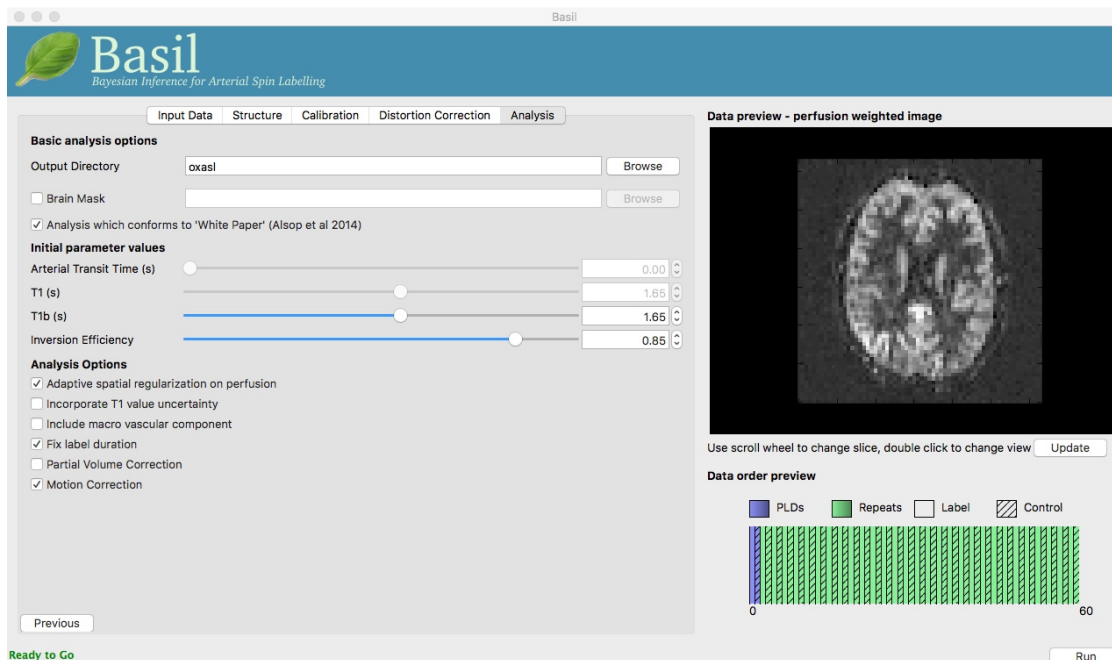
For this analysis we are still in 'White Paper' mode. Specifically this means we are using the simplest kinetic model, which assumes that all delivered blood-water has the same T1 as that of the blood and that the Arterial Transit Time should be treated as 0 seconds.

As before, the analysis should only take a few minutes, slightly longer this time due to the distortion and motion correction. Like the last exercise you might want to skip ahead and start setting up the next analysis.

To view the final result:

```
fsleyes oxasl/output/native/calib_voxelwise/perfusion.nii.gz
```

The result will be similar to the analysis in Example 1 although the effect of distortion correction should be noticeable in the anterior portion of the brain. The effects of motion correction are less obvious, this data does not have a lot of motion corruption in it.

4.2.2 Making use of Structural Images

Thus far, all of the analyses have relied purely on the ASL data alone. However, often you will have a (higher resolution) structural image in the same subject and would like to use this as well, at the very least as part of the process to transform the perfusion images into some template space.

We can repeat the analysis above but now providing structural information. The recommended way to do this is to take your T1 weighted structural image (which is most common) and firstly process using `fsl_anat`, passing the output directly from that tool BASIL.

For this practical `fsl_anat` has already been run for you and you will find the output in the data directory as `~/fsl_course_data/ASL/T1.anat`

Go back to the analysis you have setup above. On the 'Structure' tab, for 'Structural data from' select 'Existing FSL_ANAT output'. Then for the 'Existing FSL_ANAT output' choose `T1.anat`.
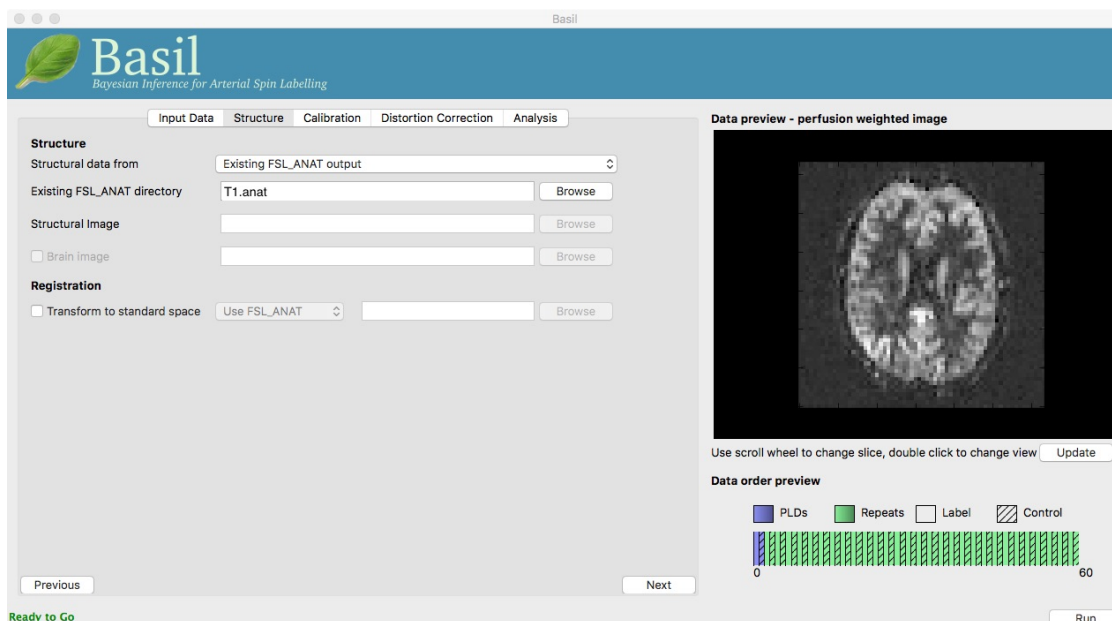
This analysis will take somewhat longer overall (potentailly 15-20 mins), the extra time is taken up doing careful registration between ASL and structural images. Thus, this is a good point to keep reading on and leave the analysis runnning.

You will find some new results in the output directory:

- `oxasl/struct_space` - this sub-drectory contains results transformed into the same space as the structural image. The files in here will match those in the `native` subdirectory of the earlier analysis, i.e., containing perfusion images with and without calibration.

- `oxasl/output/native/asl2struct.mat` - this is the (linear) transformation between ASL and structural space. It can be used along with a transformation between structural and template space to transform the ASL data into the template space. It was used to create the results in `oxasl/output/struct`.

- `oxasl/output/native/perfusion_calib_gm_mean.txt` - this contains the result of calculating the perfusion within a gray matter mask, these are in ml/100g/min. The mask was derived from the partial volume estimates created by `fsl_anat` and transformed into ASL space followed by thresholding at 70%. This is a helpful check on the absolute perfusion values found and it is not aytpical too see values in the range 30-50 here. There is also a white matter result (for which a threshold of 90% was used).

- `oxasl/output/native/gm_mask.nii.gz` - this is the gray matter mask used in the above calculations. There is also the associated white matter mask.

- `oxasl/output/native/gm_roi.nii.gz` - this is another mask that represents areas in which there is some grey matter (at least 10% from the partial volume estimates). This can be useful for visualisation, but mainly when looking at partial volume corrected data.

4.2.3 Different model and calibration choices

Thus far the calibration to get perfsion in units of ml/100g/min has been done using a voxelwise division of the realtive perfusion image by the (suitably corrected) calibration image - so called 'voxelwise' calibration. This is in keeping with the recommendations of the ASL White Paper for a simple to implement quantitative analysis. However, we could also choose to use a reference tissue to derive a single value for the equilibrium magnetization of arterial blood and use that in the calibration process.

Go back to the analysis you have already set up. We are now going to turn off 'White Paper' mode, this will provide us with more options to get a potentially more accurate analysis. To do this return to the 'Analysis' tab and deselect the 'White Paper' option, you will see that the 'Arterial Transit Time' goes from 0 seconds to 1.3 seconds (the default value for pcASL in BASIL based on our experience with pcASL labeling plane placement) and the 'T1' value (for tissue) is different to 'T1b' (for arterial blood), since the Standard (aka Buxton) model for ASL kinetics considers labeled blood both in the vascualture and the tissue.
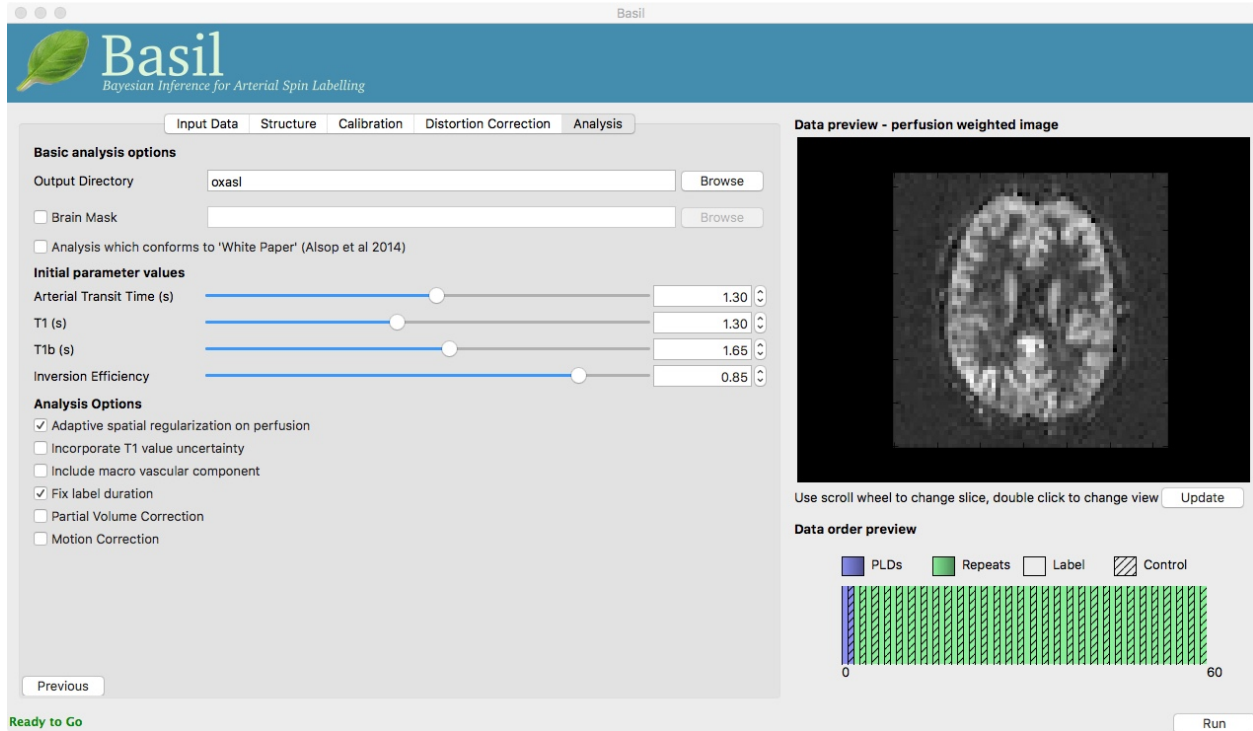
Now that we are not in 'White Paper' mode we can also change the calibration method. On the 'Calibration' tab, change the 'Calibration mode' to 'Reference Region'. Now all of the 'Reference tissue' options will become available, but leave these as they are: we will accept the default option of using the CSF (in the ventricles) for calibration.
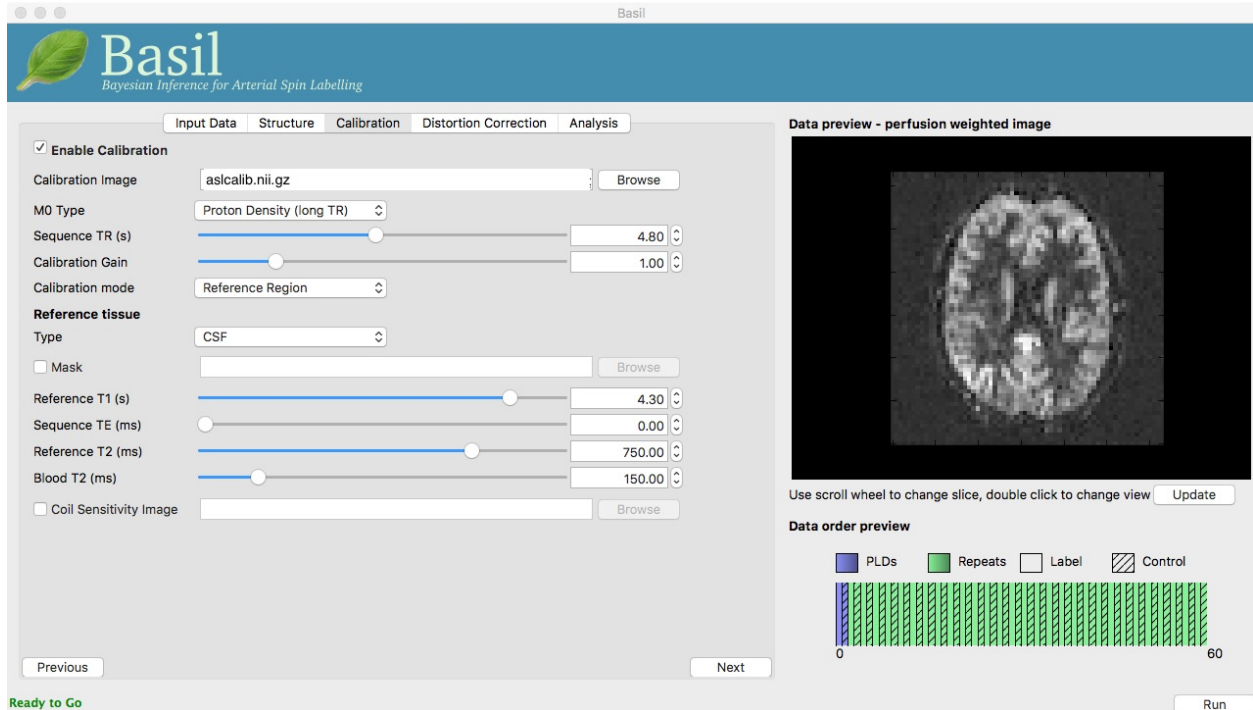
You could click 'Run' now and wait for the analysis to complete. But, in the interests of time we will save ourselves the bother of doing all of the registration all over again. Before clicking run, therefore, do:

- On the 'Calibration' tab select 'Mask' and load `csfmask.nii.gz` from the data directory. This is a ready

prepared ventricular mask for this subject. (in fact it is precisely the mask you would get if you ran the analysis as setup above).

- Go back to the 'Structure' tab and choose 'None' for 'Structural data from'. This will turn off all of the registration processes.

- You might also like to choose a different output directory name, so that you can comapre with the previous analysis.

While this is running you might want to read ahead, or if you are keen to keep moving through the examples, then skip this analysis and keep going.

The resulting perfusion images should look very similar to those produced using the voxelwise calibration, and the absolute values should be similar too. For this, and many datasets, the two methods are broadly equivalent. You can check on some of the interim calcuations for the calibration by looking in the `oxasl/calib` subdirectory: here you will find the value of the estimated equilirbirum mangetization of arterial blood for this dataset in `M0.txt` and the reference tissue mask in `refmask.nii.gz`. It is worth checking that the latter does indeed only lie in the venticles when overlaid on an ASL image (e.g. the perfusion image or the calibration image), it should be conservative, i.e., only select voxels well within the ventricles and not on the boundary with white matter.

4.2.4 Partial Volume Correction

Having dealt with structural image, and in the process obtained partial volume estimates, we are now in a position to do partial volume correction. This does more than simply attempt to estimate the mean perfusion within the grey matter, but attempts to derive and image of gray matter perfusion directly (along with a separate image for white matter).

This is very simple to do via the GUI. Return to your earlier analysis. You will need to revist the 'Structure' tab and reload the `T1.anat` result as you did above, the partial volume estimates produced by `fsl_anant` (in fact they are done using `fast`) are needed for the correction. On the 'Analysis' tab, select 'Partial Volume Correction'. That is it! You might not want to click 'Run' at this point becuase partial volume correction takes substantially longer to run.

You will find the results of this analysis already completed for you in the directory `~/fsl_course_data/ASL/oxasl_spld_pvout`. In this results directory you will still find an analysis performed without partial volume correction in `oxasl/output/native` as before. The results of partial volume correction can be found in `oxasl/output/native/pvcorr`. This new subdirectory has the same structure as the non-corrected results, only now `perfusion_calib.nii.gz` is an estimate of perfusion only in gray matter, it has been joined by a new set of images for the estimation of white matter perfusion, e.g., `perfusion_wm_calib.nii.gz`. It may be more helpful to look at `perfusion_calib_masked.nii.gz` (and the equivalent `perfusion_wm_calib_masked.nii.gz`) since this has been masked to include only voxels with more than 10% gray matter (or white matter), i.e., voxels in which it is reasonable to interpret the gray matter (white matter) perfusion values.

4.3 Perfusion Quantification (and more) using Multi-PLD pcASL

The purpose of this exercise is to look at some multi-PLD pcASL. As with the single PLD data we can obtain perfusion images, but now we can account for any differences in the arrival of labeled blood-water (the arterial transit time, ATT) in different parts of the brain. As we will also see we can extract other interesting parameters, such as the ATT in its own right, as well as arterial blood volumes.

4.3.1 The data

The data we will use in this section supplements the single PLD pcASL data above, adding multi-PLD ASL in the same subject (collected in the same session). This dataset used the same pcASL labelling, but with a label duration of 1.4 seconds and 6 post-labelling delays of 0.25, 0.5, 0.75, 1.0, 1.25 and 1.5 seconds.

The files you will also now need are:

- `mpld_asltc.nii.gz` - the label-control ASL series containing 96 volumes: each PLD was repeated 8 times, thus there are 16 volumes (label and control paired) for each PLD. The data has been re-ordered from the way it was acquired, such that all of the measurements from each PLD have been grouped together - it is important to know this data ordering when doing the analysis.

4.3.2 Perfusion Quantification

Load the GUI (`asl_gui`), it is best to start a whole new analysis as we are moving on to a new set of data and not reuse any GUI you already have open. On the 'Input Data' tab, for the 'Input Image' load `mpld_asltc.nii.gz`. Unlike the single-PLD data, we need to specify the correct number of PLD, which is 6. At this point the 'Number of repeats' should correctly read 8. Click 'Update' below the 'Data preview pane'. A perfusion-weighted image should appear - this is an average over all the PLDs (and will thus look different to Example 1).
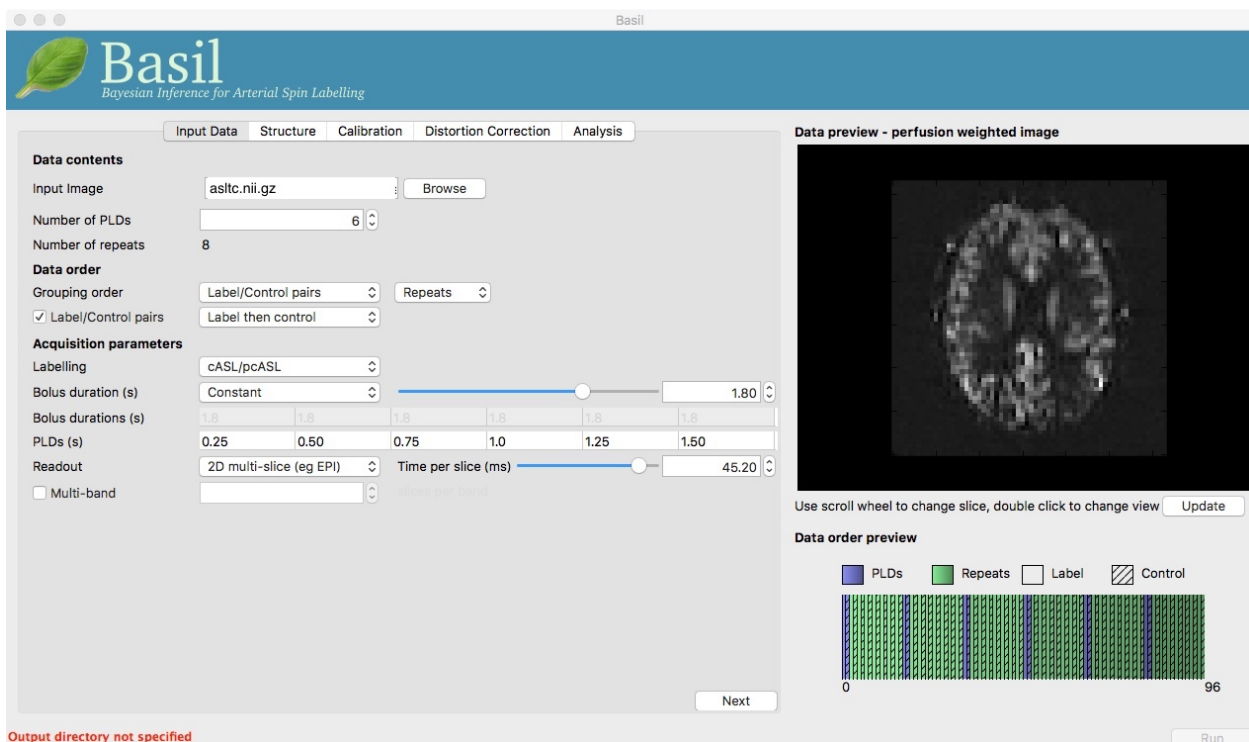
Note the 'Data order preview'. For mutli-PLD ASL it is important to get the data order specification right. In this case the default options in the GUI are not correct. The PLDs do come as label-control pairs, i.e. alternating label then control images. But, the default assumption in the GUI is that a full set of the 6 PLDs has been acquired first, then this has been repeated 8 subseqeunt times, this is indcated in the preview by colouring the first instance of a PLD as purple and subsequent as green, with different PLDs appearing as different shades of purple (or green). This is quite commonly how multi-PLD ASL data is acquired, but that might not be how the data is ordered in the final image file.

As we noted earlier, in this data all of the measurements at the same PLD are grouped together. You need to change the 'Grouping order' on the 'Input Data' tab: leave the first option along ('Label/Control pairs') and change the second option from 'PLDs' to 'Repeats'. Note that the data order preview changes to reflect the different ordering. This is now correct: remeber that the purple coloured entries indicate the first time that PLD was acquired.

Note that if you were to click 'Update' on the 'Data preview' nothing changes, the ordering doesn't affect the (simple) way in which we have calucated the PWI. Getting a plausible looking PWI is a good sign that the data order is correct, but it is not a guarantee that the PLD ordering is correct, so always check carefully. One way to do this, in this case, would be to open the data in `fsleyes` and look at the timeseries: the raw intensity of both label and control images

for one PLD are different to those from another PLD (due to the background suprresion). THe timeseries for the raw data looks like a series of steps, indicating the repeated measurements from each PLD are grouped together (groubed by 'repeats').

Once we are happy with the PWI and data order, we can set the 'Acquisition parameters':

- Labelling - 'cASL/pcASL' (default).

- Bolus duration (s) - 1.4 (shorter than the default).

- PLDs (s) - 0.25, 0.5, 0.75, 1.0, 1.254, 1.5.

- Readout - '2D multi-slice' with 'Time per slice' 45.2.

Move to the 'Calibration' tab, select 'Enable Calibration' and as the 'Calibration Image' load the `aslcalib.nii.gz` image from the Single-PLD data (it is from the same subject in the same session so we can use it here too). We have skipped the 'Structure' tab (to make the analysis quicker), this means if we want 'Calibration mode' to be 'Reference Region' we need to supply a mask of the region of tissue to use. Select 'Mask' and load `csfmask.nii.gz`. Set the 'Sequence TR' to be 4.8, but leave all of the other options alone.
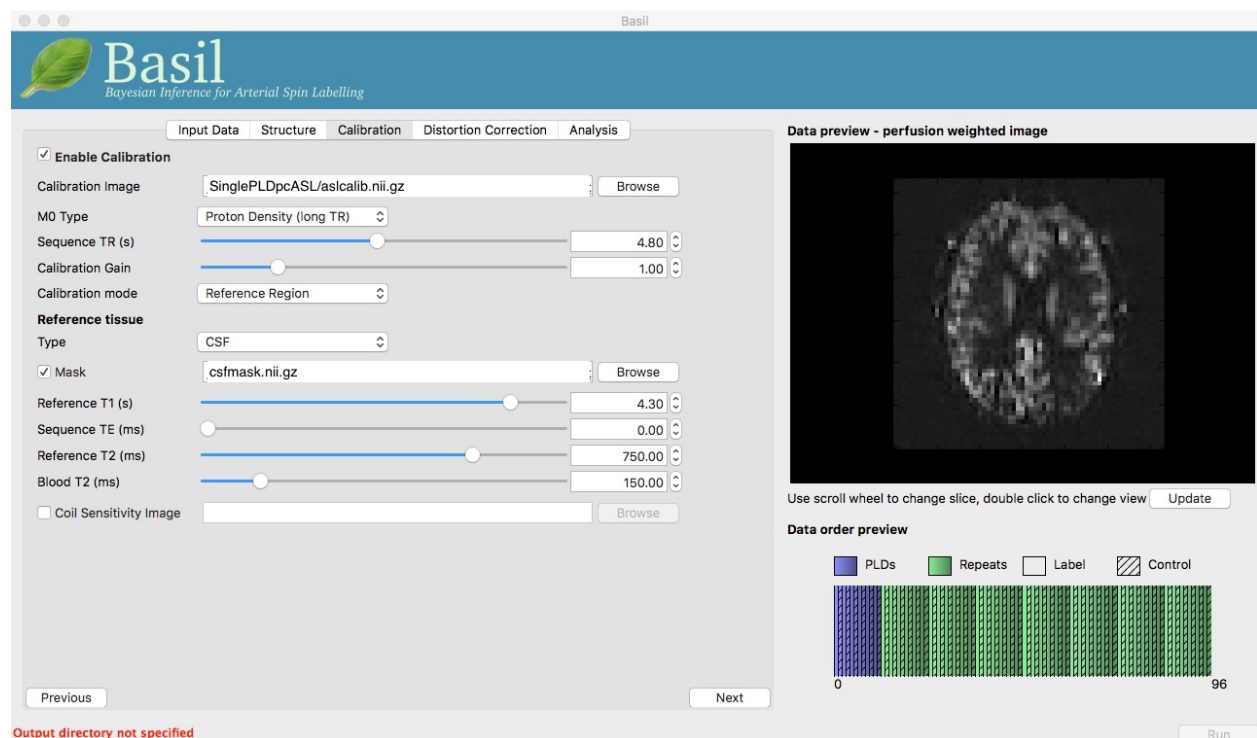
Move to the 'Distortion Correction' tab. Select 'Apply distortion correction'. Load the 'Phase-encode-reveresed calibration image' `aslcalib_PA.nii.gz` from the Single-PLD pcASL data. Set the 'Effective EPI echo spacing' to 0.95ms again and the 'Phase encoding direction' to 'y'.

Finally, move to the 'Analysis' tab. Choose an output directory, leave all of the other options alone. Click 'Run'.

This analysis shouldn't take a lot longer than the equivalent single PLD analysis, but feel free to skip ahead to the next section whilst you are waiting.

The results directory from this analysis should look similar to that obtained for the single PLD pcASL. That is reassuring as it is the same subject. The main difference is the `arrival.nii.gz` image. If you examine this image you should find a pattern of values that tells you the time it takes for blood to transit between the labeling and imaging regions. You might notice that the `arrival.nii.gz` image was present even in the single-PLD results, but if you looked at it contained a single value - the one set in the Analysis tab - which meant that it appeared blank in that case.

4.3.3 Arterial/Macrovascular Signal Correction

In the analysis above we didn't attempt to model the presence of arterial (macrovascular) signal. This is fairly reasonable for pcASL in general, since we can only start sampling some time after the first arrival of labeled blood-water in the imaging region. However, given we are using shorter PLD in our multi-PLD sampling to improve the SNR there is a much greater likelihood of arterial signal being present. Thus, we might like to repeat the analysis with this component included in the model.

Return to your analysis from before. On the 'Analysis' tab select 'Include macro vascular component'. Click 'Run'.

The results directory should be almost identical to the previous run, but now we also gain some new results:

- `aCBV.nii.gz` and

- `aCBV_calib.nii.gz`

Following the convention for the perfusion images, these are the relative and absolute arterial (cerebral) blood volumes respectively. If you examine one of these and focus on the more inferior slices you should see a pattern of higher values that map out the structure of the major arterial vasculature, including the Circle of Willis. This finding of an arterial contribution in some voxels results in a correction to the perfusion image - you may now be able to spot that in the same slices where there was some evidence for arterial contamination of the perfusion image before that has now been removed.

4.4 Partial Volume Correction

In the same way that we could do partial volume correction for single PLD pcASL, we can do this for multi-PLD. If anything partial volume correction should be even better for multi-PLD ASL, as there is more information in the data to separate grey and white matter perfusion.

Just like the single PLD case we will require structural information, entered on the 'Structure' tab. We can do as we did before and load `T1.anat`. On the 'Analysis' tab, select 'Partial Volume Correction'.

Again, this analysis will not be very quick and so you might not wish to click 'Run' right now.

You will find the results of this analysis already completed for you in the directory `~/fsl_course_data/ASL/oxasl_mpld_pvout`. This results directory contains, as a further subdirectory, `pvcorr`, within the `native` subdirectory, the partial volume corrected results: gray matter (`perfusion_calib.nii.gz` etc) and white matter perfusion (`perfusion_wm_calib.nii.gz` etc) maps. Alongside these there are also gray and white matter ATT maps (`arrival` and `arrival_wm` respectively). The estimated maps for the arterial component (`aCBV_calib.nii.gz` etc) are still present in the `pvcorr` directory. Since this is not tissue specific there are not separate gray and white matter versions of this parameter.

The End.

Example using vessel encoded pCASL data

This example shows how to process vessel encoded pCASL data with OXASL.

5.1 Obtaining and installing `VEASL`

Vessel encoded ASL can only be processed when the `VEASL` tool for vessel decoding is installed. Currently we are not able to distribute this tool publically, so if you are interested in processing VEPCASL data please contact Michael Chappell (michael.chappell@nottingham.ac.uk) to obtain a copy of the VEASL code.

VEASL must be installed in your path. The ideal installation locations are either:

- `$FSLDIR/bin` if you have access to `$FSLDIR`

- `$FSLDEVDIR/bin` if you have defined `$FSLDEVDIR`

- Anywhere else in your default `$PATH`

5.2 Running OXASL on VE data

The key distinction between running VE data and regular ASL data is the `--iaf` option. This should be set to one of:

- `--iaf=ve` for raw VEPCASL data

- `--iaf=vediff` for VEPCASL data which has undergone pairwise subtraction

In addition the following option must be provided for VE data:

- `--veslocs=<vessel locations file>`

The vessel locations file defines the initial locations for the encoded vessels - the first row gives the X co-ordinates, the second row the Y co-ordinates. For example this data defines 4 source vessels:

```
-25  25 -25  25
-13 -13 -24 -25
```

The encoding sequence is currently assumed to be determined from the estimated initial vessel locations.

An example command line for multi-PLD VEPCASL data would be:

```
oxasl -i mpld_asltc --casl --iaf=ve --ibf=tis --slicedt=0.0452 \
    --plds=0.25,0.5,0.75,1.0,1.25,1.5 --bolus=1.4 \
    --veslocs=veslocs.txt \
    -o oxasl_out
```

Other standard OXASL options can also be used to enable calibration, provide structural data, or apply preprocessing corrections to the data.

5.2.1 Pairwise subtracted VE data

By using `--iaf=vediff` it is possible to process pairwise-subtracted VE data. In general it is better to use the raw data, however some preprocessing strategies, such as denoising, may be better applied to subtracted data, where the static signal has been removed. **It is probably not a good idea to apply motion correction in this case as the subtracted images may only show signal in parts of the brain**. Do motion correction prior to subtraction instead.

5.3 Additional common VE options

5.3.1 `--nfpc=<Number of flows per class>`

VEASL can model the fact that each voxel may be fed by more than one vessel. A combination of feeder vessels is called a *class*. This option defines the number of vessels per class. The default value is 2.

5.3.2 `--infer-loc=none|xy|rigid`

This option controls how the locations of the vessels are inferred. `none` means they are fixed at their initial positions, `xy` means the X/Y co-ordinates are allowed to vary, and `rigid` infers a rigid body transformation of the co-ordinates. The default is `rigid`.

5.3.3 `--init-loc`

This option applies to multi-PLD data only. If specified an initial decoding run is performed on the mean across all PLDs to determine vessel locations. By default these vessel locations are then fixed when performing the vessel decoding on the individual PLDs. It is possible to initialize the vessel locations on the mean data and *also* infer them for each PLD using `--init-loc` together with the `--infer-loc-pld=none|xy|rigid` option.

5.4 The log output

The command line output is similar to the non-VE case, however there is an additional vessel decoding step between the preprocessing and the kinetic model inversion, which will look something like this (showing 1 PLD only):

```
Performing vessel decoding
- Initial vessel locations:
X: [-25.  25. -25.  25.]
Y: [-13. -13. -24. -25.]
```

<div align="right">(continues on next page)</div>

```
- Encoding matrix:
TWO
0.0, 0.0, 0.0, 0.0
0.0, 1.0, 0.0, 0.0
90.0, 2.0, -25.0, 25.0
90.0, 3.0, -25.0, 25.0
0.0, 2.0, -13.0, -24.5
0.0, 3.0, -13.0, -24.5
13.495733280795811, 2.0, -18.475358737629833, -7.29290476485388
13.495733280795811, 3.0, -18.475358737629833, -7.29290476485388
MAC:
0.0, 50.0, -0.0, -0.0, -3.0068078634963786, -0.3971255668768797
0.0, 8.974483684827054e-08, -18.75, -7.25, -12.52836611276087, -1.6546898639495466
0.0, 0.0, 270.0, 270.0, 76.5042667192042, 76.5042667192042
25.0, 25.0, 5.75, 5.75, 5.591226986387976, 5.591226986387976

- Fitting PLD 1
- Vessel locations (inference: rigid):
    X: [-24.36777377  25.63201797 -24.39952284  25.59738261]
    Y: [-14.52208469 -14.66639867 -25.52203887 -26.66634868]
    Translation: 0.616, -1.59  Rotation: -0.165 (degrees)
- Class proportions:
    [0.46011736 0.06254921 0.04755664 0.03730842 0.07797115 0.31449723]

DONE vessel decoding
```

The subsequent kinetic model inversion will then be performed on each vessel individually marked by the log messages:

```
Processing per-vessel decoded images

- Processing vessel 1
...etc
```

After each vessel has been individually model fitted, the output is combined for all vessels:

```
Generating combined images for all vessels
```

5.5 Output images

The output images are as usual found in the `oxasl_out/output` directory, however there is an additional layer not present for non-VE data. The `all_vessels` subdirectory contains the output for all vessels combined, while the `vessel<n>` directories contain the individual vessel outputs.

The usual OXASL output images are produced, for example:

- `perfusion.nii.gz` - This is the relative perfusion image

- `arrival.nii.gz` - This is the inferred bolus arrival time image

- `aCBV.nii.gz` - This is the inferred macrovascular signal image containing arterial volume fraction as a percentage

- `mask.nii.gz` - This is the binary brain mask used in the analysis

Calibrated outputs are also produced if calibration data is supplied, and structural space outputs are also produced where structural data is available.

5.5.1 How all-vessel output images are combined

The combination of single-vessel data into all-vessel data is not completely trivial. The following methods are used:

- For perfusion and macrovascular signal data, the output of the individual vessels is summed

- For arrival time and variance/STD outputs, the combined output is a weighted average of the individual vessels, weighted by the relative perfusion contribution from each vessel in each voxel.

5.6 Summary report

The summary report will contain, alongside the usual information, separate output images for each vessel and the combined output images:
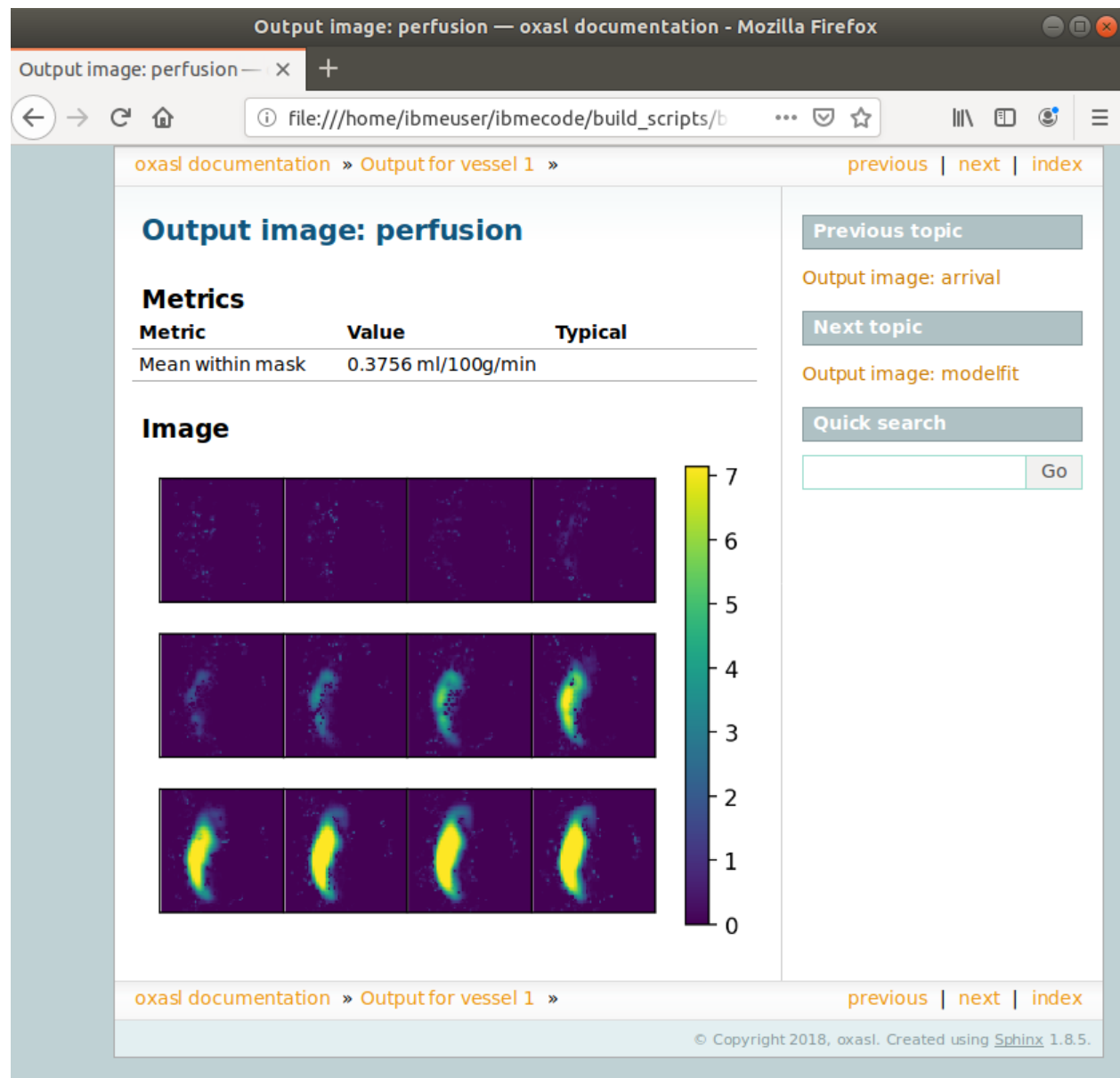
Example using multiphase ASL data

This example shows how to process multiphase data within OXASL. Multiphase data is collected at a range of inversion phases, so rather than pairs of tag/control images, one has a set of images covering angles between 0 and 180.

The multiphase plugin performs a multiphase fitting step after preprocessing but before running the kinetic model inversion. This step essentially reduces the multiphse data to a form equivalent to subtracted tag-control data.

6.1 Running OXASL on multiphase ASL data

The key distinction between running multiphase data and regular ASL data is the `--iaf` option which should be set to `--iaf=mp`.

In addition **one** of the following must be specified

- `--nphases=<Number of phases>`
- `--phases=<Comma separated list of phases in degrees>`

If `--nphases` is used, the phases are spread equally between 0 and 180 degrees.

A minimal example command line for multiphase ASL data would be:

```
oxasl -i data.nii.gz --casl --tau=1.4 --plds=0.4 \
      --iaf=mp --nphases=8 \
      --output=oxasl_mp
```

Other standard OXASL options can also be used to enable calibration, provide structural data, or apply preprocessing corrections to the data.

6.2 Additional options for multiphase data

6.2.1 `--mp-spatial`

This option uses spatial regularization during the multiphase fitting step. It is independent of the `--spatial` option which controls spatial regularization during the kinetic model fitting step.

6.2.2 `--mp-spatial-phase`

When using `--mp-spatial` this option places the spatial prior on the phase offset parameter which can give better spatial regularization than the default which performs spatial regularization on the magnitude.

6.3 `--mp-biascorr`

This option applies a bias correction to the multiphase fitting step. In the presence of significant noise, the multiphase fitting is biased towards higher magnitudes with larger amounts of noise. This option performs a multi-step process which aims to reduce the bias by initially fitting the phase offset and then fixing this while fitting the magnitude. The full method is described in:

- Msayib, Y., et al. *Robust Estimation of Quantitative Perfusion from Multi-Phase Pseudo-Continuous Arterial Spin Labelling.* Magnetic Resonance in Medicine, 2019.

6.4 `--mp-biascorr-sv`

Number of supervoxels in the bias-correction step

6.5 `--mp-biascorr-comp`

Supervoxel compactness in the bias correction step

6.6 `--mp-biascorr-sigma`

Pre-supervoxel smoothing parameter in the bias correction step

6.7 `--mp-options=<options file>`

This allows additional options to be passed to the multiphase fitting step (for example `max-iterations=20` to increase the number of iterations). The options should be placed in a text file.

6.8 The log output

The command line output is similar to the label-control case, however there is an additional multiphase fitting step between the preprocessing and the kinetic model inversion, which will look something like this (note depending on the multiphase options used there may be fewer steps performed):

```
Performing multiphase decoding:
- Using supervoxel-based bias correction
- Number of supervoxels: 20  - Compactness: 0.050000  - Pre-smoothing width: 0  -
→Step 1: Running initial biased fit 100% - DONE
- Step 2: Fitting mean signal in supervoxels 100% - DONE
- Step 3: Running final fit with fixed phase 100% - DONE

DONE multiphase decoding
```

6.9 Output images

The output images are as usual found in the `oxasl_out/output` directory. Output files from the multiphase fitting step are found in `oxasl_out/mp`.

The usual OXASL output images are produced, for example:

- `perfusion.nii.gz` - This is the relative perfusion image
- `arrival.nii.gz` - This is the inferred bolus arrival time image
- `aCBV.nii.gz` - This is the inferred macrovascular signal image containing arterial volume fraction as a percentage
- `mask.nii.gz` - This is the binary brain mask used in the analysis

Calibrated outputs are also produced if calibration data is supplied, and structural space outputs are also produced where structural data is available.

6.10 Summary report

Currently the multiphase fitting step does not generate any information in the summary report. This will be improved in the future!

OXASL command reference

The main `OXASL` command has many options, however most have sensible defaults so in practice only a few typically need to be used. For example usages see the Walkthrough section.

7.1 Full option list

Input ASL image:

-i ASLDATA, --asldata=ASLDATA ASL data file

--iaf=IAF input ASl format: diff=differenced,tc=tag-control,ct =control-tag,mp=multiphase,ve=vessel-encoded

--order=ORDER Data order as sequence of 2 or 3 characters: t=TIs/PLDs, r=repeats, l=labelling (tag/control/phases etc). First character is fastest varying

--tis=TIS TIs (s) as comma-separated list

--plds=PLDS PLDs (s) as comma-separated list - alternative to –tis

--ntis=NTIS Number of TIs (for use when processing does not require actual values)

--nplds=NPLDS Equivalent to –ntis

--rpts=RPTS Variable repeats as comma-separated list, one per TI/PLD

--nphases=NPHASES For –iaf=mp, number of phases (assumed to be evenly spaced)

--nenc=NENC For –iaf=ve, number of encoding cycles

--casl Acquisition was pseudo cASL (pcASL) rather than pASL

--tau=TAU, --taus=TAU, --bolus=TAU Bolus duration (s). Can be single value or comma separated list, one per TI/PLD

--slicedt=SLICEDT Timing difference between slices (s) for 2D readout

--sliceband=SLICEBAND Number of slices per pand in multi-band setup

--artsupp	Arterial suppression (vascular crushing) was used
--ibf=IBF	input block format - alternative to –order for compatibility. rpt=Blocks of repeats (i.e. repeats are slowest varying), tis=Blocsk of TIs/PLDs

Structural image:

-s STRUC, --struc=STRUC   Structural image

--struc-brain=STRUC_BRAIN, --sbet=STRUC_BRAIN, --struc-bet=STRUC_BRAIN   Structural image (brain extracted)

--struc2asl=STRUC2ASL   Structural->ASL transformation matrix

--asl2struc=ASL2STRUC   ASL->Structural transformation matrix

--wm-seg=WM_SEG   White matter segmentation of structural image

--gm-seg=GM_SEG   Grey matter segmentation of structural image

--csf-seg=CSF_SEG   CSF segmentation of structural image

--fslanat=FSLANAT   FSL_ANAT output directory for structural information

--fastsrc=FASTSRC   Images from a FAST segmentation - if not set FAST will be run on structural image

--struc2std=STRUC2STD   Structural to MNI152 linear registration (.mat)

--struc2std-warp=STRUC2STD_WARP   Structural to MNI152 non-linear registration (warp)

Main Options:

--wp	Analysis which conforms to the 'white papers' (Alsop et al 2014)
--mc	Motion correct data
--fixbat	Fix bolus arrival time
--fixbolus	Fix bolus duration
--artoff	Do not infer arterial component
--spatial-off	Do not include adaptive spatial smoothing on CBF

Acquisition/Data specific:

--bat=BAT	Estimated bolus arrival time (s) - default=0.7 (pASL), 1.3 (cASL)
--batsd=BATSD	Bolus arrival time standard deviation (s)
--t1=T1	Tissue T1 (s)
--t1b=T1B	Blood T1 (s)

Output options:

--save-corrected	Save corrected input data
--save-reg	Save registration information (transforms etc)
--save-basil	Save Basil modelling output
--save-calib	Save calibration output
--save-all	Save all output (enabled when –debug specified)

--output-stddev, --output-std   Output standard deviation of estimated variables

--output-var, --vars   Output variance of estimated variables

--no-report   Don't try to generate an HTML report

Calibration:

    -c CALIB, --calib=CALIB   Calibration image

    --calib-method=CALIB_METHOD, --cmethod=CALIB_METHOD   Calibration method: voxelwise or refregion

    --calib-alpha=CALIB_ALPHA, --alpha=CALIB_ALPHA   Inversion efficiency

    --calib-gain=CALIB_GAIN, --cgain=CALIB_GAIN   Relative gain between calibration and ASL data

    --tr=TR   TR used in calibration sequence (s)

Voxelwise calibration:

    --pct=PCT   Tissue/arterial partition coefficiant

    --t1t=T1T   T1 of tissue (s)

Reference region calibration:

    --mode=MODE   Calibration mode (longtr or satrevoc)

    --tissref=TISSREF   Tissue reference type (csf, wm, gm or none)

    --te=TE   Sequence TE (ms)

    --t1r=T1R   T1 of reference tissue (s) - defaults: csf 4.3, gm 1.3, wm 1.0

    --t2r=T2R   T2/T2* of reference tissue (ms) - defaults T2/T2*: csf 750/400, gm 100/60, wm 50/50

    --t2b=T2B   T2/T2* of blood (ms) - default T2/T2*: 150/50)

    --refmask=REFMASK   Reference tissue mask in perfusion/calibration image space

    --t2star   Correct with T2* rather than T2 (alters the default T2 values)

    --pcr=PCR   Reference tissue partition coefficient (defaults csf 1.15, gm 0.98, wm 0.82)

longtr mode (calibration image is a control image with a long TR):

satrecov mode (calibration image is a sequnce of control images at various TIs):

    --fa=FA   Flip angle (in degrees) for Look-Locker readouts

    --lfa=LFA   Lower flip angle (in degrees) for dual FA calibration

    --calib-nphases=CALIB_NPHASES   Number of phases (repetitions) of higher FA

    --fixa   Fix the saturation efficiency to 100% (useful if you have a low number of samples)

Registration:

    --regfrom=REGFROM   Registration image (e.g. perfusion weighted image)

Distortion correction using fieldmap:

    --fmap=FMAP   fieldmap image (in rad/s)

    --fmapmag=FMAPMAG   fieldmap magnitude image - wholehead extracted

    --fmapmagbrain=FMAPMAGBRAIN   fieldmap magnitude image - brain extracted

    --nofmapreg   Do not perform registration of fmap to T1 (use if fmap already in T1-space)

7.1. Full option list   **53**

Distortion correction using phase-encode-reversed calibration image (TOPUP):

--cblip=CBLIP   phase-encode-reversed (blipped) calibration image

General distortion correction options:

--echospacing=ECHOSPACING Effective EPI echo spacing (sometimes called dwell time) - in seconds

--pedir=PEDIR   Phase encoding direction, dir = x/y/z/-x/-y/-z

--gdcwarp=GDCWARP Additional warp image for gradient distortion correction - will be combined with fieldmap or TOPUP distortion correction

Sensitivity correction:

--cref=CREF   Reference image for sensitivity correction

--cact=CACT   Image from coil used for actual ASL acquisition (default: calibration image - only in longtr mode)

--isen=ISEN   User-supplied sensitivity correction in ASL space

--senscorr-auto, --senscorr Apply automatic sensitivity correction using bias field from FAST

--senscorr-off   Do not apply any sensitivity correction

Partial volume correction: –pvcorr Apply partial volume correction

Generic:

-o OUTPUT, --output=OUTPUT Output directory

--overwrite   Overwrite output directory if it already exists

-m MASK, --mask=MASK Brain mask image in ASL space

--optfile=OPTFILE File containing additional options

--debug   Debug mode

--version   show program's version number and exit

-h, --help   show help message and exit

OXASL API

8.1 Workspace module

OXASL - Workspace to store images and data associated with a processing pipeline

This class is conceptually simple - you can store pretty much any data by setting an attribute on the workspace and retrieve the resulting data as the same attribute. The workspace is backed by a directory and image data is save to files rather than store in memory.

This hides considerable complexity. Here are a few of the issues:

- To ensure that image data really is kicked out of memory we create an ImageProxy object for each image. This simply stores the filename and metadata associated with an image. `__getattribute__` is overridden to return ImageProxy attributes as the underlying Image.

- There is a special ImageProxy for an AslImage. This might go away if we can represent the full state of an AslImage using metadata alone.

Copyright (c) 2008-2020 Univerisity of Oxford

class `oxasl.workspace.`**`AslImageProxy`**(*fname*, *md=None*)
> Reference to a saved AslImage and it's metadata
>
> **`img`**()
> > Return an AslImage object from the file and stored metadata

class `oxasl.workspace.`**`ImageProxy`**(*fname*, *md=None*)
> Reference to a saved Image and it's metadata
>
> **`img`**()
> > Return an Image object by loading from the file

class `oxasl.workspace.`**`Workspace`**(*savedir=None*, *input_wsp='input'*, *parent=None*, *search_childs=('prequantify'*, *'filter'*, *'corrected'*, *'preproc'*, *'input')*, *auto_asldata=False*, ***kwargs*)
> A workspace for data processing

The contents of a workspace are modelled as attributes on the workspace object. An initial set of contents can be specified using keyword arguments. For command line tools, typically these are provided directly from the OptionParser results, e.g.

> options, args = parser.parse_args(sys.argv) wsp = Workspace(**vars(options))

A workspace is always associated with a physical directory. Certain types of objects are automatically be saved to the workspace. If no save directory is specified a temporary directory is created

Supported types are currently:

- `fsl.data.image.Image` - Saved as Nifti

- 2D Numpy array - Saved as ASCII matrix

All other attributes are serialized to YAML and stored in a special `_oxasl.yml` file.

To avoid saving a particular item, use the `add` method rather than directly setting an attribute, as it supports a `save` option.

ifnone(*attr*, *alternative*)
> Return the value of an attribute, if set and not None, or otherwise the supplied alternative

set_item(*name*, *value*, *save=True*, *save_name=None*, *save_fn=None*)
> Add an item to the workspace

> Normally this is achieved by assigning to an attribute directly, however this function exists to allow greater control where required.

> The item will be set as an attribute on the workspace If a save directory is configured and the value is a supported type it will be saved there. This can be disabled by setting save=False

> > **Parameters**
> >
> > - **name** – Name, must be a valid Python identifier
> >
> > - **value** – Value to set
> >
> > - **save** – If False, do not save item
> >
> > - **save_name** – If specified, alternative name to use for saving this item
> >
> > - **save_fn** – If specified, Callable which generates string representation of value suitable for saving the item to a file

sub(*name*, *parent_default=True*, ***kwargs*)
> Create a sub-workspace, (i.e. a subdir of this workspace)

> This inherits the log configuration from the parent workspace. The savedir will be a subdirectory of the original workspace. Additional data may be set using keyword arguments. The child-workspace will be available as an attribute on the parent workspace.

> > **Parameters**
> >
> > - **name** – Name of sub workspace
> >
> > - **parent_default** – If True, attribute values default to the parent workspace if not set on the sub-workspace

oxasl.workspace.**matrix_to_text**(*mat*)
> Convert matrix array to text using spaces/newlines as col/row delimiters

oxasl.workspace.**mkdir**(*dirname*, *fail_if_exists=False*, *warn_if_exists=True*, *log=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)
> Create a directory, including necessary subdirs

`oxasl.workspace.`**`text_to_matrix`**`(`*text*`)`
>   Convert space or comma separated file to matrix

8.2 AslImage module

OXASL - Classes for representing ASL data and constructing instances from command line parameters

Copyright (c) 2008-2020 Univeristy of Oxford

class `oxasl.image.`**`AslImage`**`(`*image*, *name=None*, ***kwargs*`)`
>   Subclass of fsl.data.image.Image which adds ASL structure information

>   An AslImage contains information about the structure of the data enabling it to perform operations such as reordering and label/control differencing.

>   As a *minimum* you must provide a means of determining the number of TIs/PLDs in the data. Specifying the data format and ordering explicitly is recommended, but a default ordering will be used (with a warning) if you do not.

>   Ordering can be defined in two ways:

>   1. Setting the `order` parameters to a sequence of characters (case insensitive):

>   - `l` - Labelling images (e.g. label/control pairs, sequence of multi-phases, vessel encoding cycles)
>   - `t` - TIs/PLDs
>   - `r` - Repeats
>   - `e` - TEs (optional)

>   The sequence is in order from fastest varying (innermost grouping) to slowest varying (outermost grouping). If `l` is not included this describes data which is already differenced.

>   2. Specifying the `ibf` option

>   - **`ibf` - `rpt` Blocked by repeats, i.e. first repeat of all TIs, followed by second repeat of all TIs...**
>   `tis` Blocked by TIs/PLDs, i.e. all repeats of first TI, followed by all repeats of second TI... When using –ibf, the labelling images (e.g. label/control pairs) are always adjacent

>   The data format is defined using the `iaf` parameter:

>   - `iaf` - `tc` = tag then control, `ct` = control then tag, `mp` = multiphase, `ve` = vessel encoded, `vediff` = Pairwise subtracted vessel encoded, `diff` = already differenced, hadamard` = Hadamard encoded, ``quant = already quantified

>   If oxasl_bids is installed, some/all of the image metadata can be extracted from a .json sidecar if one exists

>   **Variables**
>   - **nvols** – Number of volumes in data
>   - **iaf** – Data format - see above
>   - **order** – Data ordering string - see above
>   - **ntc** – Number of labelling images in data (e.g. 2 for TC pairs, 1 for differenced data, for vessel encoded or multiphase data the number of encoding cycles/phases, for Hadamard data the size of the Hadamard matrix, e.g. 8)
>   - **casl** – If True, acquisition was CASL/pCASL rather than PASL

- **phases** – List of phases for multiphase data (`iaf='mp'`)

- **ntis** – Number of TIs/PLDs

- **tis** – Optional list of TIs

- **plds** – Optional list of PLDs

- **have_plds** – True if PLDs were provided rather than TIs

- **tau** – Bolus durations - one per TI/PLD. If `have_plds` is True, tis are derived by adding the bolus duration to the PLDs

- **rpts** – Repeats, one value per TI (may be given as a constant but always stored as list)

- **slicedt** – Increase in TI/PLD per slice in z dimension for 2D acquisitions (default: 0)

- **sliceband** – Number of slices per band for multiband acquisitions (default: None)

- **artsupp** – If True, data was acquired with arterial suppression

derived(*image*, *name=None*, *suffix=''*, ***kwargs*)

Create a derived ASL image based on this one, but with different data

This is only possible if the number of volumes match, otherwise we cannot use the existing information about TIs, repeats etc. If the number of volumes do not match a generic Image is returned instead

Any further keyword parameters are passed to the AslImage constructor, overriding existing attributes, so this can be used to create a derived image with different numbers of repeats, etc, provided the data is consistent with this.

Note that the image space does not have to match, however in this case the 'header' kwarg should be passed to create the new image header

> **Parameters**
>
> - **data** – Numpy data for derived image
>
> - **name** – Name for new image (can be simple name or full filename)
>
> - **suffix** – If name not specified, construct by adding suffix to original image name
>
> **Returns** derived AslImage. However if the AslImage constructor fails, a basic fsl.data.image.Image is returned and a warning is output.

diff(*name=None*)

Perform tag-control subtraction.

Data will be reordered so the tag/control pairs are together

Note that currently differencing is not supported for multiphase or vessel encoded data. For Hadamard data, the output will be the decoded sub-boli images.

> **Parameters name** – Optional name for returned image. Defaults to original name with suffix `_diff`
>
> **Returns** AslImage instance containing differenced data

get_vol_index(*label_idx*, *ti_idx*, *rpt_idx*, *te_idx=0*, *order=None*)

Get the volume index for a specified label, TI and repeat index

> **Parameters**
>
> - **label_idx** – Label index starting from 0, e.g. for `iaf=ct` 0 would be the control image, for `iaf=mp` 3 would be the 4th phase encoded image
>
> - **ti_idx** – TI/PLD index, starting from 0

- **`rpt_idx`** – Repeat index, starting from 0
- **`te_idx`** – TE index for multi-TE data, starting from 0
- **`order`** – If specified use custom data ordering string (does not change ordering within this AslImage - use `reorder` for that)

`is_var_repeats`()

> **Returns** True if this data set has repeats which vary by time point

`mean`(*name=None*)
> Take the mean across all volumes
>
> This takes a naive mean without differencing or grouping by TIs
>
> > **Parameters** **name** – Optional name for returned image. Defaults to original name with suffix `_mean`
> >
> > **Returns** 3D fsl.data.image.Image. Not an AslImage as timing information has been lost

`mean_across_repeats`(*name=None*, *diff=True*)
> Calculate the mean ASL signal across all repeats
>
> > **Parameters**
> >
> > - **name** – Optional name for returned image. Defaults to original name with suffix _mean
> > - **diff** – If False do not difference the data before taking the mean (default: True)
> >
> > **Returns** Label-control subtracted AslImage with one volume per TI/PLD

`metadata_summary`()
> Generate a human-readable dictionary of metadata
>
> > **Returns** Dictionary mapping human readable metadata name to value (e.g. 'Label type' might map to 'Label-control pairs'). The keys and values are not guaranteed to be fixed and should not be parsed - use the instance attributes instead.

`perf_weighted`(*name=None*)
> Generate a perfusion weighted image by taking the mean over repeats and then the mean over TIs
>
> > **Parameters** **name** – Optional name for returned image. Defaults to original name with suffix `_pwi`
> >
> > **Returns** 3D fsl.data.image.Image. Not an AslImage as timing information lost

`reorder`(*out_order=None*, *iaf=None*, *name=None*)
> Re-order ASL data
>
> The order is defined by a string in which r=repeats, l=labelling images, and t=tis/plds. The first character is the fastest varying
>
> So for a tag-control data set with 3 TIs and 2 repeats an order of "ltr" would be: TC (TI1), TC (TI2), TC (TI3), TC(TI1, repeat 2), TC(TI2 repeat 2), etc.

`single_ti`(*ti_idx*, *order=None*, *name=None*)
> Extract the subset of data for a single TI/PLD
>
> FIXME will not correctly set have_plds flag in output if input has PLDs
>
> > **Parameters**
> >
> > - **ti_idx** – Index of the TI/PLD required starting from 0
> > - **order** – If specified the ordering string for the returned data

- **name** – Optional name for returned image. Defaults to original name with suffix `_ti<n>` where `<n>` is the TI/PLD index

    Returns AslImage instance containing data only for the selected TI/PLD

split_epochs(*epoch_size*, *overlap=0*, *time_order=None*)
    Split ASL data into 'epochs' of a specified size, with optional overlap

summary(*log=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)
    Write a summary of the data to a file stream

    Parameters **log** – Stream-like object to write the summary to. Defaults to `sys.stdout`

class oxasl.image.**Options**(*fname_opt='-i'*)
    OptionGroup which contains options for describing an ASL image

groups(*parser*)

    Parameters **parser** – OptionParser instance

    Returns Sequence of OptionGroup instances for this category of options

oxasl.image.**data_order**(*iaf*, *ibf*, *order*, *multite=False*)
    Determine the data format and ordering from `iaf` and `ibf` options

    If `iaf` is not specified (None or empty string), TC pairs are specified if the `order` parameter is given and contains the `l` character. Otherwise differenced data is assumed.

    If neither `ibf` nor `order` are specified, `ibf=rpt` is guessed.

    If `iaf` is not `diff` and `order` does not include the "l' character it is assumed that the encoded images (e.g. TC pairs) are the fastest varying.

    If any parameters had to be guessed (rather than inferred from other information) a warning is output.

    Returns Tuple of: IAF (ASL format, 'tc', 'ct', 'diff', 'quant', 've', 'vediff', 'hadamard' or 'mp'), ordering (sequence of 2 or three chars, 'l'=labelling images, 'r'=repeats, 't'=TIs/PLDs, 'e'=TEs. The characters are in order from fastest to slowest varying), Boolean indicating whether the block format needed to be guessed

oxasl.image.**summary**(*img*, *log=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)
    Write a summary of an Image to a stream

    For an AslImage the `summary` method is used which displays all the ASL data in human readable form. For generic Image objects just basic information is displayed

    Parameters

    - **img** – fsl.data.image.Image object which may or may not be an AslImage

    - **log** – Stream-like object - default is `sys.stdout`

8.3 Registration module

OXASL - Registration for ASL data

Copyright (c) 2008-2020 Univeristy of Oxford

class oxasl.reg.**Options**(***kwargs*)
    OptionCategory which contains options for registration of ASL data to structural image

groups(*parser*)

> **Parameters** `parser` – OptionParser instance

> **Returns** Sequence of OptionGroup instances for this category of options

`oxasl.reg.`**`change_space`**(*wsp*, *img*, *target_space*, *source_space=None*, ***kwargs*)
: Convert an image to a different space

Note that while the source space can be determined from the image, this may not be correct if images (e.g. ASL and calibration) share the same voxel->world transformation but still need registration to one another

> **Parameters**
>
> - **`wsp`** – Workspace object
>
> - **`img`** – Image
>
> - **`target_space`** – Either an Image in the target space, or the name of the target space
>
> - **`src_space`** – If specified, explicit indication of source image space

`oxasl.reg.`**`get_img_space`**(*wsp*, *img*)
: Find out what image space an image is in

Note that this only compares the voxel->world transformation matrix to the reference image for each space. It is quite possible for two images to be in the same space but not be registered to one another. In this case, the returned space may not be accurate when determining whether a registration is required.

> **Parameters**
>
> - **`wsp`** – Workspace object
>
> - **`img`** – Image

> **Returns** Name of image space for `img`, e.g. `asl`, `struc`

`oxasl.reg.`**`get_ref_imgs`**(*wsp*, *use_quantification_wsp=None*)
: Get the images that define the various processing 'spaces' and are used for registration to/from these spaces. The built in spaces are 'asl' (aka 'native') 'struc' and 'std' (MNI).

Note that the 'custom' space requires a user-specified reference image and transformation from structural space

aslref defines the 'asl' space

- `aslref` : User-supplied registration reference image

- `aslref_method` : Method for choosing registration reference image

- `asldata` : Raw ASL data

- `calib` : Calibration image

- `aslref` : Registration reference image in ASL space

- `strucref` : Registration reference image in structural space

- `stdref` : Registration reference image in standard space

`oxasl.reg.`**`get_transform_params`**(*mat*)
: Get motion parameters from a Flirt motion correction matrix

This is done under the assumption that the matrix may contain rotation, translation and possibly minor scaling but no reflection, shear etc. So the output could be incorrect for some extreme correction matrices, but this probably indicates an error in the registration process. We wrap the whole thing in a try block so if anything goes horribly wrong it does not at least stop the pipeline running

See http://en.wikipedia.org/wiki/Rotation_matrix for details of the rotation calculation.

> **Returns** Tuple of magnitude of translation, angle and rotation axis

oxasl.reg.**main**()
> Entry point for command line tool

oxasl.reg.**reg_asl2calib**(*wsp*)
> Register calibration image to ASL space
>
> Note that this might already have been done as part of motion correction

oxasl.reg.**reg_asl2custom**(*wsp*)
> Register custom output image to ASL space, via structural.
>
> If no output_custom_mat (struc -> custom) has been provided, then FLIRT will be used to generate this. The transformation from ASL space is the concatenation of asl2struc and struc2custom.

oxasl.reg.**reg_asl2struc**(*wsp*, *flirt=True*, *bbr=False*, *name='initial'*)
> Registration of ASL images to structural image
>
> > **Parameters**
> >
> > - **flirt** – If provided, sets whether to use FLIRT registration
> >
> > - **bbr** – If provided, sets whether to use BBR registration
>
> - `aslref` : Registration reference image in ASL space
>
> - `struc` : Structural image
>
> - `asl2struc` : ASL->structural transformation matrix
>
> - `struc2asl` : Structural->ASL transformation matrix
>
> - `regto` : `aslref` image transformed to structural space

oxasl.reg.**reg_bbr**(*wsp*)
> Perform BBR registration
>
> > **Parameters**
> >
> > - **reg_img** – Data to register, e.g. PWI or calibration image. Normally would be brain extracted
> >
> > - **struc_img** – Structural image
> >
> > - **struc_brain_img** – Brain-extracted structural image
>
> Optional keyword arguments:
>
> > **Parameters**
> >
> > - **inweight** –
> >
> > - **init** – Initial transform matrix
>
> Optional keyword arguments for fieldmap distortion correction:
>
> > **Parameters**
> >
> > - **fmap** – Fieldmap image
> >
> > - **fmapmag** – Fieldmap magnitude image
> >
> > - **fmapmagbrain** – Fieldmap magnitude image - brain extracted
> >
> > - **pedir** – Phase encoding direction (x, -x, y, -y, z, -z)

> • **echospacing** – Echo spacing

:return Tuple of registered image, transform matrix

oxasl.reg.**reg_flirt**(*wsp*, *img*, *ref*, *initial_transform=None*)
> Register low resolution ASL or calibration data to a high resolution structural image using Flirt rigid-body registration
>
> The brain extracted structural image is used as the reference image. If this is not supplied, BET will be run on the whole head structural image.
>
> > **Parameters**
> >
> > • **reg_img** – Data to register, e.g. PWI or calibration image. Normally would be brain extracted
> >
> > • **struc_brain_img** – Brain-extracted structural image
>
> Optional keyword arguments:
>
> > **Parameters**
> >
> > • **inweight** –
> >
> > • **init** – Initial transform matrix
> >
> > • **schedule** – FLIRT transform schedule file (default: xyztrans.sch")
> >
> > • **dof** – FLIRT degrees of freedom
>
> :return Tuple of registered image, transform matrix

oxasl.reg.**reg_struc2std**(*wsp*, ***kwargs*)
> Determine structural -> standard space registration
>
> • structural.struc : Structural image
>
> • fslanat : Path to existing FSLANAT data
>
> • reg.struc2std : Structural->MNI transformation matrix - either warp image or FLIRT matrix
>
> • reg.std2struc : MNI->structural transformation - either warp image or FLIRT matrix

oxasl.reg.**transform**(*wsp*, *img*, *trans*, *ref*, *use_flirt=False*, *interp='trilinear'*, *paddingsize=1*, *premat=None*, *postmat=None*, *mask=False*, *mask_thresh=0.5*)
> Transform an image
>
> > **Parameters**
> >
> > • **wsp** – Workspace, used for logging only
> >
> > • **img** – Image to transform
> >
> > • **trans** – Transformation matrix or warp image
> >
> > • **ref** – Reference image
> >
> > • **use_flirt** – Use flirt to apply the transformation which must be a matrix
> >
> > • **interp** – Interpolation method
> >
> > • **paddingsize** – Padding size in pixels
> >
> > • **premat** – If trans is a warp, this can be set to a pre-warp affine transformation matrix
>
> > **Returns** Transformed Image object

8.4 Structural module

OXASL - Structural data module

Copyright (c) 2008-2020 Univerisity of Oxford

class `oxasl.struc.`**`Options`**
> OptionGroup which contains options for describing a structural image

> **`groups`**(*parser*)

>> **Parameters** **`parser`** – OptionParser instance

>> **Returns** Sequence of OptionGroup instances for this category of options

`oxasl.struc.`**`run`**(*wsp*)
> Do initialization on supplied structural data - copy relevant image and do brain extraction

> FIXME copy across all supplied structural data

`oxasl.struc.`**`segment`**(*wsp*)
> Segment the structural image

8.5 Corrections module

OXASL - Module to apply Moco/Distortion/sensitivity corrections

This module generates corrected ASL/calibration data from previously calculated corrections with the minimum of interpolation

Currently the following sources of transformation exist:

- Motion correction of the ASL data. This generates a series of linear (rigid body) transformations in ASL space, one for each ASL volume. If calibration data is also present a calibration->ASL transform is also generated as part of this process

- Fieldmap-based distortion correction. This generates a nonlinear warp image in structural space which is then transformed to ASL space

- Phase encoding reversed (CBLIP) distortion correction using TOPUP. This generates a nonlinear warp image in ASL space FIXME calibration space?

- User-supplied nonlinear warp image for gradient distortion corection

- Sensitivity correction

Except for the TOPUP correction, all of the above can be combined in a single transformation to minimise interpolation of the ASL data

Copyright (c) 2008-2020 Univerisity of Oxford

class `oxasl.corrections.`**`Options`**
> Options for corrections of the input data

> **`groups`**(*parser*)

>> **Parameters** **`parser`** – OptionParser instance

>> **Returns** Sequence of OptionGroup instances for this category of options

`oxasl.corrections.`**`correct_img`**(*wsp*, *img*, *linear_mat*)
> Apply combined warp/linear transformations to an image in ASL space

> **Parameters**
>
> - **img** – fsl.data.image.Image to correct
> - **linear_mat** – img->ASL space linear transformation matrix.
>
> **Returns** Corrected Image

If a jacobian is present, also corrects for quantitative signal magnitude as volume has been locally scaled

FIXME there are slight differences to oxford_asl here due to use of spline interpolation rather than applyxfm4D which uses sinc interpolation.

- `asldata_mean` : Mean ASL image used as reference space

- `total_warp` : Combined warp image

- `jacobian` : Jacobian associated with warp image

- `senscorr` : Sensitivity correction

`oxasl.corrections.`**run**(*wsp*)

Apply distortion and motion corrections to ASL and calibration data

- `asldata_orig` : Uncorrected ASL data image

- `calib_orig` : Calibration image

- `cref_orig` : Calibration reference image

- `cblip_orig` : Calibration BLIP image

- `asldata_mc_mats` : ASL motion correction matrices

- `calib2asl` : Calibration -> ASL transformation matrix

- `distcorr_warp` : Distortion correction warp image

- `gdc_warp` : Gradient distortion correction warp image

- `asldata` : Corrected ASL data

- `calib` : Corrected calibration image

- `cref` : Corrected calibration reference image

- `cblip` : Corrected calibration BLIP image

8.6 OXASL - Python package for ASL-MRI analysis

Copyright (c) 2018 University of Oxford

The modules in this package are mostly Python replacements for existing shell script and C++ code from the oxford_asl FSL module.

For many tools, FSL is required. The `fslpy` package is used to wrap required tools such as BET and FAST.

8.6.1 Design

AslImage

The `oxasl.image.AslImage` class represents a captured ASL data file. It contains information about the acquisition (number/values of TIs/PLDs, repeats, ordering of label/control image etc). It also has methods which act directly on the data, for example performing tag-control subtraction, or generation of a perfusion-weighted image.:

```
img = AslImage("mpld.nii.gz", plds=[0.25, 0.5, 0.75, 1.0], iaf="tc", order='lrt')
diffdata = img.diff()
pwi = img.perf_weighted()
```

Workspaces

The `workspace` module contains the `Workspace` class which can be used to build a higher-level interface for complex workflows. A workspace is simply an object whose attributes are images, data, etc being used as part of the workflow. Unlike a normal object, requesting an attribute that does not exist returns None rather than raising an exception.:

```
wsp = Workspace()
print(wsp.asldata) # prints None
```

A workspace can optionally be associated with a physical directory. If it is, then setting attributes causes files to be saved in this directory for supported data types, such as images or 2D matrices.:

```
wsp = Workspace(savedir="myoutput")
wsp.asldata = AslImage("mpld.nii.gz", ntis=1) # Saves myoutput/asldata.nii.gz
```

A workspace is also associated with a log stream (sys.stdout by default) and a prepared logging dictionary `fsllog` for passing to FSL Python wrapper commands:

```
wsp = Workspace()
wsp.log.write("Hello World
```

") wsp.rois.mask = fslmaths(img).bin().run(log=wsp.fsllog)

Module functions

Other modules typically contains one or more functions which operate on a workspace, in some cases with additional parameters (but not always).

Module functions operate under the general rule that data stored directly as a workspace attribute is unprocessed, user-supplied data. Derived data is then stored in a sub-workspace. Module functions will usually create a sub-workspace to store their own output in, for example the `struc` module places it's output (brain extractions, segmentations) in the wsp.structural sub-workspace.

For example the `calib` module contains the `calibrate` function which calibrates a perfusion image to physical units using either voxelwise or reference region methods. It reads parameters required for this from the workspace, including the calibration method to use.

Most of these functions write their output back into the workspace under a standard name, however in some cases the function might be called on different input images and might therefore return an image directly, which can be added to the workspace by the caller under whatever name they prefer

Command line tools

Most modules include a `main()` function which implements a command line tool to wrap the module functionality. For example the `preproc` module implements the `oxasl_preproc` command line tool which can be used to do simple preprocessing of ASL data, such as the following to perform label-control subtraction:

```
oxasl_preproc -i asldata.nii.gz --nplds=5 --diff -o asldata_diff.nii.gz
```

Current ASL processing modules

- `basil` - ASL Bayesian Model fitting using the Fabber code
- `calib` - Calibration of perfusion data using voxelwise or reference region methods
- `corrections` - Calculate and apply corrections (motion, distortion)
- `mask` - Calculation of a suitable mask for brain data
- `pipeline` - Unified processing pipeline for ASL brain data
- `preproc` - Basic ASL preprocessing (label-control subtraction, etc)
- `reg` - Registration between ASL, structural and standard spaces
- `region_analysis` - Summary stats within ROIs

Other modules

- `image` - Definition of the main AslImage class
- `reporting` - Generation of HTML reports from processing operations
- `workspace` - Definition of the Workspace class

8.7 Indices and tables

- genindex
- modindex
- search

Python Module Index

o

Index

T

text_to_matrix() (*in module oxasl.workspace*), 56
transform() (*in module oxasl.reg*), 63

W

Workspace (*class in oxasl.workspace*), 55