Neural Networks 21 (2008) 1085–1093

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

2008 Special Issue

FIND — A unified framework for neural data analysis

Ralph Meier [a,*], Ulrich Egert [a,b], Ad Aertsen [a], Martin P. Nawrot [c,d,a]

[a] Bernstein Center for Computational Neuroscience, Albert-Ludwigs-University, Freiburg, Germany
[b] Department of Microsystems Engineering, Faculty of Applied Sciences, Albert-Ludwigs-University, Freiburg, Germany
[c] Neuroinformatics and Theoretical Neuroscience, Institute of Biology – Neurobiology, Freie Universität Berlin, Germany
[d] Bernstein Center for Computational Neuroscience, Berlin, Germany

ARTICLE INFO

Article history:
Received 2 November 2007
Received in revised form
7 June 2008
Accepted 17 June 2008

Keywords:
Coefficient of variation
Gamma process
Point process simulation
Spike train analysis
Neural activity data analysis
Open source toolbox
Mushroom body
Visual cortex
Rat
Honey bee

ABSTRACT

The complexity of neurophysiology data has increased tremendously over the last years, especially due to the widespread availability of multi-channel recording techniques. With adequate computing power the current limit for computational neuroscience is the effort and time it takes for scientists to translate their ideas into working code. Advanced analysis methods are complex and often lack reproducibility on the basis of published descriptions. To overcome this limitation we develop FIND (Finding Information in Neural Data) as a platform-independent, open source framework for the analysis of neuronal activity data based on MATLAB (Mathworks). Here, we outline the structure of the FIND framework and describe its functionality, our measures of quality control, and the policies for developers and users. Within FIND we have developed a unified data import from various proprietary formats, simplifying standardized interfacing with tools for analysis and simulation. The toolbox FIND covers a steadily increasing number of tools. These analysis tools address various types of neural activity data, including discrete series of spike events, continuous time series and imaging data. Additionally, the toolbox provides solutions for the simulation of parallel stochastic point processes to model multi-channel spiking activity. We illustrate two examples of complex analyses with FIND tools: First, we present a time-resolved characterization of the spiking irregularity in an *in vivo* extracellular recording from a mushroom-body extrinsic neuron in the honeybee during odor stimulation. Second, we describe layer specific input dynamics in the rat primary visual cortex *in vivo* in response to visual flash stimulation on the basis of multi-channel spiking activity.

1. Motivation

In parallel to the tremendous technical progress in data acquisition (e.g. large number of simultaneous electrode recordings), there is a growing need for new computational tools to analyze and interpret the resulting large data flow from experiments and simulations. While there is undeniable progress in novel analysis methods, implementations are difficult to reproduce based on print publications or they are hidden in 'in-house' software collections which are often ill-documented and thus hardly accessible.

Since computing power is nowadays affordable, the current limit for computational analysis is how quickly and reliably scientists can translate their ideas into working code (Wilson, 2006). Moreover, neuroscientists are usually not trained as software engineers and have to cope with problems in the increasingly complex design of analysis software (Baxter, Day, Fetrow, & Reisinger, 2006).

As a contribution to overcome these problems, we are developing the open source platform-independent FIND toolbox to address the urgent need of a unified, well-documented framework, interfacing standard electrophysiology data sources and to collate various analysis and simulation tools. FIND stands for *Finding Information in Neural Data*, builds on the Neuroshare interface definition and is shared with the neuroscience community. To enable the incorporation of new algorithms, FIND provides the possibility to extend the collection of algorithms and data formats with new ones by supplying templates to integrate existing code into the toolbox. The open source idea allows concise review of methods and algorithms, and opens possibilities for enhancements. We expect that this will facilitate the development and distribution of new data analysis techniques among the scientific community.

2. Concept

FIND implements a unified import of multiple proprietary data formats, based on the Neuroshare Project (http://neuroshare.

* Corresponding author. Tel.: +49 761 203 2786; fax: +49 761 203 2860.
E-mail addresses: meier@biologie.uni-freiburg.de (R. Meier), aertsen@biologie.uni-freiburg.de (A. Aertsen).

0893-6080/$ – see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2008.06.019

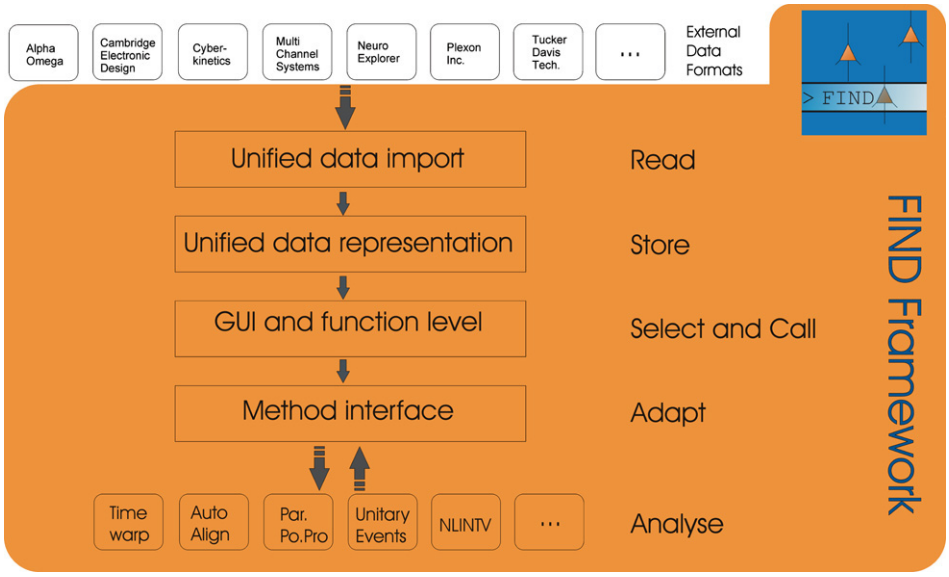

Fig. 1. Schematic overview of the FIND framework. Currently, data acquisition with different products results in files with numerous proprietary data formats for neural data (top row). The FIND framework provides unified data import to read these and various open data formats (**Read**). The imported data is represented in a unified manner within the Matlab programming environment (**Store**). On this level, FIND supports browsing the existing data set and the selection of a subset of data for further processing. Then analysis functions can be called using a GUI and/or directly from the command line (**Select and Call**). Our approach includes full interchangeability between GUI calls and programmed scripts using the function calls. All calls generated are passed through a unified interface (**Adapt**) to call the actual analysis-functions that process the data. On the next level (**Analyse**), we provide high-level analysis functions. Interaction between several high-level analyses and cascading of data processing steps is supported by the unified data representation. For all steps, we offer template functions and documentation to facilitate and encourage the addition of further methods to the FIND framework. For a full list of functions currently available, see the project website http://find.bccn.uni-freiburg.de.

sourceforge.net), but also includes import routines for data formats not covered by the Neuroshare interface. Physiological data from different acquisition systems: Alpha Omega, Cambridge Electronic Design, Multi Channel Systems, NeuroExplorer, Plexon, Tucker-Davis Technologies, Cyberkinetics and others as well as data from network simulations environments (e.g. NEST: www.nest-initiative.org, Morrison, Mehring, Geisel, Aertsen, and Diesmann (2005); and PyNN: http://neuralensemble.org/trac/PyNN) are now accessible through one interface and can be analyzed using identical methods. This allows the verification of results across experiments and laboratories and enables the direct comparison of simulation results with electrophysiological measurements.

We are hosting a professional software development platform including a software repository (using the Trac Project, http://trac.edgewall.org/) to allow developers of algorithms used in neuroscience to add/incorporate their tools within the FIND framework. To facilitate this process we developed templates for functions and templates for graphical user interfaces (GUIs), and specified application interfaces (API) for data storage and function calls. A review board controls the addition of software (i.e. in most cases analysis tools) to ensure quality of the code and plausibility of the tools provided. In addition, we evaluate and propagate the functionality of the FIND toolbox in parts of our teaching and training program in computational neuroscience at the German Bernstein Centers for Computational Neuroscience. In the future, we will provide curricula and training material based on FIND. Full information on functionality, software updates and access to recent releases, a list of contributors and partners, and our invitation to join the team of developers can be found on the FIND website http://find.bccn.uni-freiburg.de.

3. Design and community interaction

In this section we briefly outline the design structure of the FIND framework (Fig. 1). A central element is the unified data import and representation of many different data formats. This makes the analysis tools independent of the hard- and software used for data acquisition. In addition, the storage in a unified format readily offers a means for the standardized exchange of neural activity data. The tools collected in the FIND toolbox interface to the data in a standardized fashion. Individual tools are represented by individual Matlab functions with a standardized input and output format. On top of the basic functions, a graphical user interface (GUI) provides easy access to data and tools, also for the programming novice. Quality management is introduced by a technical review process of contributed tools before public release. Participation in the FIND community should result in a win–win situation for both, developers and users. We outline the FIND policy which finds its expression in guidelines for developers and for users to serve their mutual interests.

3.1. Data import and internal data representation

In a first step, neural activity data and meta-information can be imported from many proprietary (via the Neuroshare Interface) and open (e.g. MEA-Bench format http://www.danielwagenaar.net/meabench/) data formats. The information thus retrieved is subsequently represented within the Matlab environment in a unified structure, open for further additions. As defined in the Neuroshare API (Neurshare Consortium, 2004), four basic data entities exist within this structure: *Analog, Event, Segment* and *Neural*. Their properties can be described in brief as:

- An *Analog* entity contains continuous data, e.g. recordings of the voltage variable sampled over a certain time interval. One analog entity contains a single recording channel, i.e. one continuous time series.
- An *Event* entity contains points in time at which a specific event, e.g. a particular stimulus occurred. These entities consist of a time stamp and a label per event. Spike times are NOT stored here — they are stored as a *Neural* entity.
- A *Segment* entity contains a specific part of an analog entity with a specific duration that starts at a certain point in time, e.g. cut-outs of all spike wave-forms from one analog entity.

- A *Neural* entity contains special data that was derived from the original data set and is defined as a neural signal. Thus far, spike times of one single-unit or one multi-unit channel is considered as a neural signal and stored as one neural entity. Multiple single-unit recordings are consequently represented as separate cells within the *neural* entity class. Further additions consisting of analog and segment data as substructures within the neural entity are possible.

3.2. External data representation and storage

FIND supports the import of data in various proprietary formats and represents it as a unified data structure within MATLAB. We implemented an export of this structure to the Hierarchical Data Format (HDF5). HDF5 is a unique technology suite enabling the management of very large and complex data collections. It is widely in use in many scientific areas (e.g. weather forecasts and particle physics). The major advantages of the HDF5 technology suite (see e.g. http://hdf.ncsa.uiuc.edu/index.html), are (i) it is a completely portable file format with no limit on the number or size of data objects in the collection, (ii) it is distributed as a software library that runs on a range of computational platforms, from laptops to massively parallel systems, and it implements a high-level API with C, C++, Fortran 90, Java and Python interfaces, (iii) it has a rich set of integrated performance features that allow for access time and storage space optimizations, and finally (vi) there is a variety of tools and applications for managing, manipulating, viewing, and analyzing the data in the collection. Most importantly, the HDF5 data model, file format, API, library, and tools are open source and distributed without charge.

An HDF5 file has a hierarchical structure and appears to the user as a directed graph, conceptually similar to the UNIX type file system, i.e. it can consist of "folders" and "files". A Data set, which corresponds to files, comprises collections of scalars or arrays. Moreover, HDF5 is unique in its ability to physically and conceptually separate data (including raw data and stored analysis results) from metadata (stored as data attributes), even if they are merged in the same file entity.

Benefiting from this structure, FIND additionally provides tools for merging and consolidation of data sets exported to HDF5. This allows for efficient storage and easy access to large data collections. Moreover, analysis of such a collection can make use of the hierarchical data structure of HDF5, e.g. by the concurrent analysis of all branches and leaves of a given hierarchical tree that share certain properties. In effect, the use of HDF5 provides many beneficial features that are similar to those offered by a conventional data basing system, but does not require the installation and administration of dedicated data base servers and software

3.3. Addressing the need for different programming languages by using plug-ins

There are numerous analysis methods that are computationally expensive and are, thus, difficult to implement for large data sets. Fortunately, distributed computing, i.e. the use of computing clusters, can be used as a measure to reduce these problems. Usually, such clusters are not specifically designed to work with Matlab. Therefore, we developed a Python-based plug-in to FIND that is operating on the HDF5 data structure (details see above). This plug-in uses open MPI (www.open-mpi.org, (Gabriel et al., 2004)) to send analysis packages to the nodes of the cluster. The results of the calculation are then stored within the same HDF5 structure that holds the original raw data. A detailed example of this approach was outlined elsewhere (Meier, Garbers, Haeussler, Egert, & Aertsen, 2008).

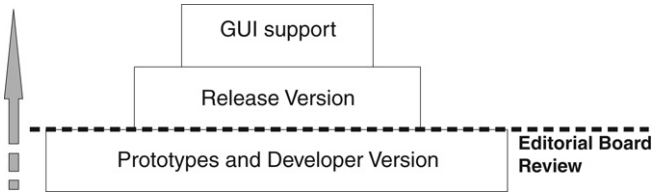

Fig. 2. Levels of function access and release within the FIND framework. We have established three stages of code addition to FIND. On the first level (developers' version) prototypes are submitted and included into the FIND framework. Within this version, adaptation to the FIND standards, testing and documentation are accomplished. After passing the review of an editorial board the functions are offered to the public as a release version. The full support of all functions available using the GUI is the highest level of abstraction offered by our framework. Stability and coverage of the functionality increases with higher levels within FIND. We warmly welcome contributors, developers and users of the FIND toolbox for all levels mentioned.

3.3.1. Application of tools

FIND provides access to the data at two different levels of abstraction. At the basic level, FIND provides functions that represent elemental algorithms for individual steps of data analysis. These can be called at the command line or integrated and combined in Matlab scripts and software projects. At the higher level, FIND provides a GUI for easy and fast access to data analysis, also for the inexperienced programmer. It allows the selection of data to be loaded and the navigation through the accessible data. Whenever a function to process data is called from the GUI level, this generates a function call that could also be issued at the command line and the function calls can be stored in a log-file (FIND history). This functionality eases the transition from the GUI usage to the level of automated and reproducible script-based analyses.

3.3.2. Tool development and integration

To allow the addition of further tools to the FIND framework we provide template functions. They specify the structure of the GUIs, the interface, and contain functions to test the parameters of the functions called as well as hierarchical error handling. Guidelines on how to structure the documentation of new tools are defined to ensure standardized calls and interfaces to the data structure. Such guidelines guarantee long-term stability of the FIND framework and ensure the interoperability of its tools. Note, that we also devised guidelines for FIND users, in particular: users should reference the functions used to analyse their data in their publications and cite the original publications referenced in the documentation of these functions. This is one of the rewards for the developers and it will assure reproducibility of the findings presented.

3.3.3. Quality management

As a measure to ensure a high quality of the tools and software code in the FIND toolbox we installed a three-level structure of the toolbox (Fig. 2). The first level (developers' version) comprises the highest functionality, but it is also the most unstable version. At this level all testing and adaptation takes place. When the developers are satisfied, the approval of a member of the separately installed review board is required to allow any tool to 'go public'. After positive review, tools are included into the release version, the second level in the hierarchy. Here, all functionality can be used on the function level. On the final and third level, full support of functionality of included functions is ensured within a GUI. One important concept of the FIND framework is that everyone is invited to join the developers' team and will be granted access to the currently developed functions. We expect that testing of the software tools will be performed in close cooperation with other

developers. For the exchange of reports, we have installed a bug tracking and code annotation system that has proven its usefulness in many open source projects (winner of the UK Linux & Open Source Awards 2006).

3.3.4. Teaching and training

Today, the neurophysiological data obtained in ever more complex experiments at the cellular as well as the systems level require appropriate methods of analysis. Thus, within the neuroscience community increasing importance is placed on high quality teaching and training of model-driven approaches to data analysis. FIND supports this in two ways. The FIND website provides teaching material to be used for practical training in data analysis and numerical data simulation. This material will include self-contained exercises based on experimental data samples and tools implemented in FIND. In addition, compact workshops on data analysis with FIND address students and scientists with hands-on experience in data analysis.

4. Functionality

Currently, the FIND toolbox is focused on single- and multi-channel spike train characterization (e.g. Meier and Gruen et al. (2007)). However, FIND also offers specialized tools to analyze continuous time series such as local field potentials (LFP), electroencephalography (EEG), intracellular voltage and current recordings, etc., as well as non-specialized tools, e.g. for information theory based analyses. Generally, we aim at the inclusion of additional tools for any type of electrophysiological (e.g. the MEA-tools, http://material.brainworks.uni-freiburg.de/research/meatools; Egert et al. (2002)) and neural imaging data.

To create reference datasets, the FIND toolbox provides several tools for simulating stochastic point process models. These are particularly helpful for testing and calibrating new tools of analysis or even complex analysis scripts. In the next section we demonstrate some of the functions available in the FIND toolbox in two different analyses of example data sets. An updated list of all currently released functions and GUIs is available on the FIND website.

5. Examples of analysis with FIND

To illustrate the functionality of FIND we exercise step by step two examples of data analysis using functions in FIND: The first example estimates dynamic changes of inter-spike-interval statistics to characterize stimulus-driven dynamic changes in spiking irregularity in a certain insect neuron type. The second example illustrates the layer specific response latencies to sensory input in the primary visual cortex of the rat. Where applicable, we refer to other analysis tools available within FIND to demonstrate the effects of interaction of different data processing tools. The complete data sets analyzed in this section along with documented software scripts is publicly available on the FIND website for reproduction of the results and hands-on experience with FIND tools. All computations were made under Matlab version 7.1.

5.1. Odor-response dynamics of spiking irregularity in a mushroom-body extrinsic neuron of the honeybee

The activity of spiking neurons *in vivo* naturally exhibits variability on various short and long time scales. An individual spike train observed under spontaneous activity conditions shows a typical irregular temporal pattern of individual action potentials. This observation has led to the notion of *stochastic* spiking, and it has introduced the analysis and modeling of spike trains within the framework of point process theory (e.g. Dayan and Abbott (2001), Johnson (1996), (Perkel, Gerstein, & Moore, 1967a, 1967b) and Tuckwell (1988). One prominent example of point processes used in theoretical neuroscience is the class of renewal processes. It assumes that the inter-spike interval (ISI) can be modelled as a random variable that is independently and identically distributed (IID) according to some pre-defined interval distribution. The variance of this distribution indicates the variability of ISIs.

Empirically, we can estimate statistical parameters of the experimental ISI distribution. The coefficient of variation

$$\mathrm{CV} = sd(\mathrm{ISI})/\mu(\mathrm{ISI}) \quad (1)$$

with the standard deviation sd of ISIs normalized by the mean μ of ISIs, is often used to measure spike train irregularity. It indicates variability on a short time scale, determined by the mean interval μ. In practice, the application of the CV is limited to cases of a stationary rate. A time-varying firing rate modulates the interval length, violating the assumption of a fixed ISI distribution. Thus the CV estimated from such a spike train is not a useful measure to describe the randomness of the underlying spiking process. In our experiments, however, we typically observe neurons in stimulus-response conditions or during behavioral or mental tasks associated with pronounced changes of the firing rate. Thus, it is of interest to investigate whether single neuron variability is increased or reduced during such rate responses. Several measures that are local in time have been derived from the CV to be able to measure temporal changes of interval variability (Davies, Gerstein, & Baker, 2006; Holt, Softky, Koch, & Douglas, 1996; Miura, Okada, & Amari, 2006; Shin et al., 2007; Shinomoto, Miura, & Koyama, 2005). Here, we take a different approach. We first transform the experimental time axis to a new time axis where the empirical firing rate becomes constant. We call this new time axis 'operational time', a term that has previously been introduced for the special case of a non-homogenous Poisson process. In a second step, we apply the time-resolved analysis of the CV in a sliding window. As an example data set for this analysis, we chose a single-unit recording from the alpha lobe of the honeybee mushroom body. The data analyzed in this section was kindly provided by Martin Strube and Randolf Menzel at the Freie Universität Berlin, Germany. Preliminary results have been published in abstract form (Nawrot & Benda, 2006).

5.1.1. Experiments

Several different odors were presented to the antennae of a harnessed honey bee (*apis melifera*) for 3s during repeated trials to investigate stimulus specificity and trial-by-trial response reliability in mushroom body extrinsic neurons (Strube, Stiller, Nawrot, & Menzel, 2007). A custom-built stimulation setup (adopted from Galizia, Joerges, Küttner, Faber, and Menzel (1997)) was used to inject a small volume from an odor chamber containing 10 μl of 10x-diluted odor in paraffin oil on filter paper into a constantly delivered air stream. Details of the *in vivo* preparation and recording method are described elsewhere (Okada, Rybak, Manz, & Menzel, 2007). In brief, extracellular differential recordings were performed from three thin polyurethane-coated copper wire electrodes (Okada, Ikeda, & Mizunami, 1999) inserted into the ventral region of the alpha lobe of the mushroom body at the border of the peduncle. Raw signals were recorded, band-pass filtered between 0.1–9 kHz, and digitized at 20 kHz (Lynx-8 Amplifier system, Neuralynx, Tucson, AZ, USA). A template-based spike sorting was performed (Spike2 software, Cambridge Electronic Design, Cambridge, UK).

5.1.2. Rate estimation

The raster plot in Fig. 3(a) depicts all 66 single-trial spike trains in response to peppermint during the first 800 ms of

Fig. 3. Time-resolved inter-spike interval statistics during odor presentation. (a) Repeated measurements of spiking activity from a single mushroom body alpha-lobe extrinsic neuron of the honeybee in response to $N = 66$ repeated stimulations with the same odor (peppermint). Each tick marks the time of occurrence of one spike, each row of spikes represents one experimental trial. The presentation of the odor stimulus with onset at time $t = 0$ is indicated by the gray shading. (b) Trial-averaged rate function, estimated using an alpha-shaped convolution kernel with an optimal kernel width of $\sigma = 8$ ms (inset). (c) Illustration of the time-unwarping method. Event times in experimental time (top) are translated into event times in operational time (right) according to the integrated rate function (black curve). The gray shading again indicates the odor stimulation. Operational time has no units, but counts indicate the number of expected events, i.e. for the operational time interval [0, 10] we expect on average 10 spikes per trial. The stimulus onset time $t' = t = 0$ serves as a reference in both time frames. (d) All 66 single-trial spike trains in operational time. The rate of spike events is now normalized to unity. (e) Time-resolved CV(t) of the inter-spike intervals in experimental time. CV(t') was first measured in operational time and subsequently presented in experimental time, using the inverse time transformation $t' \rightarrow t$ to warp the time axis. The results of three different methods of estimation are shown (see text). The red curve represents the average of the trial-to-trial estimated CV_i; the blue curve gives the CV based on the pooled ISIs from all trials. The black curve assumes a gamma renewal model fitted the ISI distribution (shown in f), where $CV = 1/sqrt(\alpha)$. (f) This time-resolved fit of a gamma distribution to the ISIs pooled from all trials gives the gamma order α of the model. For comparison, the gray dashed line indicates the gamma order of a Poisson process. (g) The time-resolved histogram of the ISI distribution in experimental time (normalized to area = 1).

odor presentation (gray shading), aligned to stimulus onset (valve opening time) at $t = 0$. We obtained an estimate of the time-varying and trial-averaged rate function in Fig. 3(b) using the method of kernel convolution (Nawrot, Aertsen, & Rotter, 1999; Parzen, 1962). We chose a kernel of asymmetric shape, the so-called alpha-function, as depicted in the inset of Fig. 3(b), which is implemented in the FIND function **makeKernel**. The crucial parameter for obtaining a good rate estimate is the kernel width which determines the time resolution of the estimate. We applied an automatic method to determine the optimal kernel width as described in Nawrot et al. (1999). This is implemented in the function **optimizeKernelWidth**. To generate a kernel of predefined shape and width it calls the function **makeKernel**. We parameterized the kernel width by the standard deviation σ of the associated distribution (Nawrot et al., 1999). The optimum kernel width was determined here as $\sigma = 8$ ms, which was subsequently used to construct the optimized rate estimate shown in Fig. 3(b). To obtain a trial-averaged estimate of the rate, we first constructed a discrete array representation (time resolution 0.5 ms) of each single trial spike train. Averaging across all trials results in a discrete representation of the average spike train. We convolved this array with the kernel array using Matlab's *filter* function.

5.1.3. Time warp

In a next step, we transformed the experimental time axis t where all single spike trains are aligned with respect to the temporal marker of the stimulus onset at time $t = 0$ to the operational time axis t', defined as the time frame where the trial-averaged event rate is constant, i.e. $\lambda(t') \equiv 1$. We will briefly outline this procedure and refrain from a detailed description of its mathematical basis. For an in-depth description of the theoretical concept of time rescaling and the underlying assumptions we refer to previous publications (Brown, Barbieri, Ventura, Kaas, & Frank, 2001; Johnson, Tsuchitani, Linebarger, & Johnson, 1986). The method as implemented in FIND and as outlined here has previously been described and applied by Reich, Victor, and Knight (1998) and by Nawrot et al. (2008).