Nitime: time-series analysis for neuroimaging data

Ariel Rokem (arokem@berkeley.edu) – *University of California, Berkeley, Berkeley, CA* USA
Michael Trumpis (mtrumpis@berkeley.edu) – *University of California, Berkeley, Berkeley, CA* USA
Fernando Pérez (Fernando.Perez@berkeley.edu) – *University of California, Berkeley, Berkeley, CA* USA

Nitime is a library for the analysis of time-series developed as part of the Nipy project, an effort to build open-source libraries for neuroimaging research. While nitime is developed primarily with neuroimaging data in mind (espespecially functional Magnetic Resonance Imaging data), its design is generic enough that it should be useful to other fields with experimental time-series. The package starts from a purely functional set of algorithms for time-series analysis, including spectral transforms, event-related analysis and coherency. An object-oriented layer is separated into lightweight data container objects for the representation of time-series data and high-level analyzer objects that couple data storage and algorithms. Each analyzer is designed to deal with a particular family of analysis methods and exposes a high-level object oriented interface to the underlying numerical algorithms. We briefly describe functional neuroimaging and some of the unique considerations applicable to time-series analysis of data acquired using these techniques, and provide examples of using nitime to analyze both synthetic data and real-world neuroimaging time-series.

Introduction

`Nitime` (http://nipy.sourceforge.net/nitime) is a library for time-series analysis of data from neuroimaging experiments, with a design generic enough that it should be useful for a wide wide array of tasks involving experimental time-series data from any source.

`Nitime` is one of the components of the NiPy [NiPy] project, an effort to develop a set of open-source libraries for the analysis and visualization of data from neuroimaging experiments.

Functional MRI: imaging brain activity

One of the major goals of neuroscience is to understand the correspondence between human behavior and activity occurring in the brain. For centuries, physicians and scientists have been able to identify brain areas participating in various cognitive functions, by observing the behavioral effects of damage to those areas. Within the last $\sim$ 25 years, imaging technology has advanced enough to permit the observation of brain activity *in-vivo*. Among these methods, collectively known as functional imaging, fMRI (functional Magnetic Resonance Imaging) has gained popularity due to its combination of low invasiveness, relatively high spatial resolution with whole brain acquisition, and the development of sophisticated experimental analysis approaches. fMRI measures changes in the concentration of oxygenated blood in different locations in the brain, denoted as the BOLD (Blood Oxygenation Level Dependent) signal [Huettel04]. The cellular processes occurring when neural impulses are transmitted between nerve cells require energy derived from reactions where oxygen participates as a metabolite, thus the delivery of oxygen to particular locations in the brain follows neural activity in that location. This fact is used to infer neural activity in a region of the brain from measurements of the BOLD signal therein. In a typical fMRI experiment this measurement is repeated many times, providing a spatio-temporal profile of the BOLD signal inside the subject's brain. The minimal spatial unit of measurement is a volumetric pixel, or "voxel", typically of the order of a few mm^3.

The temporal profile of the acquired BOLD signal is limited by the fact that blood flow into neurally active tissue is a much slower process than the changes in the activity of neurons. From a signal processing perspective, this measured profile can be seen as the convolution of a rapid oscillation at rates of $\mathcal{O}(10 - 1000)$Hz (neuronal activity) with a much slower function that varies at rates of ~ 0.15Hz (changes in blood flow due to neighboring blood vessels dilating and contracting). This slowly varying blood flow profile is known as the *hemodynamic response function* (HRF), and it acts as a low-pass filter on the underlying brain activity [Aguirre97].

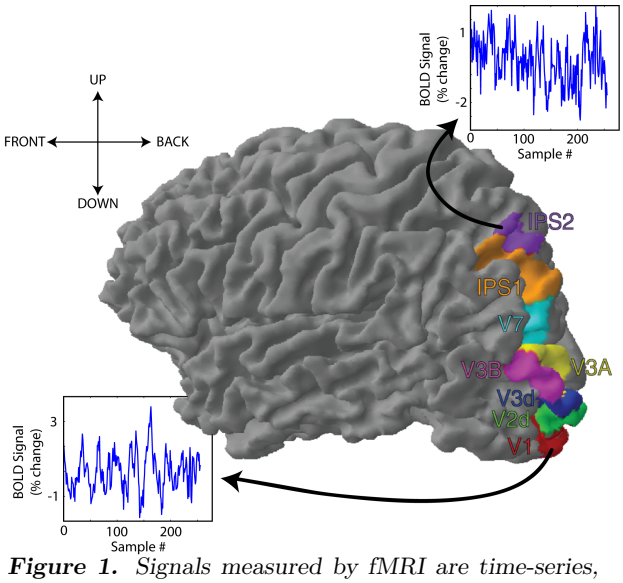

Figure 1. *Signals measured by fMRI are time-series, illustrated from areas in the human visual cortex (adapted from Silver et al., 2005; with permission).*

fMRI data analysis and time-series analysis

The interpretation of fMRI data usually focuses on the analysis of the relationship between the BOLD time-series in each voxel and the occurrence of events of behavioral significance, such as the presentation of a stimulus to the subject or the production of a motor response by the subject.

Figure 1 presents a rendered image of the brain, presented with the anterior part of the brain turning leftwards. Visual areas of the cerebral cortex are located in the posterior part of the brain. The colored patches represent different functional regions in the visual cortex: these areas labeled V1-V7 respond to visual stimulation. Each contains a representation of the visual field such that adjacent locations on the brain surface respond to stimuli appearing in adjacent locations in the visual field. The areas IPS1 and IPS2 (named after their anatomical location in the intraparietal sulcus) contain an ordered representation of the visual field, but respond to the allocation of attention instead of direct visual stimulation [Silver05]. By averaging the measured BOLD signal over all the voxels in each area, we obtain time-series representative of the activity in the region; this is illustrated in the insets for V1 and IPS2.

Typically, univariate analyses calculate a statistical measure of the correlation between the actual activity in each voxel and the activity expected, if the voxel contains neurons which respond to the behaviorally relevant events. Clusters of voxels that significantly correlate with this model are then considered to contain neurons participating in the cognitive function associated with the behavior in question. Sometimes, this approach is sufficient in order to answer interesting questions about functional specialization of various parts of the brain. However, in most cases it is beneficial to further study the time-series measured in the fMRI with approaches that go beyond univariate analysis.

One important aspect in further analysis of the time-series is the definition of regions of interest (ROI) and the targeting of the analysis to those regions [Poldrack06]. ROIs are defined based on criteria such as the known functional specialization of a region in the brain or the anatomical location of that region. In an ROI-based analysis, the time-series from all the voxels in this ROI are extracted and averaged and the average time-series is then subject to further analysis. This approach is readily adopted in areas of the brain where functionally distinct areas can be spatially defined. For example, the areas of the cortex which participate in processing of visual information (often referred to as "visual areas" and denoted by V1, for "primary visual cortex", V2, for "secondary visual cortex", etc.) each contain an ordered representation of the entire visual field (Figure 1) [Wandell07]. These neighboring regions can thus be spatially separated based on the layout of their respective visual field "maps" relative to each other.

One extension of the univariate approach mentioned above, is to examine the functional relations between different time-series, extracted from different locations in the brain. One of the major advantages of fMRI is that measurements are performed simultaneously from many different regions of the brain. Therefore, this method allows us to define not only the correspondence between the time-series of BOLD in one particular region and the events that occurred in the environment while this data was collected, but also the correspondence between time-series collected from one region and the time-series simultaneously collected in another region of the brain. This approach is often referred to as "functional connectivity analysis" [Friston94], where bivariate and multivariate measures of covariance between two or more time-series, taken from different areas in the brain, are calculated. This kind of analysis allows us to infer not only about the participation of a particular brain region in a cognitive process of interest, but also about the functional network of regions and the interplay between activity in these different regions. These analysis techniques can be done using an ROI based approach (see example, below). However, they can also be utilized as exploratory data analysis techniques, in which the connectivity between every pair of voxels in the brain is calculated and the voxels are clustered into functional modules according to their connectivity.

`Nitime`

The `nitime` package tries to provide scientists conducting brain-imaging research with a clean and easy-to-use interface to algorithms that calculate quantities derived from the time-series acquired in fMRI experiments. It contains implementations both of methods previously used in neuroimaging and of time-series analysis techniques that have yet to be widely used in neuroimaging. We also aim to develop new and original ways of analyzing fMRI data and to make these methods available to the community of scientists conducting brain-imaging research through `nitime`. The algorithms are built to allow calculation of some univariate quantities pertaining to the time-series in question, as well as bivariate and multivariate quantities. They are meant to allow an ROI based approach, as well as analyses done at the single-voxel level. Many of the algorithms could be used in the analysis of other kinds of time-series, whether taken from other modalities of neuroscientific research or even in other fields of science. Therefore, we have decoupled the parts of the implementation that are directly related to neuroimaging data (functions for reading data from standard neuroimaging formats, for example), from our object-oriented design and from the implementation of algorithms.

`Nitime` fits within the broader context of other related Python projects: The `TimeSeries` SciPy scikit [TimeSeries] focuses on the analysis of calendar-based time-series (such as those used in finance), while the

design of `nitime` is more directly geared towards experimental data analysis. BrainVISA [Favre09], is a package that focuses on fMRI data and provides a complete array of analysis facilities, including preprocessing, visualization and batch workflows. In contrast, `nitime` is a developer-oriented library that focuses on implementation of algorithms and generic data-management objects. `Pyhrf` [Makni08] is a library which implements a joint detection-estimation approach to brain activity. Future development will hopefully lead to tighter integration of `nitime` with this library. Finally, `Nitime` time-series objects can use the newly introduced `datetime` data type in NumPy but do not depend on it, and can thus be used to manipulate any data set that fits into an n-dimensional NumPy array.

Importantly, analysis of fMRI data requires several steps of pre-processing, such as motion correction and file-format conversion to occur before this kind of analysis can proceed. Several software packages, such as BrainVISA, FSL [Smith04] , AFNI [Cox97] and SPM [Friston95] implement algorithms which can be used in order to perform these steps. Furthermore, the `nipype` library (`http://nipy.sourceforge.net/nipype/`), which is also part of the NiPy project, provides a Python programming interface to some of these packages. The design of `nitime` assumes that the data has been pre-processed and can be read in either as a NumPy `ndarray` or in the standard NIfTI file-format.

Next, we will describe the design of `nitime` and the decision-making process leading to this implementation. Then, we will demonstrate how `nitime` can be used to analyze real-world data.

Software design

Today, most high-level software uses object oriented (OO) design ideas to some extent. The Python language offers a full complement of syntactic support to implement OO constructs, from simple one-line classes to complex hierarchies. Previous experience has shown us that designing good OO interfaces is far, far harder than it appears at first. Most of us who have come to write software as part of our scientific work but without much formal training in the matter, often prove surprisingly poor performers at this task.

The simplest description of what an object is in computer science, presents it as the coupling of data and functions that operate on said data. The problem we have seen in the past with a literal interpretation of this description, however, is that it is very easy to build object hierarchies where the data and the algorithms are more tightly coupled than necessary, with numerical implementation details living inside the methods of the resulting objects, and the objects holding too much state that is freely reused by all methods. This effectively buries the algorithms inside of objects and makes it difficult to reuse them in a different design without carrying the containing objects.

To a good extent, this is the problem that the C++ Standard Template Library tries to address by separating containers from algorithms and establishing interfaces for generically coupling both at use time. For `nitime`, we have tried to follow this spirit by separating our implementation into three distinct parts:

1. A purely functional library, `nitime.algorithms`, that implements only the numerical part of time-series analysis algorithms. These functions manipulate NumPy arrays and standard Python types (integers, floats, etc.), which makes their calling signatures necessarily somewhat long, in the classical style of well known libraries such as LAPACK.

2. A set of "dumb" data container objects for time-series data, that do as little as possible. By forcing them to have a very minimal interface, we hope to reduce the chances of making significant design mistakes to a minimum, or of creating objects whose interface 'over-fits' to our needs and is thus not useful for others. In this respect, we try to follow the excellent example laid out by Python itself, whose core objects have small but remarkably general and useful interfaces.

3. A set of "smart" objects, which we have called *analyzers*, that provide access to the algorithms library via easy to use, high-level interfaces. These objects also assist in bookkeeping of state, and possibly caching information that may be needed in different computations (such as the Fourier Transform of a signal).

Our analyzer objects are a lightweight binding of #1 and #2 above, and thus represent a minimal investment of effort and code. If they prove to be a poor fit for a new user of `nitime` (or for us in the future), there is no significant loss of functionality and no major investment of time has been made for naught. The real value of the library lies in its algorithms and containers, and users should be free to adapt those to the task at hand in any way they see fit. We now provide a brief overview of these three components, whose use we will illustrate in the next section with a more detailed example.

Algorithms

The `nitime.algorithms` module currently implements the following algorithms:

Spectral transforms

Transforms of time-series data to the frequency-domain underlie many methods of time-series analysis. We expose previous implementations of spectral transforms taken from `mlab`, Matplotlib's library of numerical algorithms [Matplotlib]. In addition, we have written algorithms for the calculation of a standard periodogram and cross-spectral density estimates

©2009, A. Rokem, M. Trumpis, F. Pérez

based both on regular and multi-taper periodograms. The multi-taper periodogram was implemented here using discrete prolate spheroidal (Slepian) sequences ([NR07], [Percival93], [Slepian78]).

Coherency

Coherency is an analog of cross-correlation between two time-series calculated in the frequency domain, which can be used to study signals whose underlying spectral structure is similar despite containing substantial differences in the time domain. In fMRI analysis, this technique is used in order to calculate the functional connectivity between time-series derived from different voxels in the brain, or different ROIs and in order to infer the temporal relations them [Sun05]. One of the inherent problems in the analysis of functional connectivity of fMRI signals is that the temporal characteristics of the hemodynamic response in different areas of the brain may differ due to variations in the structure of the local vasculature in the different regions. Consequently, the delay between neural activity in a voxel and the peak of the ensuing influx of oxygenated blood may differ quite significantly between different voxels, even if the neural activity which is the root cause of the BOLD response and the quantity of interest, were identical. Thus, the correlation between the two time-series derived from the BOLD response in two different regions may be quite low, only because the hemodynamic response in one area begins much later than the hemodynamic response in the other area.

This limitation can be overcome to some degree, by conducting the analysis in the frequency domain, instead of the time domain. One type of analysis technique which examines the correspondence between two or more time-series in the frequency domain is coherency analysis. *Coherency* is defined as:

$$Coherency_{xy}(\nu) = \frac{f_{xy}(\nu)}{\sqrt{f_{xx}(\nu) f_{yy}(\nu)}}, \qquad (1)$$

where $f_{xy}(\nu)$ is the cross-spectral density between time-series x and time-series y in the frequency band centered on the frequency ν; $f_{xx}(\nu)$ and $f_{yy}(\nu)$ are the frequency-dependent power-spectral densities of time-series x and y respectively.

The squared magnitude of the coherency, known as *coherence*, is a measure of the strength of the functional coupling between x and y. It varies between 0 and 1 and will be high for two time-series which are functionally coupled even if the delays in their respective hemodynamic responses differ substantially. The phase $\phi(\nu)$ of the coherency can relay the temporal delay between the two time-series, via the relation $\Delta t(\nu) = \phi(\nu)/(2\pi\nu)$.

Importantly, the temporal resolution at which the delay can be faithfully reproduced in this method does not depend on the sampling rate (which is rather slow in fMRI), but rather depends on the reliability with which the hemodynamic response is produced given a particular activity. Though the hemodynamic response may vary between different subjects and between different areas in the same subject, it is rather reproducible for a given area in a given subject [Aguirre98].

In our implementation, these quantities can be computed with various methods to extract the cross-spectral density f_{xy} and the spectral densities f_{xx} and f_{yy}.

Regularized coherency

In addition to the standard algorithm for computing coherency, we have implemented a regularized version, which permits robust computation of coherence in the presence of small denominators (that occur for frequency bands where f_{xx} or f_{yy} is very small). Omitting the frequency ν for notational brevity, we replace eq. (1) with:

$$Coh^{R}_{xy,\alpha\epsilon} = \frac{|\alpha f_{xy} + \epsilon|^2}{\alpha^2 (f_{xx} + \epsilon)(f_{yy} + \epsilon)}, \qquad (2)$$

where α and ϵ are real numbers. This expression tends to Coh_{xy} when $\epsilon \to 0$, but is less sensitive to numerical error if either f_{xx} or f_{yy} is very small. Specifically, if $|f| \gg \epsilon$ then $Coh^{R}_{xy,\alpha\epsilon} \to Coh_{xy}$ (where f is any of f_{xx}, f_{yy} or f_{xy}), and if $f \approx \epsilon$ then:

$$Coh^{R}_{xy,\alpha\epsilon} \to \frac{(\alpha+1)^2}{4\alpha^2} Coh_{xy} \approx \frac{1}{4} Coh_{xy} \qquad (3)$$

for $\alpha \gg 1$. We note that this is only an order-of-magnitude estimate, not an exact limit, as it requires replacing ϵ by f_{xx}, f_{yy} and f_{xy} in different parts of eq. (1) to factor out the Coh_{xy} term.

For the case where $|f| \ll \epsilon \ll \alpha$, $Coh^{R}_{xy,\alpha\epsilon} \to \frac{1}{\alpha^2}$. In this regime, which is where small denominators can dominate the normal formula returning spurious large coherence values, this approach suppresses them with a smooth decay (quadratic in α).

Event-related analysis

A set of algorithms for the calculation of the correspondence between fMRI time-series and experimental events is available in `nitime`. These are univariate statistics calculated separately for the time-series in each voxel or each ROI. We have implemented a standard least squares estimate of the hemodynamic response function in response to a series of different events [Dale00].

In addition, we have implemented a fast algorithm for calculation of the cross-correlation between a series of events and a time-series and comparison of the resulting event-related response functions to the baseline variance of the time-series.

Containers

A `TimeSeries` object is a container for an arbitrary n-dimensional array of data (a NumPy `ndarray` object), along with a single one-dimensional array of time points. In the data array, the first $n-1$ dimensions are considered to describe the data elements (if $n = 1$, the elements are simply scalars) and the last dimension is the time axis. Since the native storage order of NumPy arrays is C-style (last index varies fastest), our choice gives greater data locality for algorithms that require taking elements of the data array and iterating over their time index. For example, a set of recordings from a multichannel sensor can be stored as a 2-d array A, with the first index selecting the channel and the second selecting the time point. In C-order storage, the data for channel i, $A[i]$ will be contiguous in memory and operations like an FFT on it will benefit from cache locality.

The signature of the `UniformTimeSeries` constructor is:

```
def __init__(self, data, t0=None,
             sampling_interval=None,
             sampling_rate=None,
             time=None, time_unit='s')
```

Any attribute not given at initialization time is computed at run time from the others (the constructor checks to ensure that sufficient information is provided, and raises an error otherwise). The standard Python approach for such problems is to use properties, but properties suffer from the problem that they involve a call to a getter function on every access, as well as requiring explicit cache management to be done in the getter. Instead, we take advantage of the dynamic nature of Python to find a balance of property-like delayed evaluation with attribute-like static storage.

We have defined a class called `OneTimeProperty` that exposes the descriptor protocol and acts like a property but, on first access, computes a value and then sets it statically as an instance attribute. The function is then called only once, and any further access to the name requires only a normal, static attribute lookup with no overhead. The code that implements this idea, stripped of comments and docstrings for the sake of brevity but otherwise complete, is:

```
class OneTimeProperty(object):
   def __init__(self,func):
       self.getter = func
       self.name = func.func_name

   def __get__(self,obj,type=None):
       if obj is None:
           return self.getter
       val = self.getter(obj)
       setattr(obj, self.name, val)
       return val
```

When writing a class such as `UniformTimeSeries`, one then declares any property whose first computation should be done via a function call using this class as a decorator. As long as no circular dependencies are introduced in the call chain, multiple such properties can depend on one another. This provides for an implicit and transparent caching mechanism. Only those attributes accessed, either by user code or by the computation of other attributes, will be computed. We illustrate this with the implementation of the `time`, `t0`, `sampling_interval` and `sampling_rate` attributes of the `UniformTimeSeries` class:

```
@OneTimeProperty
def time(self):
    npts = self.data.shape[-1]
    t0 = self.t0
    t1 = t0+(npts-1)*self.sampling_interval
    return np.linspace(t0,t1,npts)

@OneTimeProperty
def t0(self):
    return self.time[0]

@OneTimeProperty
def sampling_interval(self):
    return self.time[1]-self.time[0]

@OneTimeProperty
def sampling_rate(self):
    return 1.0/self.sampling_interval
```

We have found that this approach leads to very readable code, that lets us delay computation where desired without introducing layers of management code (caching, private variables for getters, etc.) that obscure the main intent.

We have so far overlooked one important point in our discussion of "automatic attributes": the case where the quantities depend on mutable data, so that their previously computed values become invalid. This is a problem that all caching mechanisms need to address, and in its full generality it requires complex machinery for cache state control. Since we rely on an implicit caching mechanism and our properties become regular attributes once computed, we can not use regular cache dirtying procedures. Instead, we have provided a `ResetMixin` class that can be used for objects whose automatic attributes may become invalid. This class provides only one method, `reset()`, that resets *all* attributes that have been computed back to their initial, unevaluated state. The next time any of them is requested, its accessor function will fire again.

Analyzers

We now describe our approach to exposing a high-level interface to the analysis functions. We have constructed a set of lightweight objects called *analyzers*, that group together a set of conceptually related analysis algorithms and apply them to a specific time-series object. These objects have a compact implementation and no significant numerical code; their purpose is to do simple book-keeping and to allow for end-user code that is readable and compact. Their very simplicity also means that they shouldn't be judged too severely if they don't fit a particular application's needs: it is easy enough to implement new analysis objects as needed. We do hope that the ones provided by `nitime` will serve many common cases, and will also be useful

©2009, A. Rokem, M. Trumpis, F. Pérez

reference implementations for cases that require writing new ones.

All analyzers follow a similar pattern: they are instantiated with a `TimeSeries` object of interest, which they keep an internal reference to, and they expose a series of attributes and methods that compute specific algorithms from the library for this time series. For all the main quantities of interest that have a static meaning, the analyzer exposes an attribute accessor that, via the `OneTimeProperty` class, calls the underlying algorithm with all required parameters and stores the result in the attribute for further use. In addition to these automatic attributes, analyzers may also expose normal methods that provide simplified interfaces (with less parameters) to algorithms. If any of these methods requires one of the automatic attributes, it will be naturally computed by the accessor on first access and this result will be then stored in the instance. We will now present examples showing how to analyze both synthetic and real fMRI data with these objects.

Examples: coherency analysis

Analysis of synthetic time-series

The first example we present is a simple analysis stream on a pair of synthetic time-series (Figure 2), of the form

$$x(t) = \sin(\alpha t) + \sin(\beta t) + \epsilon_x \quad (4)$$

$$y(t) = \sin(\alpha t + \phi_1) + \sin(\beta t - \phi_2) + \epsilon_y \quad (5)$$

where $\epsilon_{x,y}$ are random Gaussian noise terms and $\phi_i > 0$ for $i = 1, 2$, such that each is a superposition of two sinusoidal functions with two different frequencies and some additional uncorrelated Gaussian white noise and the relative phases between the time-series have been set such that in one frequency, one series leads the other and in the other frequency the relationship is reversed.

We sample these time series into an array `data` from which a `UniformTimeSeries` object is initialized:

```
In [3]: TS = UniformTimeSeries(data,sampling_rate=1)
```

A correlation analyzer object is initialized, using the time-series object as input:

```
In [4]: Corr = CorrelationAnalyzer(TS)
```

`Corr.correlation` now contains the full correlation matrix, we extract the position `[0,1]`, which is the correlation coefficient between the first and the second series in the object:

```
In [5]: Corr.correlation[0,1]
Out[5]: 0.28727768
```

The correlation is rather low, but there is a strong coherence between the time-series (Figure 2B) and in particular in the two common frequencies. We see this by initializing a coherence analyzer with the time-series object as input:

```
In [6]: Coh = CoherenceAnalyzer(TS)
```

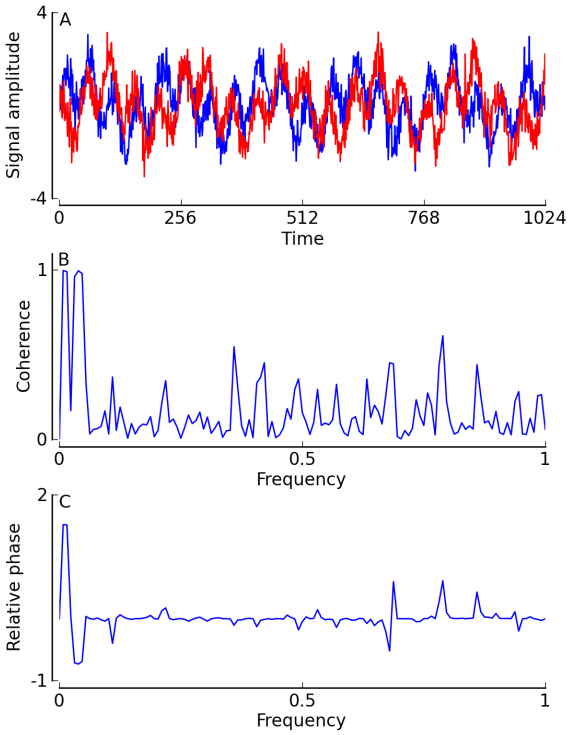

Figure 2. Coherency analysis - an example: ***A****: two synthetic time-series are displayed.* ***B****: The coherence is displayed as a function of frequency.* ***C****: The coherency phase-delay between the time-series is presented, as a function of frequency.*

We examine specifically the coherence in the frequency bands of interest with indices 2 and 6:

```
In [7]: Coh.coherence[0,1,2]
Out[7]: 0.9893900459215027

In [8]: Coh.coherence[0,1,6]
Out[8]: 0.97800470864819844
```

These two high coherence values are what gives rise to the two prominent peaks in the coherence in Figure 2B. In addition, the relative phases are reversed in the two frequencies composing these time series. This is reflected in the relative phase between the time-series (Figure 2C), which can be calculated in a similar way.

Analysis of fMRI data

Our second example (Figure 3) demonstrates the analysis of actual experimental fMRI data, acquired by David Bressler and Michael Silver. In this experiment, subjects fixated their gaze on a dot at the center of the visual field and viewed a wedge-shaped section of a circle (like a pizza slice), which slowly rotated around the fixation dot at a rate of one full cycle every 32 seconds. The wedge contained a flickering checker-board

pattern. This pattern of stimulation is known to stimulate activity in visual areas, which can be measured with fMRI. In half of the scans, the subject was instructed to detect the appearance of targets inside the checker-board pattern. In the other half of the scans, the subject was instructed to detect targets appearing in the fixation point at the center of gaze. Thus, in both cases, the subject's attention was engaged in a difficult detection task (tasks were adjusted so that the difficulty in both cases was similar). The only difference between the two conditions was whether attention was directed into the area covered by the checker-board wedge or out of this area. Previous research [Lauritzen09] has shown that allocation of attention tends to increase coherence between areas in early visual cortex and between areas in visual cortex and IPS areas (see Figure 1). Data was recorded from subjects' brains in the scanner, while they were performing this task. Visual ROIs were defined for each subject. These ROIs contain the parts of cortex which represent the areas of the visual field through which the checker-board wedge passes in its rotation, but not the area of the visual field close to the center of the focus of the gaze, the area in which the fixation point is presented. Thus, the activity measured in the experiment is in response to the same visual stimulus of the checker-board wedge; in half the scans, while attention is directed to the wedge and in the other half, when attention is directed away from the wedge.

In order to examine the functional connectivity between the ROIs, we start from data stored on disk in a `.npy` file containing an array with time-series objects, created from the raw fMRI data for a single subject:

```
In [1]: tseries_arr = np.load('tseries.npy')
```

Each `TimeSeries` object in this array corresponds to the data for a separate scan, and it contains the mean BOLD data for 7 separate ROIs (one per visual area of interest, see Figure 3). Attention was directed to the fixation point in the even scans and to the checker-board wedge in the odd scans. We initialize coherence analyzers for each of the scans and store those in which attention was directed to the wedge separately from those in which attention was directed to the fixation point:

```
In [2]: C_fix = map(CoherenceAnalyzer,
  ....:             tseries_arr[0::2])  # even scans

In [3]: C_wedge = map(CoherenceAnalyzer,
  ....:              tseries_arr[1::2])  # odd scans
```

We extract the cross-coherence matrix for all the ROIs in one frequency band (indexed by 1) and average over the scans:

```
In [4]: mean_coh_wedge = array([C.coherence[:,:,1]
  ....:   for C in C_wedge]).mean(0)

In [5]: mean_coh_fix = array([C.coherence[:,:,1]
  ....:   for C in C_fix]).mean(0)
```

In order to characterize the increase in coherence with attention to the wedge, we take the difference of the resulting array:

```
In [6]: diff = mean_coh_wedge - mean_coh_fix
```

In Figure 3, we have constructed a graph (using NetworkX [NetworkX]) in which the nodes are the visual area ROIs, presented in Figure 1. The edges between the nodes represent the *increase* in coherence in this frequency band, when subjects are attending to the wedge, relative to when they are attending to the appearance of targets in the fixation point. This graph replicates previous findings [Lauritzen09]: an increase in functional connectivity between visual areas, with the allocation of voluntary visual attention to the stimulus.

These examples demonstrate the relative simplicity and brevity with which interactive data analysis can be conducted using the interface provided by `nitime`, resulting in potentially meaningful conclusions about the nature of the process which produced the time-series. This simplicity should facilitate the study of complex data sets and should enable more sophisticated methods to be developed and implemented.

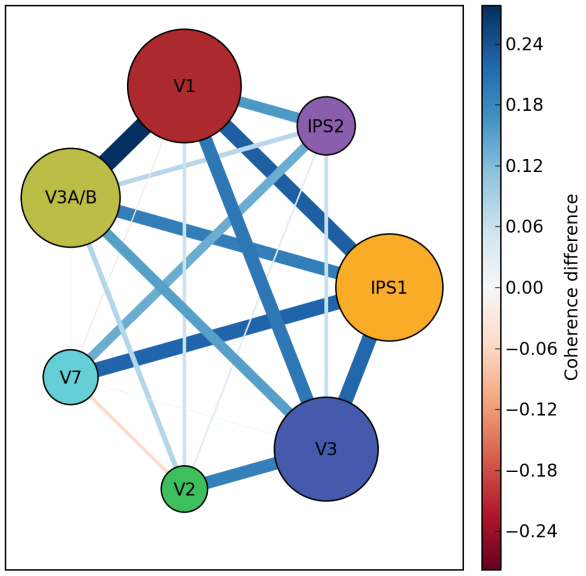

Figure 3. Functional connectivity in the visual cortex: *In the graph presented, the nodes represent the areas of the brain described in Figure 1 (node colors and labels match those of Figure 1).*

Summary and outlook

We have introduced `nitime`, a library for the analysis of time-series data from neuroimaging experiments and in particular from fMRI experiments, developed as part of the NiPy project. `Nitime` provides implementations of several algorithms, including coherency analysis, and a high-level interface for interaction with time-series data. Its design emphasizes a decoupling of algorithmic implementation and object-oriented features. This is meant to facilitate use of the algorithms in contexts other than neuroscience and contributions

©2009, A. Rokem, M. Trumpis, F. Pérez

from developers in other fields. Future developments will include implementations of additional algorithms for calculation of bivariate and univariate quantities, as well as tools for visualization of time-series.

Acknowledgments

We would like to thank the NiPy development team, in particular Matthew Brett for many helpful discussions, Gaël Varoquaux, for discussions on properties that led to `OneTimeProperty` and Dav Clark for feedback on the time-series object interface. We are thankful to David Bressler and Michael Silver for providing us with experimental data, and to all the members of Michael Silver's lab for comments and feedback on this work. Felice Sun, Thomas Lauritzen, Emi Nomura, Caterina Gratton, Andrew Kayser, Ayelet Landau and Lavi Secundo contributed Matlab code that served as a reference for our implementation of several algorithms. Finally, we'd like to thank Mark D'Esposito, director of the NiPy project, for supporting the development of nitime as part of the overall effort. NiPy is funded by the NIH under grant #5R01MH081909-02.

References

[Aguirre97] Aguirre GK, Zarahn E, D'Esposito M (1997). *Empirical Analyses of BOLD fMRI Statistics II. Spatially Smoothed Data Collected under Null-Hypothesis and Experimental Conditions.* Neuroimage **5**: 199-212.

[Aguirre98] Aguirre GK, Zarahn E, D'esposito M (1998). *The variability of human, BOLD hemodynamic responses.* Neuroimage **8**: 360-9.

[BSD] *The BSD Software License.* http://www.opensource.org/licenses/bsd-license.php.

[Cox97] Cox RW and Hyde JS (1997). *Software tools for analysis and visualization of FMRI data.* NMR in Biomedicine, **10**: 171-178.

[Dale00] Dale, AM (2000). *Optimal Experimental Design for Event-Related fMRI.* Human Brain Mapping **8**: 109-114.

[Favre09] Favre L, Fouque A-L, et al. (2009) *A Comprehensive fMRI Processing Toolbox for BrainVISA.* Human Brain Mapping **47**: S55.

[Friston94] Friston, KJ (1994). *Functional and Effective Connectivity in Neuroimaging: A Synthesis.* Human Brain Mapping **2**: 56-78.

[Friston95] Friston KJ, Ashburner J, et al. (1995) *Spatial registration and normalization of images.* Human Brain Mapping, **2**: 165-189.

[Huettel04] Huettel, SA, Song, AW, McCarthy, G (2004). *Functional Magnetic Resonance Imaging.* Sinauer (Sunderland, MA).

[Kayser09] Kayser AS, Sun FT, D'Esposito M (2009). *A comparison of Granger causality and coherency in fMRI-based analysis of the motor system.* Human Brain Mapping, *in press.*

[Lauritzen09] Lauritzen TZ , D'Esposito M, et al. (2009) *Top-down Flow of Visual Spatial Attention Signals from Parietal to Occipital Cortex.* Journal of Vision, *in press.*

[Makni08] Makni S, Idier J, et al. (2008). *A fully Bayesian approach to the parcel-based detection-estimation of brain activity in fMRI.* Neuroimage **41**: 941-969.

[Matplotlib] Hunter, JD (2007). *Matplotlib: A 2D graphics environment.* Comp. Sci. Eng. **9**: 90-95.

[NetworkX] Hagberg AA, Schult DA, Swart PJ (2008). *Exploring network structure, dynamics, and function using NetworkX.* in Proc. 7th SciPy Conf., Varoquaux G, Vaught T, and Millman J (Eds), pp. 11–15.

[NiPy] Millman KJ, Brett M (2007). *Analysis of functional Magnetic Resonance Imaging in Python.* Comp. Sci. Eng. **9**: 52-55.

[NR07] Press, WH, Teukolsky, SA, Vetterling, WT, Flannery, BP (2007). *Numerical Recipes: The Art of Scientific Computing.* 3rd edition, Cambridge University Press (Cambridge, UK).

[Percival93] Percival DB and Walden, AT (1993). *Spectral Analysis for Physical Applications: Multitaper and Conventional Univariate Techniques.* Cambridge: Cambridge University Press.

[Poldrack06] Poldrack, R (2006). *Region of interest analysis for fMRI.* Soc. Cog. Aff. Neurosci., **2**: 67-70.

[Silver05] Silver MA, Ress D ,Heeger DJ (2005). *Topographic maps of visual spatial attention in human parietal cortex.* J Neurophysiol, **94**: 1358-71.

[Slepian78] Slepian, D (1978). *Prolate spheroidal wave functions, Fourier analysis, and uncertainty V: The discrete case.* Bell System Technical Journal, **57**: 1371-1430.

[Smith04] Smith SM, Jenkinson M, et al. (2004). *Advances in functional and structural MR image analysis and implementation as FSL.* NeuroImage, **23**: 208-219.

[Sun05] Sun FT, Miller LM, D'Esposito M (2005). *Measuring interregional functional connectivity using coherence and partial coherence analyses of fMRI data.* Neuroimage **21**: 647-658.

[TimeSeries] Gerard-Marchant P, Knox M (2008). *Scikits.TimeSeries: Python time series analysis.* http://pytseries.sourceforge.net.

[Wandell07] Wandell BA, Dumoulin, SO, Brewer, AA (2007). *Visual field maps in human cortex.* Neuron **56**: 366-83.