PyNIfTI - Python-style access to NIfTI and ANALYZE files

Author:	Michael Hanke <michael.hanke@gmail.com>
Contact:	pkg-exppsy-pymvpa@lists.alioth.debian.org
Homepage:	http://niftilib.sf.net/pynifti
IRC:	#exppsy on OTFC/Freenode
Revision:	0.20081017.1

Table of Contents

1 What is NIfTI and what do I need PyNIfTI for?

1.1 NIfTI

NIfTI is a new Analyze-style data format, proposed by the NIfTI Data Format Working Group as a *"short-term measure to facilitate inter-operation of functional MRI data analysis software packages"*. Meanwhile a number of toolkits are NIfTI-aware (e.g. FSL, AFNI, SPM, Freesurfer and a to a certain degree also Brainvoyager). Additionally, dicomnifti allows the direct conversion from DICOM images into the NIfTI format.

With libnifti there is a reference implementation of a C library to read, write and manipulate NIfTI images. The library source code is put into the public domain and a corresponding project is hosted at SourceForge.

In addition to the C library, there is also an IO library written in Java and Matlab functions to make use of NIfTI files from within Matlab.

1.2 Python

Unfortunately, it is not that trivial to read NIfTI images with Python. This is particularly sad, because there is a large number of easy-to-use, high-quality libraries for signal processing available for Python (e.g. SciPy).

Moreover Python has bindings to almost any important language/program in the fields of maths, statistics and/or engineering. If you want to use R to calculate some stats in a Python script, simply use RPy and pass any data to R. If you don't care about R, but Matlab is your one and only friend, there are at least two different Python modules to control Matlab from within Python scripts. Python is the glue between all those helpers and the Python user is able to combine as many tools as necessary to solve a given problem -- the easiest way.

1.3 PyNIfTI

PyNIfTI aims to provide easy access to NIfTI images from within Python. It uses SWIG-generated wrappers for the NIfTI reference library and provides the `NiftiImage` class for Python-style access to the image data.

While PyNIfTI is not yet complete (i.e. doesn't support everything the C library can do), it already provides access to the most important features of the NIfTI-1 data format and *libniftiio* capabilities. The following features are currently implemented:

- PyNIfTI can read and write any file format supported by libniftiio. This includes NIfTI (single and pairs) as well as ANALYZE files, both also in gzipped versions.
- PyNIfTI provides fast and convenient access to the image data via NumPy arrays. This should enable users to process image data with most (if not all) numerical routines available for Python. The NumPy array automatically uses a datatype corresponding to the NIfTI image data -- no unnecessary upcasting is performed.
- PyNIfTI provides full read and write access to the NIfTI header data. Header information can be exported to a Python dictionary and can also be updated by using information from a dictionary.

- Besides accessing NIfTI data from files, PyNIfTI is able to create NIfTI images from NumPy arrays. The appropriate NIfTI header information is determined from the array properties. Additional header information can be optionally specified -- making it easy to clone NIfTI images if necessary, but with minor modifications.
- Most properties of NIfTI images are accessible via attributes and/or accessor functions of the `NiftiImage`. Inter-dependent properties are automatically updated if necessary (e.g. modifying the Q-Form matrix also updates the pixdim properties and quaternion representation).
- All properties are accessible via Python-style datatypes: A 4x4 matrix is an array not 16 individual numbers.
- PyNIfTI should be resonably fast. Image data will only be loaded into the memory if necessary. Simply opening a NIfTI file to access some header data is performed with virtually no delay independent of the size of the image. Unless image resizing or datatype conversion must be performed the image data can be shared by the NIfTI image and accessing NumPy arrays, and therefore memory won't be wasted memory with redundant copies of the image data. However, one should be careful to make a copy of the image data if you intend to resize and cast the image data (see the docstring of the `NiftiImage.asarray()` method).
- Additionally PyNIfTI can access uncompressed NIfTI or ANALYZE files by providing memory-mapped access to them via NumPy's memmap arrays. In this mode it is possible to modified existing files of any size without having to load them in memory first.

1.3.1 Scripts

Some functions provided by PyNIfTI also might be useful outside the Python environment and it might be useful to export them via some command line scripts.

Currently there is only one: `pynifti_pst` (pst: peristimulus timecourse). Using this script one can compute the signal timecourse for a certain condition for all voxels in a volume at once. Although it is done by simply averaging the timecourses of the involved events (nothing fancy), this might nevertheless be useful for exploring a dataset and accompanies similar tools like FSL's `tsplot`. The output of `pynifti_pst` can be loaded into FSLView to simultaneously look at statistics and signal timecourses. Please see the corresponding example below.

1.4 Known issues aka bugs

- PyNIfTI currently ignores the origin field of ANALYZE files - it is neither read nor written. A possible workaround is to convert ANALYZE files into the NIfTI format using FSL's `fslchfiletype`.

2 License

PyNIfTI is written by Michael Hanke as free software (both beer and speech) and licensed under the MIT License.

3 Get the sources

Since June 2007 PyNIfTI is part of the niftilibs family. The source code of PyNIfTI releases can be obtained from the corresponding Sourceforge project site. Alternatively, one can also download a tarball of the latest development snapshot (i.e. the current state of the *master* branch of the PyNIfTI source code repository).

If you want to have access to both, the full PyNIfTI history and the latest development code, you can use the PyNIfTI Git repository on the Alioth server, a service kindly provided by the Debian project. To view the repository, please point your web browser to gitweb:

http://git.debian.org/?p=pkg-exppsy/pynifti.git

The gitweb browser also allows to download arbitrary development snapshots of PyNIfTI. For a full clone (aka checkout) of the PyNIfTI repository simply do:

```
git clone http://git.debian.org/git/pkg-exppsy/pynifti.git
```

4 Installation

4.1 Binary packages

4.1.1 GNU/Linux

PyNIfTI is available in recent versions of the Debian (since lenny) and Ubuntu (since gutsy in universe) distributions. The name of the binary package is `python-nifti` in both cases.

- PyNIfTI versions in Debian
- PyNIfTI versions in Ubuntu

Binary packages for some additional Debian and (K)Ubuntu versions are also available. Please visit Michael Hanke's APT repository to read about how you have to setup your system to retrieve the PyNIfTI package via your package manager and stay in sync with future releases.

If you are using Debian lenny (or later) or Ubuntu gutsy (or later) or you have configured your system for Michael Hanke's APT repository all you have to do to install PyNIfTI is this:

```
apt-get update
apt-get install python-nifti
```

This should pull all necessary dependencies. If it doesn't, it's a bug that should be reported.

Additionally, there are binary packages for several RPM-based distributions, provided through the OpenSUSE Build Service. To install one of these packages first download it from the OpenSUSE software website. Please note, that this site does not only offer OpenSUSE packages, but also binaries for other distributions, including: CentOS 5, Fedora 9, Mandriva 2007-2008, RedHat Enterprise Linux 5, SUSE Linux Enterprise 10, OpenSUSE 10.2 up to 11.0. Once downloaded, open a console and invoke (the example command refers to PyMVPA 0.3.1):

```
rpm -i python-nifti-0.20080710.1-4.1.i386.rpm
```

The OpenSUSE website also offers 1-click-installations for distributions supporting it.

A more convenient way to install PyNIfTI and automatically receive software updates is to included one of the RPM-package repositories in the system's package management configuration. For e.g. OpenSUSE 11.0, simply use Yast to add another repository, using the following URL:

http://download.opensuse.org/repositories/home:/hankem/openSUSE_11.0/

For other distributions use the respective package managers (e.g. Yum) to setup the repository URL. The repositories include all dependencies of PyMIfTI, if they are not available from other repositories of the respective distribution.

4.1.2 Windows

A binary installer for a recent Python version is available from the nifticlibs Sourceforge project site.

There are a few Python distributions for Windows. In theory all of them should work equally well. However, I only tested the standard Python distribution from www.python.org (with version 2.5.2).

First you need to download and install Python. Use the Python installer for this job. Yo do not need to install the Python test suite and utility scripts. From now on we will assume that Python was installed in *C:\Python25* and that this directory has been added to the *PATH* environment variable.

In addition you'll need NumPy. Download a matching NumPy windows installer for your Python version (in this case 2.5) from the SciPy download page and install it.

PyNIfTI does not come with the required *zlib* library, so you also need to download and install it. A binary installer is available from the GnuWin32 project. Install it in some arbitrary folder (just the binaries nothing else), find the *zlib1.dll* file in the *bin* subdirectory and move it in the Windows *system32* directory.

Now, you can use the PyNIfTI windows installer to install PyNIfTI on your system. As always: click *Next* as long as necessary and finally *Finish*. If done, verify that everything went fine by opening a command promt and start Python by typing *python* and hit enter. Now you should see the Python prompt. Import the nifti module, which should cause no error messages:

```
>>> import nifti
>>>
```

4.1.3 Macintosh

A binary package for a recent Python version is provided in the download area of the SourceForge project.

4.2 Compile from source: General instructions

If no binary packages are provided for your platfom, you can build PyNIfTI from source. It needs a few things to build and run properly:

- Python 2.4 or greater
- NumPy
- SWIG
- **NIfTI C libraries** Proper developer packages are prefered, but for convenience reasons a minimal copy is included in the PyNIfTI source package.

Make sure that the compiled nifticlibs and the corresponding headers are available to your compiler. If they are located in a custom directory, you might have to specify `--include-dirs` and `--library-dirs` options to the build command below. In case, you want to build and use the nifticlibs copy that is shipped with PyNIfTI, this is automatically done for you.

Once you have downloaded the sources, extract the tarball and enter the root directory of the extracted sources. If you *do not* have the nifticlibs installed, run:

```
make
```

in the root of the extracted source tarball. If you have system-wide installed nifticlibs available on your system, instead simply do:

```
python setup.py build
```

That should build the SWIG wrappers. If this has been done successfully, all you need to do is install the modules by invoking:

```
sudo python setup.py install
```

If sudo is not configured (or even installed) you might have to use `su` instead.

Now fire up Python and try importing the module to see if everything is fine. It should look similar to this:

```
Python 2.4.4 (#2, Oct 20 2006, 00:23:25)
[GCC 4.1.2 20061015 (prerelease) (Debian 4.1.1-16.1)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import nifti
>>>
```

4.2.1 Building on Windows Systems

On Windows the whole situation is a little more tricky, as the system doesn't come with a compiler by default. Nevertheless, it is easily possible to build PyNIfTI from source. One could use the Microsoft compiler that comes with Visual Studio to do it, but as this is commercial software and not everybody has access to it, I will outline a way that exclusively involves free and open source software.

First one needs to install the Python and NumPy, if not done yet. Please refer to the installation intructions for the Windows binary package below.

Next we need to obtain and install the MinGW compiler collection. Download the *Automated MinGW Installer* from the MinGW project website. Now, run it and choose to install the *current* package. You will need the *MinGW base tools*, *gcc* **and** *g++* compiler and *MinGW Make*. For the remaining parts of the section, we will assume that MinGW got installed in *C:\MinGW* and the directory *C:\MinGW\bin* has been added to the *PATH* environment variable, to be able to easily access all MinGW tools. Note, that it is not necessary to install MSYS to build PyNIfTI, but it might handy to have it.

In addition, PyNIfTI needs the developer version of the *zlib* library, so you also need to download and install it. A binary installer is available from the GnuWin32 project. It is best to install it into the same directory as MinGW (i.e. *C:\MinGW* in this example), as all paths will be automatically configured properly.

You also need to download SWIG (actually *swigwin*, the distribution for Windows). SWIG does not have to be installed, just unzip the file you downloaded and add the root directory of the extracted sources to the *PATH* environment variable (make sure that this directory contains *swig.exe*, if not, you haven't downloaded *swigwin*).

PyNIfTI comes with a specific build setup configuration for Windows -- *setup.cfg.mingw32* in the root of the source tarball. Please rename this file to *setup.cfg*. This is only necessary, if you have *not* configured your Python distutils installation to always use MinGW instead of the Microsoft compilers.

Now, we are ready to build PyNIfTI. The easiest way to do this, is to make use of the *Makefile.win* that is shipped with PyNIfTI to build a binary installer package (*.exe*). Make sure, that the settings at the top of *Makefile.win* (the file is located in the root directory of the source distribution) correspond to your Python installation -- if not, first adjust them accordingly before your proceed. When everything is set, do:

```
mingw32-make -f Makefile.win installer
```

Upon success you can find the installer in the *dist* subdirectory. Install it as described below.

4.2.2 MacOS X and MacPython

When you are comiling PyNIfTI on MacOS X and want to use it with MacPython, please make sure that the NIfTI C libraries are compiled as fat binaries (compiled for both *ppc* and *i386*). Otherwise PyNIfTI extensions will not compile.

One can achieve this by adding both architectures to the `CFLAGS` definition in the toplevel Makefile of the NIfTI C library source code or in the file *3rd/nifticlibs/Makefile* if you are using the nifticlibs copy that is shipped with the PyNIfTI sources. Like this:

```
CFLAGS=-Wall -O2 -I. -DHAVE_ZLIB -arch ppc -arch i386
```

4.2.3 Troubleshooting

If you get an error when importing the `nifti` module in Python complaining about missing symbols your niftiio library contains references to some unresolved symbols. Try adding `znzlib` and `zlib` to the linker options the PyNIfTI `setup.py`, like this:

```
libraries = [ 'niftiio', 'znz', 'z' ],
```

5 Documentation

A printable version of this documentation is available in PDF format:

http://niftilib.sourceforge.net/pynifti/manual.pdf

Additonally, there is an EpyDoc generated API documentation.

5.1 Things to know

When accessing NIfTI image data through NumPy arrays the order of the dimensions is reversed. If the *x*, *y*, *z*, *t* dimensions of a NIfTI image are 64, 64, 32, 456 (as for example reported by `nifti_tool`), the shape of the NumPy array (e.g. as returned by `NiftiImage.data`) will be: 456, 32, 64, 64.

This is done to be able to slice the data array much easier in the most common cases. For example, if you are interested in a certain volume of a timeseries it is much easier to write `data[2]` instead of `data[:,:,:,2]`, right?

5.2 Examples

The next sections contains some examples showing ways to use PyNIfTI to read and write imaging data from within Python to be able to process it with some random Python library.

All examples assume that you have imported the PyNIfTI module by invoking:

```
from nifti import *
```

5.2.1 Fileformat conversion

Open the MNI standard space template that is shipped with FSL. No filename extension is necessary as libniftiio determines it automatically:

```
>>> nim = NiftiImage('avg152T1_brain')
```

The filename is available via the 'filename' attribute:

```
>>> print nim.filename
avg152T1_brain.img
```

This indicates an ANALYZE image. If you want to save this image as a single gzipped NIfTI file simply do:

```
>>> nim.save('mni.nii.gz')
```

The filetype is determined from the filename. If you want to save to gzipped ANALYZE file pairs instead the following would be an alternative to calling the `save()` with a new filename:

```
>>> nim.filename = 'mni_analyze.img.gz'
>>> nim.save()
```

Please see the docstring of the `NiftiImage.setFilename()` method to learn how the filetypes are determined from the filenames.

5.2.2 NIfTI files from array data

The next code snipped demonstrates how to create a 4d NIfTI image containing gaussian noise. First we need to import the NumPy module

```
>>> import numpy as N
```

Now generate the noise dataset. Let's generate noise for 100 volumes with 16 slices and a 32x32 inplane matrix.

```
>>> noise = N.random.randn(100,16,32,32)
```

Please notice the order in which the dimensions are specified: (t, z, y, x).
The datatype of the array will most likely be *float64* -- which can be verified by invoking `noise.dtype`.
Converting this dataset into a NIfTI image is done by invoking the `NiftiImage` constructor with the noise dataset as argument:

```
>>> nim = NiftiImage(noise)
```

The relevant header information is extracted from the NumPy array. If you query the header information about the dimensionality of the image, it returns the desired values:

```
>>> print nim.header['dim']
[4, 32, 32, 16, 100, 0, 0, 0]
```

First value shows the number of dimensions in the datset: 4 (good, that's what we wanted). The following numbers are dataset size on the x, y, z, t, u, v, w axis (NIfTI files can handle up to 7 dimensions). Please notice, that the order of dimensions is now 'correct': We have 32x32 inplane resolution, 16 slices in z direction and 100 volumes.
Also the datatype was set appropriately:

```
>>> nim.header['datatype'] == nifticlib.NIFTI_TYPE_FLOAT64
True
```

To save the noise file to disk, just call the `save()` method:

```
>>> nim.save('noise.nii.gz')
```

5.2.3 Select ROIs

Suppose you want to have the first ten volumes of the noise dataset we have just created in a separate file. First open the file (can be skipped if it is still open):

```
>>> nim = NiftiImage('noise.nii.gz')
```

Now select the first ten volumes and store them to another file, while preserving as much header information as possible

```
>>> nim2 = NiftiImage(nim.data[:10], nim.header)
>>> nim2.save('part.hdr.gz')
```

The `NiftiImage` constructor takes a dictionary with header information as an optional argument. Settings that are not determined by the array (e.g. size, datatype) are copied from the dictionary and stored to the new NIfTI image.

5.2.4 Linear detrending of timeseries (SciPy module is required for this example)

Let's load another 4d NIfTI file and perform a linear detrending, by fitting a straight line to the timeseries of each voxel and substract that fit from the data. Although this might sound complicated at first, thanks to the excellent SciPy module it is just a few lines of code.

```
>>> nim = NiftiImage('timeseries.nii')
```

Depending on the datatype of the input image the detrending process might change the datatype from integer to float. As operations that change the (binary) size of the NIfTI image are not supported, we need to make a copy of the data and later create a new NIfTI image.

```
>>> data = nim.asarray()
```

Now detrend the data along the time axis. Remember that the array has the time axis as its first dimension (in contrast to the NIfTI file where it is the 4th).

```
>>> from scipy import signal
>>> data_detrended = signal.detrend( data, axis=0 )
```

Finally, create a new NIfTI image using header information from the original source image.

```
>>> nim_detrended = NiftiImage( data_detrended, nim.header)
```

5.2.5 Make a quick plot of a voxels timeseries (Gnuplot module is required)

Plotting is essential to get a 'feeling' for the data. The Gnuplot python bindings make it really easy to plot something with Gnuplot (e.g. when running Python interactively via IPython). Please note, that there are many other possibilities for plotting, e.g. R via RPy or Matlab-style plotting via matplotlib.

However, using Gnuplot is really easy. First import the Gnuplot module and create the interface object

```
>>> from Gnuplot import Gnuplot
>>> gp = Gnuplot()
```

We want the timeseries as a line plot and not just the datapoints, so let's talk with Gnuplot

```
>>> gp('set data style lines')
```

now load a 4d NIfTI image

```
>>> nim = NiftiImage('perfect_subject.nii.gz')
```

and finally plot the timeseries of voxel (x=20, y=30, z=12)

```
>>> gp.plot(nim.data[:,12,30,20])
```

A Gnuplot window showing the timeseries should popup now. Please refer to the Gnuplot manual to learn what it can do -- and it can do a lot more than just simple line plots (have a look at some Gnuplot demos if you are interested).

5.2.6 Show a slice of a 3d volume (Matplotlib module is required)

This example demonstrates howto use the Matlab-style plotting of matplotlib to view a slice from a 3d volume.

This time I assume that a 3d nifti file is already opened and available in the `nim3d` object. At first we need to load the necessary Python module.

```
>>> from pylab import *
```

If everything went fine, we can now view a slice (x,y):

```
>>> imshow(nim3d.data[200], interpolation='nearest', cmap=cm.gray)
>>> show()
```

It is necessary to call the `show()` function one time after importing pylab to actually see the image when running Python interactively.

When you want to have a look at a yz-slice, NumPy array magic comes into play.

```
>>> imshow(nim3d.data[::-1,:,100], interpolation='nearest', cmap=cm.gray)
```

The `::-1` notation causes the z-axis to be flipped in the images. This makes a much nicer view, if the used example volume has the z-axis originally oriented upsidedown.

5.2.7 Compute and display peristimulus signal timecourse of multiple conditions

Sometimes one wants to look at the signal timecourse of some voxel after a certain stimulation onset. An easy way would be to have some fMRI data viewer that displays a statistical map and one could click on some activated voxel and the peristimulus signal timecourse of some condition in that voxel would be displayed.

This can easily be done by using `pynifti_pst` and `FSLView`.

`pynifti_pst` comes with a manpage that explains all options and arguments. Basically `pynifti_pst` needs a 4d image (e.g. an fMRI timeseries; possibly preprocessed/filtered) and some stimulus onset information. This information can either be given directly on the command line or is read from files. Additionally one can specify onsets as volume numbers or as onset times.

`pynifti_pst` understands the FSL custom EV file format so one can easily use those files as input.

An example call could look like this:

```
pynifti_pst --times --nvols 5 -p uf92.feat/filtered_func_data.nii.gz \
    pst_cond_a.nii.gz uf92.feat/custom_timing_files/ev1.txt \
    uf92.feat/custom_timing_files/ev2.txt
```

This computes a peristimulus timeseries using the preprocessed fMRI from a FEAT output directory and two custom EV files that both together make up condition A. `--times` indicates that the EV files list onset times (not volume ids) and `--nvols` requests the mean peristimulus timecourse for 4 volumes after stimulus onset (5 including onset). `-p` recodes the peristimulus timeseries into percent signalchange, where the onset is always zero and any following value is the signal change with respect to the onset volume.

This call produces a simple 4d NIfTI image that can be loaded into FSLView as any other timeseries. The following call can be used to display an FSL zmap from the above results path on top of some anatomy. Additionally the peristimulus timeseries of two conditions are loaded. The figure shows how it could look like. One of the nice features of FSLView is that its timeseries window can remember selected curves, which can be useful to compare signal timecourses from different voxels (blue and green line in the figure).

6 PyNIfTI Development Changelog

Modifications are done by Michael Hanke, if not indicated otherwise. 'Closes' statement IDs refer to the Debian bug tracking system and can be queried by visiting the URL:

```
http://bugs.debian.org/<bug id>
```

The full VCS changelog is available here:

http://git.debian.org/?p=pkg-exppsy/pynifti.git;a=shortlog;h=upstream

Unreleased changes Changes described here are not yet released, but available from VCS repository.

None yet.

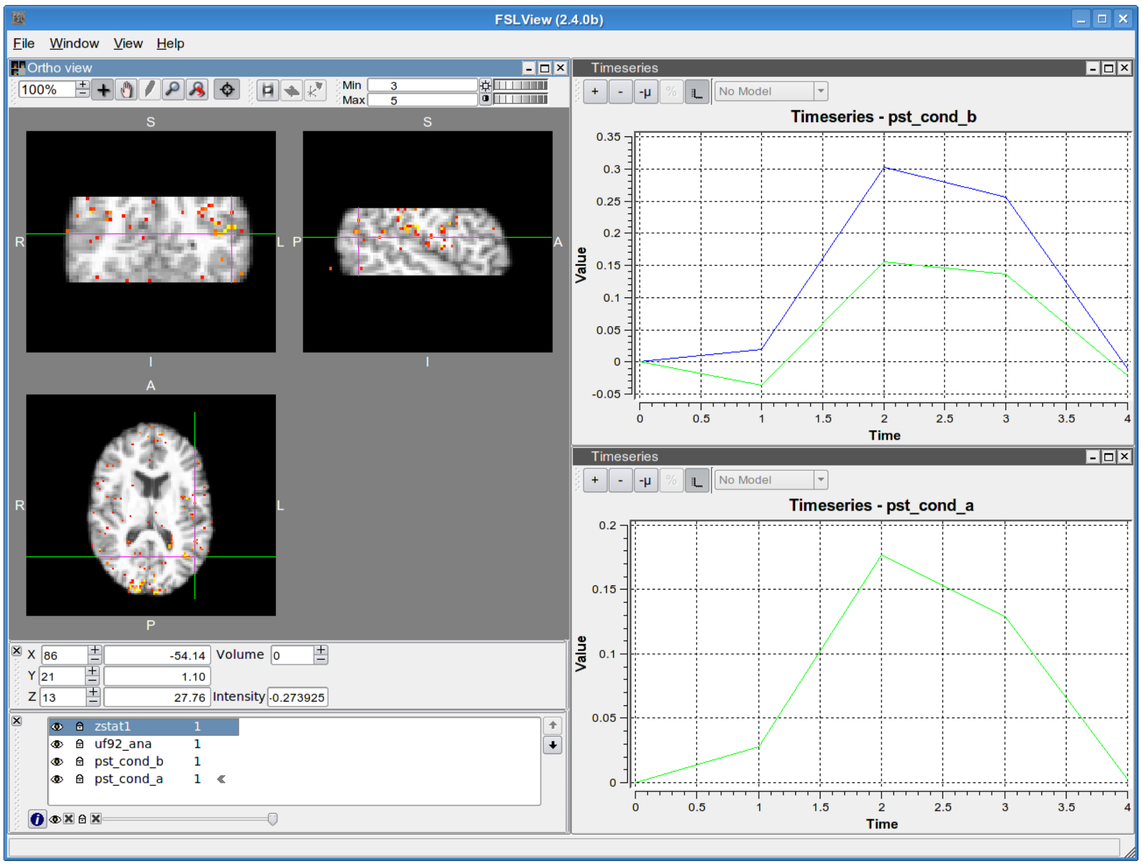

Figure 1: FSLView with `pynifti_pst` example.

6.1 Releases

- 0.20081017.1 (Fri, 17 Oct 2008)
  - Updated included minimal copy of the nifticlibs to version 1.1.0.
  - Few changes to the Makefiles to enhance Posix compatibility. Thanks to Chris Burns.
  - When building on non-Debian systems, only add include and library paths pointing to the local nifticlibs copy, when it is actually built. On Debian system the local copy is still not used at all, as a proper nifticlibs package is guaranteed to be available.
  - Added minimal setup_egg.py for setuptools users. Thanks to Gaël Varoquaux.
  - PyNIfTI now does a proper wrapping of the image data with NumPy arrays, which no longer leads to accidental memory leaks, when accessing array data that has not been copied before (e.g. via the *data* property of NiftiImage). Thanks to Gaël Varoquaux for mentioning this possibility.
- 0.20080710.1 (Thu, 7 Jul 2008)
  - Bugfix: Pointer bug introduced by switch to new NumPy API in 0.20080624 Thanks to Christopher Burns for fixing it.
  - Bugfix: Honored DeprecationWarning: sync() -> flush() for memory mapped arrays. Again thanks to Christopher Burns.
  - More unit tests and other improvements (e.g. fixed circular imports) done by Christopher Burns.
- 0.20080630.1 (Tue, 30 Jun 2008)
  - Bugfix: NiftiImage caused a memory leak by not calling the NiftiFormat destructor.
  - Bugfix: Merged bashism-removal patch from Debian packaging.
- 0.20080624.1 (Tue, 24 Jun 2008)
  - Converted all documentation (including docstrings) into the restructured text format.
  - Improved Makefile.
  - Included configuration and Makefile support for profiling, API doc generation (via epydoc) and code quality checks (with PyLint).
  - Consistently import NumPy as N.
  - Bugfix: Proper handling of [qs]form codes, which previously have not been handled at all. Thanks to Christopher Burns for pointing it out.
  - Bugfix: Make NiftiFormat work without setFilename(). Thanks to Benjamin Thyreau for reporting.
  - Bugfix: setPixDims() stored meaningless values.
  - Use new NumPy API and replace deprecated function calls (*PyArray_FromDimsAndData*).
  - Initial support for memory mapped access to uncompressed NIfTI files (*MemMappedNifti-Image*).
  - Add a proper Makefile and setup.cfg for compiling PyNIfTI under Windows with MinGW.
  - Include a minimal copy of the most recent nifticlibs (just libniftiio and znzlib; version 1.0), to lower the threshold to build PyNIfTI on systems that do not provide a developer package for those libraries.
- 0.20070930.1 (Sun, 30 Sep 2007)
  - Relicense under the MIT license, to be compatible with SciPy license. http://www.opensource.org/licenses/mit-license.php
  - Updated documentation.
- 0.20070917.1 (Mon, 17 Sep 2007)

  - Bugfix: Can now update NIfTI header data when no filename is set (Closes: #442175).
  - Unloading of image data without a filename set is no checked and prevented as it would damage data integrity and the image data could not be recovered.
  - Added 'pixdim' property (Yaroslav Halchenko).

- 0.20070905.1 (Wed, 5 Sep 2007)
  - Fixed a bug in the qform/quaternion handling that caused changes to the qform to vanish when saving to file (Yaroslav Halchenko).
  - Added more unit tests.
  - 'dim' vector in the NIfTI header is now guaranteed to only contain non-zero elements. This caused problems with some applications.

- 0.20070803.1 (Fri, 3 Aug 2007)
  - Does not depend on SciPy anymore.
  - Initial steps towards a unittest suite.
  - pynifti_pst can now print the peristimulus signal matrix for a single voxel (onsets x time) for easier processing of this information in external applications.
  - utils.getPeristimulusTimeseries() can now be used to compute mean and variance of the signal (among others).
  - pynifti_pst is able to compute more than just the mean peristimulus timeseries (e.g. variance and standard deviation).
  - Set default image description when saving a file if none is present.
  - Improved documentation.

- 0.20070425.1 (Wed, 25 Apr 2007)
  - Improved documentation. Added note about the special usage of the header property. Also added notes about the relevant properties in the docstring of the corresponding accessor methods.
  - Added property and accessor methods to access/modify the repetition time of timeseries (dt).
  - Added functions to manipulate the pixdim values.
  - Added utils.py with some utility functions.
  - Added functions/property to determine the bounding box of an image.
  - Fixed a bug that caused a corrupted sform matrix when converting a NumPy array and a header dictionary into a NIfTI image.
  - Added script to compute peristimulus timeseries (pynifti_pst).
  - Package now depends on python-scipy.

- 0.20070315.1 (Thu, 15 Mar 2007)
  - Removed functionality for "NiftiImage.save() raises an IOError exception when writing the image file fails." (Yaroslav Halchenko)
  - Added ability to force a filetype when setting the filename or saving a file.
  - Reverse the order of the 'header' and 'load' argument in the NiftiImage constructor. 'header' is now first as it seems to be used more often.
  - Improved the source code documentation.
  - Added getScaledData() method to NiftiImage that returns a copy of the data array that is scaled with the slope and intercept stored in the NIfTI header.

- 0.20070301.2 (Thu, 1 Mar 2007)
  - Fixed wrong link to the source tarball in README.html.

- 0.20070301.1 (Thu, 1 Mar 2007)
  - Initial upload to the Debian archive. (Closes: #413049)
  - NiftiImage.save() raises an IOError exception when writing the image file fails.
  - Added extent, volextent, and timepoints properties to NiftiImage class (Yaroslav Halchenko).
- 0.20070220.1 (Tue, 20 Feb 2007)
  - NiftiFile class is renamed to NiftiImage.
  - SWIG-wrapped libniftiio functions are no available in the nifticlib module.
  - Fixed broken NiftiImage from Numpy array constructor.
  - Added initial documentation in README.html.
  - Fulfilled a number of Yarik's wishes ;)
- 0.20070214.1 (Wed, 14 Feb 2007)
  - Does not depend on libfslio anymore.
  - Up to seven-dimensional dataset are supported (as much as NIfTI can do).
  - The complete NIfTI header dataset is modifiable.
  - Most image properties are accessable via class attributes and accessor methods.
  - Improved documentation (but still a long way to go).
- 0.20061114 (Tue, 14 Nov 2006)
  - Initial release.