*** Title Page (With all author details listed)**

FIND – a unified framework for neural data analysis

Ralph Meier[1*] , Ulrich Egert[1,2], Ad Aertsen[1] and Martin P. Nawrot[3,1]

1) Bernstein Center for Computational Neuroscience, Albert-Ludwigs-University, Freiburg, Germany.

2) Department of Microsystems Engineering, Faculty of Applied Sciences, Albert-Ludwigs-University Freiburg, Germany

3) Neuroinformatics and Theoretical Neuroscience, Institute of Biology – Neurobiology, Freie Universität Berlin and Bernstein Center for Computational Neuroscience Berlin, Germany

* Corresponding Author:

Ralph Meier
BCCN Freiburg
Hansastrasse 9a
79104 Freiburg, Germany

Phone: +49 – 761 203 2864
Fax: +49 – 761 203 2860

meier@biologie.uni-freiburg.de

FIND – a unified framework for neural data analysis

Acknowledgements

We are grateful to Martin Strube and Randolf Menzel at the Institute of Biology – Neurobiology, Freie Universität Berlin, Germany for providing us with the honeybee data and to Clemens Boucsein and Dymphie Suchanek at the Bernstein Center for Computational Neuroscience Freiburg, Germany for making available the rat cortex data. We thank all developers of FIND and specifically Stefan Rotter for contributing the code for the *warpTime,* the *unWarpTime,* and the *makeSavGol* functions used in this paper. Furthermore, we thank Sonja Grün for supporting FIND. The FIND framework is supported in parts by the German Federal Ministry of Education and Research (BMBF) grant 01GQ0420 to the BCCN Freiburg and 01GQ0421 to Multi Channel Systems, and the 6th RFP of the EU (grant no. 15879-FACETS). The contribution of M.N. is funded by the BMBF grant 01GQ0413 to BCCN Berlin.

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 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 wide range of a steadily increasing number of tools. 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.

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

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 et al. 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.

Concept

FIND implements a unified import of multiple proprietary data formats, based on the Neuroshare Project (http://neuroshare.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 GmbH, NeuroExplorer, Plexon Inc., Tucker-Davis Technologies, Cyberkinetics Inc. and others as well as data from network simulations environments (e.g. NEST: www.nest-initiative.org, Morrison et al. 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's website **http://find.bccn.uni-freiburg.de**.

Design and Community Interaction

In this section we briefly outline the design structure of the FIND framework (Figure 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.

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 (Neuroshare 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.

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 is 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

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 et al. 2008).

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.

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.

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 (Figure 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).

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.

Functionality

Currently, the FIND toolbox is focused on single- and multi-channel spike train characterization. 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.

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.

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. Perkel et al. 1967a,b; Tuckwell 1988, Johnson 1996, Dayan & Abbott 2003). 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

$$CV = sd(ISI)/\mu(ISI), \qquad (1)$$

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 (Softky et al. 1993, Shinomoto et al. 2005, Miura et al. 2006, Davies et al. 2006, Shin et al. 2007). 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).

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 et al. 2007). A custom-built stimulation setup (adopted from Galizia et al. 1997) was used to inject a small volume from an odor chamber containing 10µl 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 et al. 2007). In brief, extracellular differential recordings were performed from three thin polyurethane-coated copper wire electrodes (Okada et al. 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 0.1-9kHz, and digitized at 20kHz (Lynx-8 Amplifier system, Neuralynx, Tucson, AZ, USA). A template based spike sorting was performed (Spike2 software, Cambridge Electronic Design, Cambridge, UK).

Rate estimation. The raster plot in Fig. 3a depicts all 66 single-trial spike trains in response to peppermint during the first 800 ms of 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. 3b using the method of kernel convolution (Parzen et al. 1962; Nawrot et al. 1999). We chose a kernel of asymmetric shape, the so-called alpha-function, as depicted in the inset of Fig. 3b, 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 σ=8 ms, which was subsequently used to construct the optimized rate estimate shown in Fig. 3b. 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.

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 et al. 2001, Johnson et al. 1986). The method as implemented in FIND and as outlined here has previously been described and applied by Reich et al. 1998 and by Nawrot et al. 2008.
We define the transformation of time by the integral over the trial-averaged rate function $\lambda(t)$ up to the time t

$$t' = \Lambda(t) = \int_0^t ds\, \lambda(s) \qquad (2)$$

where λ(t) describes the explicitly time-dependent intensity function of the process. Obviously, for our set of experimental spike trains we cannot know the 'true' stimulus related response rate function λ(t). Instead, we may use the empirical trial-averaged estimate in Fig. 3a, assuming that it was a reasonably good approximation. To give an intuitive picture of the time transformation: We warped the time axis such that we straighten out the rate function until it becomes constant, i.e. we need to stretch time where the rate is high and compress time where the rate is low. The schematic drawing in Fig. 3c visualizes this transformation for individual spike times. In the top panel we indicate the spike times of the first experimental trial, each by a single tick mark. The monotonic black curve in the central panel depicts the integral Λ(t) over the rate function in Eqn. 2 as a function of experimental time *t*. To translate an individual experimental spike time (horizontal axis) into the associated spike time in operational time (vertical axis) we need to pass it through the transformation, as indicated by the blue lines. In FIND we applied the function **unWarpTime** to each single trial to perform this transformation numerically. The spike raster in Fig. 3d shows the resulting spike trains for all 66 trials in operational time.

Time-resolved estimation of CV. After the transformation of all spike times to operational time we are now in the position to estimate the CV of ISIs (Eqn.1) in a sliding window in operational time t' where the firing rate is constant and, thus, does not distort the interval distribution. The choice of the window width is a trade-off: For smaller windows we gain temporal resolution, for larger windows we reduce the variance of the estimator and the bias of the estimator caused by short observation windows (Nawrot et al. 2008). Here, we chose a width of $w' = 5$ in operational time, i.e. a window that covers on average 5 intervals per trial. We then slid the window along operational time and in each of *n* time steps measured the CV by two alternative methods, resulting in the blue and red curves in Fig. 3e, respectively. The first method collected in all 66 trials all intervals that fell within the observation window of length w', which yielded, on average, 66 x 5 = 330 intervals. Fig. 3g depicts the time-resolved histogram of ISIs. From these intervals, we calculated the CV. The second method computed for each trial *i* separately CV_i across all ISIs within the observation window. Subsequently we averaged across all 66 CV_i values to obtain the trial averaged CV. For each method we, thus, obtained a discrete vector CV(t') of length *n* in operational time.
To compare the CV to other dynamic variables of the experiment, e.g. the firing rate of the neuron, the stimulus times, or the behavioural dynamics associated with the experimental paradigm, it is, however, desirable to present the time-resolved vector CV(t) in experimental time. We therefore back-transformed the time vector $T' = t'_1, \ldots, t'_n$, which is equally spaced in operational time, using the inverse transformation of Eqn. 2 to translate it into a vector $T = \Lambda^{-1}(T')$ in experimental time. In FIND this is implemented in the function **warpTime**. Note that the resulting time points $t_1, \ldots, t_n$ are *not* equidistant. We then plotted the vector CV(t') against *T* to visualize dynamic changes of the CV(t) in real time (Fig. 3e).

Dynamic changes of the gamma order. The model of a gamma renewal process (Cox et al. 1966) is a frequently used point process model to describe the statistics of single neuron spiking (Kuffler57, Stein 1965, Tuckwell 1988, Teich et al. 1997, Barbieri et al. 2001, Brown et al. 2001). Its inter-event intervals are independently and identically distributed according to a gamma distribution which is defined by two parameters: the process rate and the so-called order α of the gamma-process defining the shape of the distribution. The Poisson process marks the special case of a gamma process of order $\alpha = 1$ and has an exponential interval distribution. A gamma process of order alpha < 1 exhibits a higher variability of the ISIs,

whereas a process of order alpha > 1 shows less variable ISIs and, thus, more regular spike trains. The non-homogeneous gamma process is a generalization that incorporates a dynamic rate function (Reich et al. 1998, Barbieri et al. 2001, Brown et al. 2001, Nawrot et al. 2008). We performed a time-resolved fit of the gamma distribution to the empirical distribution of ISIs in operational time as pooled from all trials, again in a moving window of width w'=5. For this, we used the **gamfit** function in the statistics toolbox of Matlab to obtain a maximum likelihood estimate of α. The resulting vector $\alpha(t)$ as a function of experimental time is shown in Fig. 3f. Observe that the estimated shape parameter is not constant over time. Rather, it rises sharply during the initial phase of the neuron's odor response. For the gamma model we obtained the model-based estimate of CV = 1/sqrt(α) shown in Fig. 3e (black curve). Fig. 3g depicts the time-resolved histogram of the empirical ISI distribution as estimated in the moving window. The observed dynamic change of the shape parameter α shows that the spiking of this neuron could not be modelled by a rate-modulated gamma renewal process which assumes that the order of the gamma process is fixed, independent of the process rate.

Conclusion and Interpretation. We found in this example of a mushroom body extrinsic neuron that it dynamically reduces CV of ISIs during the initial phasic period of its phasic-tonic stimulus response. Estimation of CV on a trial-by-trial basis (Fig. 3e, red curve) generally yielded lower values than CV based on the pooled across-trial interval distribution (blue curve). This might indicate that a non-trivial trial-by-trial variability of the spike rate led to an increase of the variance of the ISI distribution (Nawrot et al. 2008). The observed inverse relation of irregularity and rate may have different origins. On the one hand, neuron-intrinsic mechanisms could be responsible, e.g. an absolute refractory period of the neuron leads to an increasing regularization with increasing rate (Johnson 1996). We could show elsewhere (Farkhooi et al., submitted) that this neuron type typically exhibits negative serial interval correlation. This indicates the presence of a mechanism for spike-frequency adaptation, which in turn could have a similar regularizing effect. On the other hand, the stimulus related reduction of interval variability might also be a consequence of an altered, possibly oscillatory structure of the input from the presynaptic Kenyon cells following stimulus onset, as has been previously suggested in the locust (Laurent et al. 1994, Cassenaer et al. 2007

Layer-specific stimulus response latency in the rat visual cortex

In this example, we analyzed the temporal pattern of activity onset across electrodes in different cortical layers in response to visual bright full-field stimulation. For this, we used a tool for the automatic alignment of waveforms based on pair-wise cross-correlation and aligned the time derivatives of multi-unit spike rates. The same tool has been applied in a recent study (Krofczik et al. 2007 submitted). The data analyzed in this section was kindly provided by Clemens Boucsein and Dymphie Suchanek, Bernstein Center for Computational Neuroscience, Albert-Ludwigs-University Freiburg, Germany.

Experiments. To study response latencies of neuronal populations in the visual cortex of the anaesthetized rat, repeated full-field bright flash stimuli of 200 ms duration were applied to the contra-lateral eye using a white LED. In brief, an adult Sprague-Dawley rat was anaesthetized with urethane. Supplementary doses of ketamin and xylazine were added every 30-50min. An 3 x 4 array (540µm inter-electrode distance) of glass-coated single platinum-tungsten electrodes (Thomas Recording, Giessen, Germany) was used to record from different

cortical layers II-VI. Each electrode was lowered independently until it reached a predefined cortical depth. Signals were band-pass filtered between 0.1-5kHz and amplified (Multi Channel Systems, Reutlingen, Germany). After the experiment, DC current was passed through the electrodes for silver deposition. After tissue fixation, brains were sliced with a vibratome (100µm vibroslice, Campden instruments, Loughborough, UK) and a standard silver intensification was applied (HQ silver, Nanoprobes, Yaphank, NY). Electrode positions were recovered with a conventional microscope and sections containing silver deposits were counterstained with cresyl violet to determine laminar borders.

Spike detection: Multi-unit activity was detected using the **spikeDetection** method implemented in the FIND toolbox. From the rectified signal of each recording channel, it calculates the median *m*. Spike times are determined as the times where the signal crosses the threshold n x *m*. For the current data set we used n=8 for all electrodes. We here analyzed 7 electrodes for which the recording layer could be unequivocally recovered (Fig. 4a) and which showed MUA responses to the visual stimulation.

Estimation of firing rates and their derivatives. In a first step, we estimated the trial-averaged multi-unit rate function for each channel separately. We again used the method of kernel convolution as described above, however, with a symmetric Savitzky-Golay filter kernel (Savitzky & Golay 1964; Press et al. 1992, order 0, width *w*=10ms). This was obtained by the FIND function **makeSavGol**. Fig. 4c shows the resulting firing rate profiles for all channels. Color indicates the layer of the respective recording electrode (Fig. 4f). How can we define the response onset for each individual electrode? One possibility would be to use a threshold detection method. Another option is to define the onset by the maximum slope of the rate increase after stimulus onset at *t*=0, i.e. the point in time where the time derivative of the rate function is positive and maximal. In a further step we therefore estimated the temporal derivative of the spike rate from the trial-averaged spike train, using a centered asymmetric Savitzky-Golay filter (order 1, width *w*=20 ms). In each step of the numeric convolution, the asymmetry of the filter automatically results in an estimate of the slope of the rate for all data points that fall within the kernel window. For most channels, the derivatives showed a marked peak during the early response (Fig. 4d).

Estimation of temporal delays between channels. The central step of our analysis estimates the relative time-delays between the responses on individual channels. The method used is implemented in the FIND function **autoAlign** and is explained in detail elsewhere (Nawrot et al. 2003). Here, we used it to align the derivatives of the multi-unit firing rates. However, the method can be used for the alignment of any data vectors in time or space. In brief, we normalized the estimated rate derivative for each channel by its integral. We then calculated the cross-correlation for each pair (*i,j*) of derivatives for different relative time lags t, resulting in ½ * N * (N-1) cross-correlation functions $CC_{ij}(t_{ij})$. A parabolic function $p_{ij}(t_{ij})$ was fitted to the peak of each individual function $CC_{ij}(t_{ij})$. The sum $P(t)=\sum_{i,j} p_{ij}(t_{ij})$ is a parabolic sum with N-1 degrees of freedom and we can numerically determine its unique maximum for N-1 relative time delays τ_k. We arbitrarily set the Nth delay $\tau_N=0$ and then subtracted from each delay the mean delay to obtain N delays τ'_k with zero mean. Then we shifted each single channel spike train in time by the interval $\Delta t_k = -\tau'_k$ to optimally align all spike trains. The center of mass of the average derivative profile after alignment remains unchanged with respect to the initial state as the mean shift is zero. We repeated this procedure iteratively until the total sum of resulting shifts reached some lower threshold. In our example, after the second run the total sum of resulting shifts was already smaller than 5 ms and we terminated the iteration. The firing rates estimated from the aligned spike trains are shown in Fig. 4e. The time t=0 now no longer indicates the time of stimulus onset, but rather the average onset time.

The method implemented in **autoAlign** yields N-1 relative shifts. It can, thus, be used to align data without knowledge of any external event or trigger, but we cannot infer the absolute latency with respect to an external trigger.

Estimation of the absolute latencies. To finally calculate the absolute latencies of response onset with respect to the stimulus onset for each channel, we estimated an absolute value for the mean latency across all channels, and then subtracted for each channel the relative shifts τ'_k as computed in the previous step. To do this, we averaged the aligned spike trains from all channels and then applied the function **estimateLatencyByDerivative** in FIND. It first estimates the derivative, again using an asymmetric Savitzky-Golay filter as explained above, and then detects its peak. In our example, the average latency was estimated to be 104 ms. To this value, we added the individual relative latencies $-\tau'_k$ to obtain the individual values for all 7 channels (Fig. 4f). Unfortunately, we could not readily compute confidence intervals for these latencies due to limitations of the present data set. In principle, the reliability of the layer specific onset dynamics could be estimated by repeated analysis over recordings in different animals. A second possibility would be to quantify inter-trial variability. However, single-trial spike rates in our example were quite low and, thus, rate functions and the subsequent latencies could not be safely estimated on a single trial basis.

Conclusion and Interpretation. By selection, one can choose between a smoothing and a time-derivative Savitzky-Golay filter by adjusting the order of derivative of the filter polynomial. During the alignment procedure the correlation functions were normalized, which in effect equally weighs each profile and, thus, makes it independent of individual firing rates. The single channel latencies were obtained from the alignment in two steps: We first calculated the N-1 possible *relative* latencies between individual channels; only in a second step, the absolute latencies of rate onset were determined from the trial averaged activity. This sequence increases the reliability of the estimate, because we use the N-fold amount of data compared to a single channel onset detection. Our preliminary results for the present data indicate that after visual stimulation the firing rate of multi-unit activity tends to increase earlier in layer IV of the rat primary visual cortex than in layers V and VI. This is in line with the concepts of layer specific processing of sensory information in the visual cortex established in different species (e.g. Mitzdorf and Singer 1978, Kenan-Vaknin and Teyler 1994, Nowak et al 1995, Hirsch et al. 2002 and Hirsch et al 2006).

Conclusions, Discussion and Outlook

FIND establishes a framework for a single code repository for developing, exchanging and discussing code for the analysis of neural data, and it includes a unified access to various proprietary and free data formats. FIND is based on the Matlab programming language for two major reasons, (i) it is probably the most widely spread high level programming environment in theoretical as well as experimental laboratories in systems neuroscience today, and (ii) it provides an intuitive approach for students and scientists that are not formally trained in any programming language. Moreover, there already exists a large collection of tutorials and courses on Matlab as well as Matlab-based courses in the neurosciences. FIND wants to contribute to the general effort within the computational neuroscience community towards independently verifiable results. To achieve this goal, we believe it is necessary to develop an open source platform that is transparent and inclusive, and which provides effective means of quality management. It should allow the exchange of and comparability across data analysis methods. Central features of FIND comprise (i) the unified data import, representation and storage allowing for a standardized interfacing, independently of the experimental hardware and software used, and (ii) the hierarchical toolbox structure in combination with a review board that distinguishes a developmental test stage, a release version with full and tested functionality, and the graphical user interface granting easy access with limited functionality. The FIND initiative is aware of the difficulties in generating and maintaining a toolbox that will be accepted by a larger community and, thus, guarantees a high and sustained quality level. We outlined our strategy to ensure support by contributors and acceptance with the users. This strategy also includes the collaboration with competent partners, i.e. individuals and scientific labs with an internationally distinguished reputation in the development of data analysis methods. In general, our strategy aims to be in line with the goals and solution approaches discussed at the first INCF workshop on sustainability (van Hoorn and van Pelt 2008). We consider FIND as a stepping-stone towards a community based data analysis and storage solution. Moreover, the FIND initiative develops curricula for graduate courses on the collection and analysis of neural activity data that make use of the FIND framework. We invite *developers* of data analysis methods and *users* that are looking for sophisticated solutions to their data analysis problems to become a member of the FIND community.

References

Barbieri R, Quirk MC, Frank LM, Wilson MA, and Brown EN. (2001) Construction and analysis of non-Poisson stimulus-response models of neural spiking activity. J Neurosci Methods, 105, 25–37.

Baxter SM, Day SW, Fetrow JS, Reisinger SJ. (2006) Scientific software development is not an oxymoron. PLoS Comput Biol. 2(9), e87.

Brown EN, Barbieri R, Ventura V, Kaas RE, Frank LM. (2001) The time-rescaling theorem and its application to neural spike train data analysis. Neural Comput, 14, 325–46.

Cassenaer S, Laurent G. (2007). Hebbian STDP in mushroom bodies facilitates the synchronous flow of olfactory information in locusts. Nature, 448(7154), 709-13.

Cox DR and Lewis PAW. (1966). The Statistical Analysis of Series of Events. Methuen's Monographs on Applied Probability and Statistics. London: Methuen

Dayan P, Abbott LF. (2001) Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems. MIT Press

Davies RM, Gerstein GL, Baker SN. (2006) Measurement of time-dependent changes in the irregularity of neural spiking. J Neurophysiol, 96, 906–18.

Egert U, Knott Th, Schwarz C, Nawrot M, Brandt A, Rotter S, Diesmann M. (2002) MEA-Tools: an open source toolbox for the analysis of multielectrode-data with Matlab. J Neurosci Methods, 117, 33-42

Farkhooi F, Strube M, Nawrot MP. Experimental evidence and point process modeling of serial correlation in neural spike trains (*submitted*)

Gabriel E., Fagg GE., Bosilca G. Angskun T., Dongarra JJ., Squyres JM., Sahay V., Kambadur P., Barrett **.**, Lumsdaine A., Castain RH., Daniel DJ., Graham RL., Woodall TS.. (2004). Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. In Proceedings, 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary, September 2004.

Galizia CG, Joerges J, Küttner A, Faber T, Menzel R. (1997) A semi-in-vivo preparation for optical recording of the insect brain. J Neurosci Meth, 76, 61–69

Heimel JA, Van Hooser SD, Nelson SB. (2005) Laminar organization of response properties in primary visual cortex of the gray squirrel (Sciurus carolinensis). J Neurophysiol., 94(5), 3538-54.

Hirsch JA, Martinez LM, Alonso JM, Desai K, Pillai C, Pierre C. (2002) Synaptic physiology of the flow of information in the cat's visual cortex in vivo. J Physiol, 540(Pt 1), 335-50.

Hirsch JA, Martinez LM. (2006) Laminar processing in the visual cortical column. Current Opinion in Neurobiology, 16(4), 377-384.

Holt GR, Softky WR, Koch C, Douglas RJ. (1996) Comparison of discharge variability in vitro and in vivo in cat visual cortex neurons. J Neurophysiol, 75, 1806–14.

van Horn J, van Pelt J. (2008) 1st INCF Workshop on Sustainability of Neuroscience Databases, Stockholm, Dec. 13/13, 2007. Available at International Neuroinformatics Coordinating Facility Secretariat Stockholm, Sweden.

Johnson DH, Tsuchitani C, Linebarger DA, Johnson MJ. (1986) Application of a point process model to responses of cat lateral superior olive units to ipsilateral tones. Hearing Res, 21, 135-159

Johnson DH. (1996) Point process models of single-neuron discharges. J Comput Neurosci, 3, 275–99.

Kenan-Vaknin G, Teyler TJ. (1994) Laminar pattern of synaptic activity in rat primary visual cortex: comparison of in vivo and in vitro studies employing the current source density analysis. Brain Res, 635(1-2), 37-48.

Krofczik S, Menzel R, Nawrot M. Odor and odor mixture encoding by antennal lobe projection neurons in the honeybee (*submitted)*

Kuffler SW, Fitzhugh R, Barlow HB. (1957). Maintained activity in the cat's retina in light and darkness. J Gen Physiol, 40, 683-702

Laurent G, Naraghi M. (1994) Odorant-induced oscillations in the mushroom bodies of the locust. J Neurosci.14(5 Pt 2), 2993-3004.

Meier R, Haussler U, Aertsen A, Deransart C, Depaulis A, Egert U. (2007). Short-term changes in bilateral hippocampal coherence precede epileptiform events. Neuroimage, 38(1), 138-49.

Meier R, Boven KH, Aertsen A, Egert U. (2007) FIND – Finding Information in Neural Data - An open-source analysis toolbox for multiple-neuron recordings and network simulations. Abstr. NWG Conference 2007 T38-1B

Meier R, Gruen S, Boucsein C, Aertsen A, Boven KH, Egert U (2007). Characterizing neural network dynamics: Analyzing neural activity data using the FIND – Toolbox, Soc. Neurosci. Abstr., Program No. 319.11.

Meier R., Garbers C., Haeussler U., Egert U., Aertsen A. (2008). Nonlinear interdependencies in epileptiform network dynamics revealed with the FIND toolbox using distributed computing FENS Abstr. vol 4, *150.22.*

Miura K, Okada M, Amari M (2006). Estimating spiking irregularities under changing environments. Neural Comp 18, 2359-2386

Mitzdorf U, Singer W (1978). Prominent excitatory pathways in the cat visual cortex (A 17 and A 18): a current source density analysis of electrically evoked potentials. Exp Brain Res. 33(3-4), 371-94.

Morrison A, Mehring C, Geisel T, Aertsen AD, Diesmann M (2005). Advancing the boundaries of high-connectivity network simulation with distributed computing. Neural Comput, 17(8), 1776-801.

Nawrot MP, Aertsen A, Rotter S (2003). Elimination of response latency variability in neuronal spike trains. Biol Cybern., 88(5), 321-34.

Nawrot M, Aertsen A, Rotter S (1999). Single-trial estimation of neuronal firing rates: from single-neuron spike trains to population activity. J Neurosci Meth, 94(1), 81-92.

Nawrot MP, Benda J (2006). Two methods for time-resolved inter-spike interval analysis. Berlin Neuroforum Abstracts 2006, 62

Nawrot MP, Boucsein C, Rodriguez-Molina V, Riehle A, Aertsen A, Rotter S (2008). Measurement of variability dynamics in cortical spike trains. J Neurosci Methods, 169, 374-90, doi10.1016/j.jneumeth.2007.10.013

Neurshare Consortium, The Neuroshare API (2004). Version 1.3, http://neuroshare.sourceforge.net/API-Documentation/NeuroshareAPI-1-3.pdf

Nowak LG, Munk MH, Girard P, Bullier J (1995). Visual latencies in areas V1 and V2 of the macaque monkey. Vis Neurosci.12(2), 371-84.

Okada R, Ikeda J, Mizunami M (1999). Sensory responses and movement-related activities in extrinsic neurons of the cockroach mushroom bodies. J Comp Physiol A, 185, 115-129

Okada R, Rybak J, Manz G, Menzel R (2007). Associative plasticity of mushroom body-extrinsic neurons during olfactory learning in honeybees. J Neurosci, 27(43), 11736 –47

Parzen E (1962). On estimation of a probability density function and mode. Ann of Math Stat, 33, 1065–76

Perkel DH, Gerstein GL, Moore GP (1967a). Neuronal spike trains and stochastic point processes. I. The single spike train. Biophys, 7, 391–418.

Perkel DH, Gerstein GL, Moore GP (1967b). Neuronal spike trains and stochastic point processes. II. Simultaneous spike trains. Biophys J, 7,419–40.

Press WH, Flannery BP, Teukolsky SA, Vetterling WT (1992). Numerical Recipes in C: The Art of Scientific Computing, , 2nd Ed, Cambridge: Cambridge University Press.

Reich DS, Victor JD, Knight BW (1998). The power ratio and the interval map: Spiking models and extracellular recordings. J Neurosci,18, 10090–10104.

Savitzky A, Golay MJE (1964). Smoothing and Differentiation of Data by Simplified Least Squares Procedures. Anal. Chem., 36, 1627

Shin S-L, Hoebeek FE, Schonewille M, De Zeeuw CI, Aertsen A, De Schutter E (2007). Regular patterns in cerebellar Purkinje cell simple spike trains, PLoS ONE 2(5), e485, doi:10.1371/journal.pone.0000485

Shinomoto S, Miura K, and Koyama S (2005). A measure of local variation of inter-spike intervals. Biosystems, 79, 67–72

Stein RB (1965). A theoretical analysis of neuronal variability. Biophys J, 5, 173–194.

Strube M, Stiller S, Nawrot MP, Menzel R (2007). Olfactory Coding in the Honeybee Brain III. Sparseness, Reliability and Reward Conditioning in Alpha-Lobe Extrinsic Neurons. 7th Göttingen Meeting of the German Neuroscience Society, Suppl. Neuroforum, 8(1), T20-3A

Teich MC, Heneghan C, Lowen SB, Ozaki T, and Kaplan E (1997). Fractal character of the neural spike train in the visual system of the cat. J Opt Soc of Am A, 14, 529–546.

Tuckwell HC (1988). Introduction to Theoretical Neurobiology, Volume 2. Cambridge: Cambridge University Press

Wilson GV (2006). Where's the Real Bottleneck in Scientific Computing? American Scientist, 94(1), 5-6, doi:10.1511/2006.1.5

Figure 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**.

Figure 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.

Figure 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 σ= 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' → 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(α). (**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).

Figure 4

Layer-specific response latency in rat primary visual cortex in response to optical full field stimulation. (a) The recovered positions of the tips of 7 electrodes (dots) within the indicated cortical layers. White lines indicate the boundaries of the layers. (b) The spike density histogram for all trials (N=37) aligned to stimulus onset (time t=0 in experimental time) for each electrode (light gray – low firing rate, dark: high firing rate). (c) The multi-unit activity rate profile for each electrode estimated using a smoothing Savitzky-Golay filter (half-width: 10 ms, order 0). (d) The first derivative of the firing rates (Savitzky-Golay filter, half-width 20 ms, order 1). These profiles were used for the autoAlign procedure. (e) The firing rates (as in (c) after alignment of rate derivatives. Note that the time axis for each trace is now relative to a zero mean shift (see text). (f) Absolute latencies of the maximal increase in firing rate for each electrode.

Figure 1

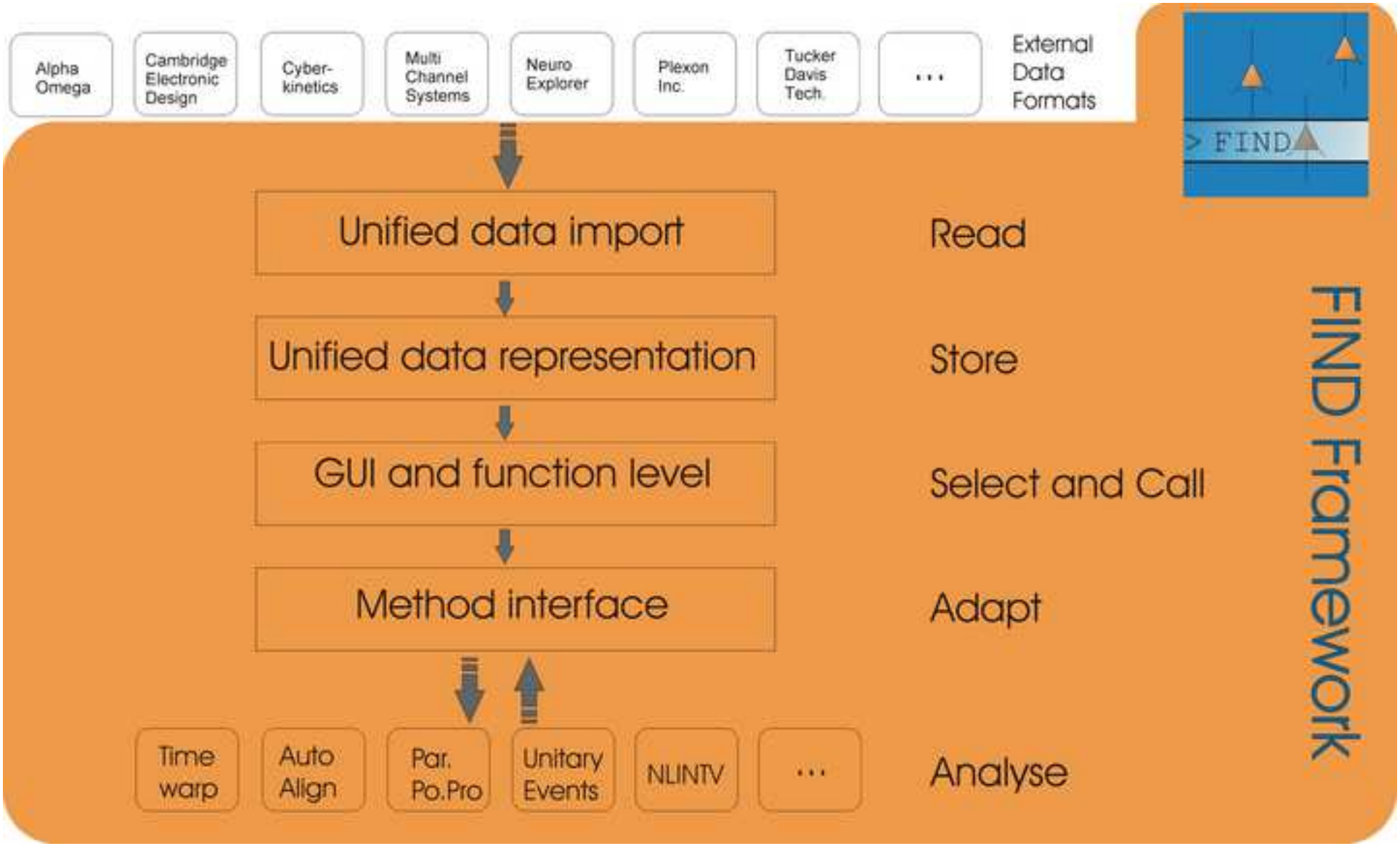

Alpha Omega
Cambridge Electronic Design
Cyber-kinetics
Multi Channel Systems
Neuro Explorer
Plexon Inc.
Tucker Davis Tech.
...
External Data Formats
> FIND
Unified data import
Read
Unified data representation
Store
GUI and function level
Select and Call
Method interface
Adapt
Time warp
Auto Align
Par. Po.Pro
Unitary Events
NLINTV
...
Analyse
FIND Framework

Figure 2

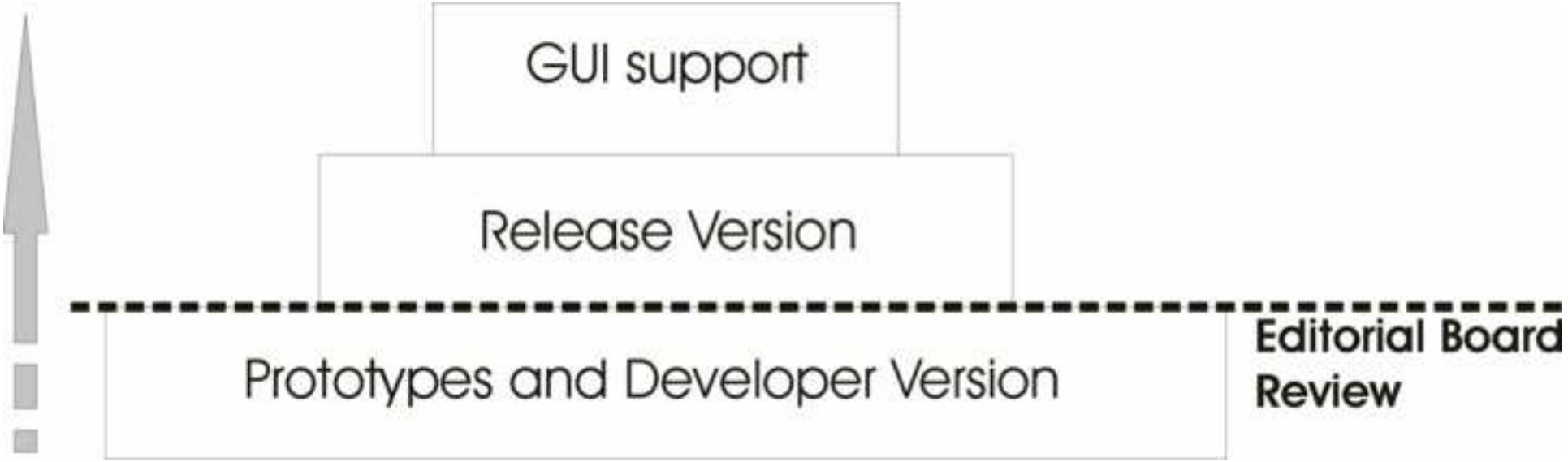

GUI support
Release Version
Prototypes and Developer Version
Editorial Board
Review

Figure 3

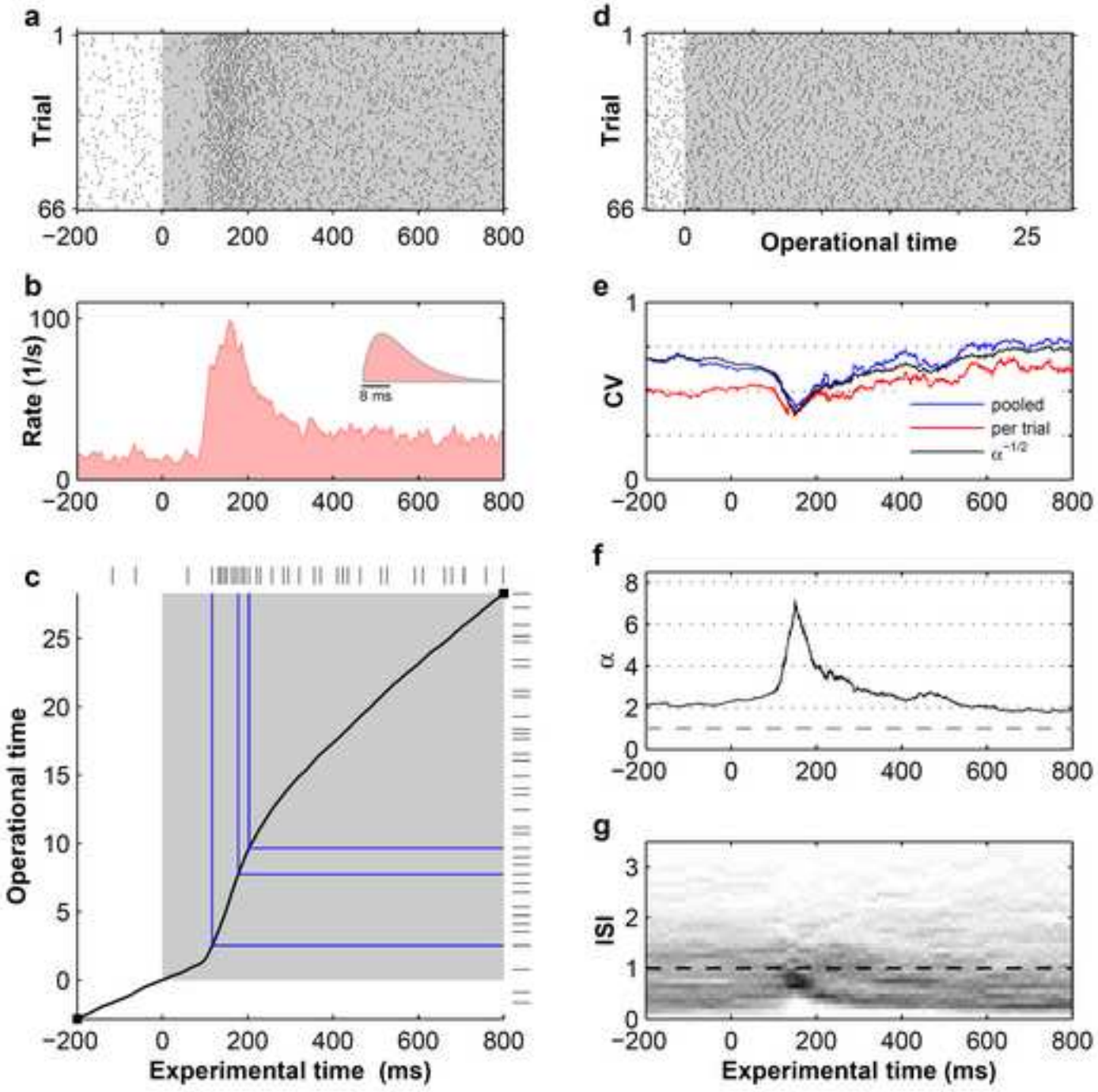
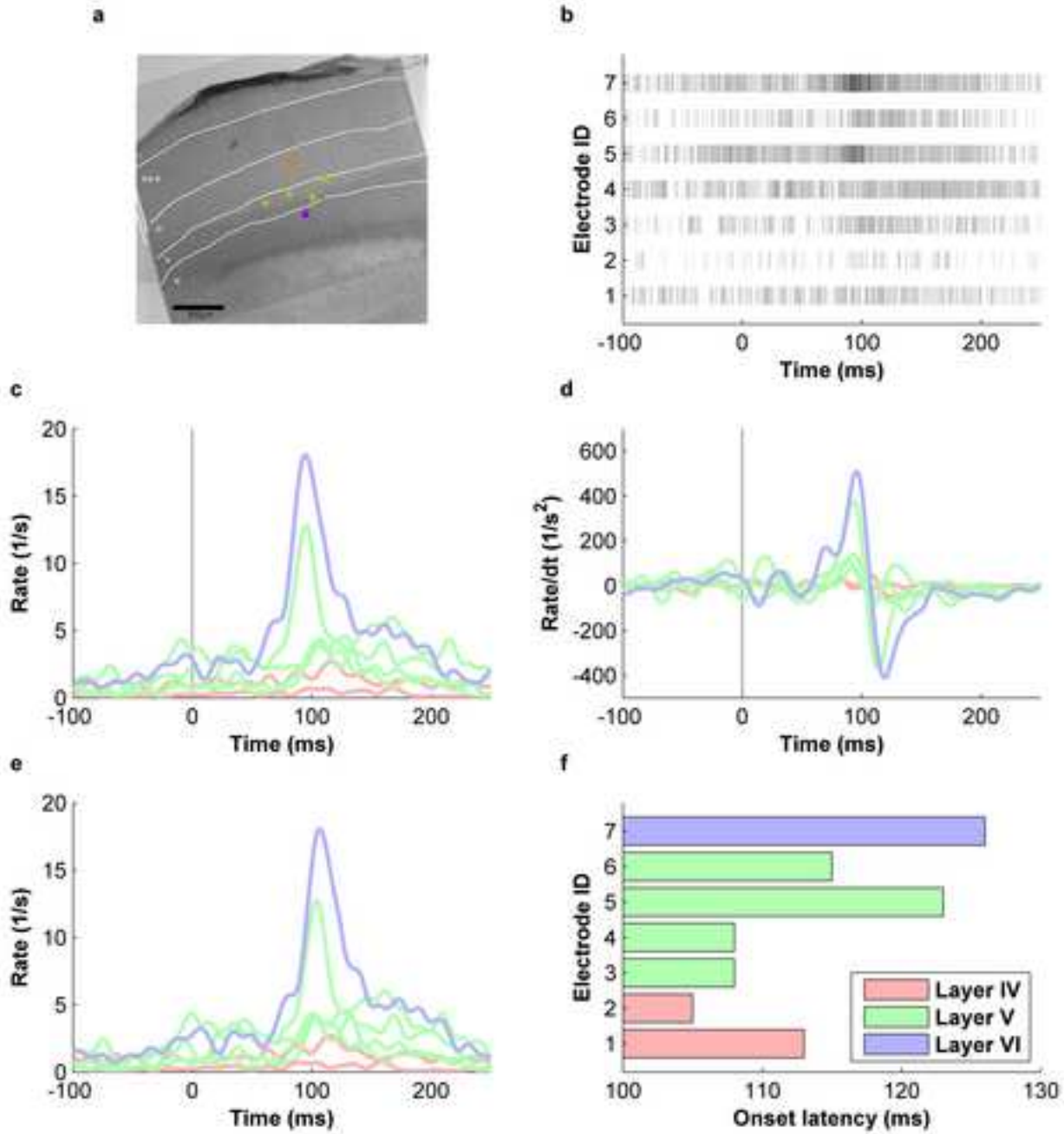

a
Trial
1
66
−200
0
200
400
600
800
b
Rate (1/s)
100
0
8 ms
c
Operational time
Experimental time (ms)
d
Operational time
25
e
CV
pooled
per trial
f
α
g
ISI
Experimental time (ms)

Figure 4


a
b
Electrode ID
Time (ms)
c
Rate (1/s)
Time (ms)
d
Rate/dt (1/s²)
Time (ms)
e
Rate (1/s)
Time (ms)
f
Electrode ID
Layer IV
Layer V
Layer VI
Onset latency (ms)

FIND Toolbox Demo

A Matlab generated report of the functions used in the present Manuscript

Please note:

- The plots for the analysis of the latency in the response onset for different layers are shown in a simple “unpolished” way.
- The plots for the time – warp example are generated as shown in the Manuscript – thus somewhat lengthy plotting routines are shown, esp. to obtain very nice graphs.
- This does not infer in any way with the functionality of the FIND – Toolbox.

DEMO: FIND-Toolbox, illustrative examples for the Article

```
This script is designed to demonstrate some of the abilities of
the FIND Toolbox.
```

This matlab script is a supplement to the follwing article

```
 Meier R, Egert U, Aertsen A, Nawrot M (2008)
 FIND – a unified framework for neural data analysis

to appear in the journal "Neural Networks".

Two examples will be shown:
  a) Realignement of firing rate onset obtained in different layers of
     primary visual cortex upon full field visual stiumulation (N=37
trials)
     This function illustrates the order in which a stiumulus response
     appears in different cortical layers.

  b) Odor-response dynamics of spiking irregularity in a mushroom-body
     extrinsic neuron of the honeybee. The provided data set comprises
spike times of one
     extracellular recorded unit from the alpha lobe of the mushroom body
in the
     honeybee brain in response to the repeated stimulation (N=66 trials)
with
     one and the same odorant (peppermint).

Futher information, source, the full FIND-Toolbox and data can be found at
<http://find.bccn.uni-freiburg.de>

Contact: Ralph Meier, BCCN Freiburg
         <mailto:meier@biologie.uni-freiburg.de meier@biologie.uni-
freiburg.de>

      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Precautions, clean up Matlab:

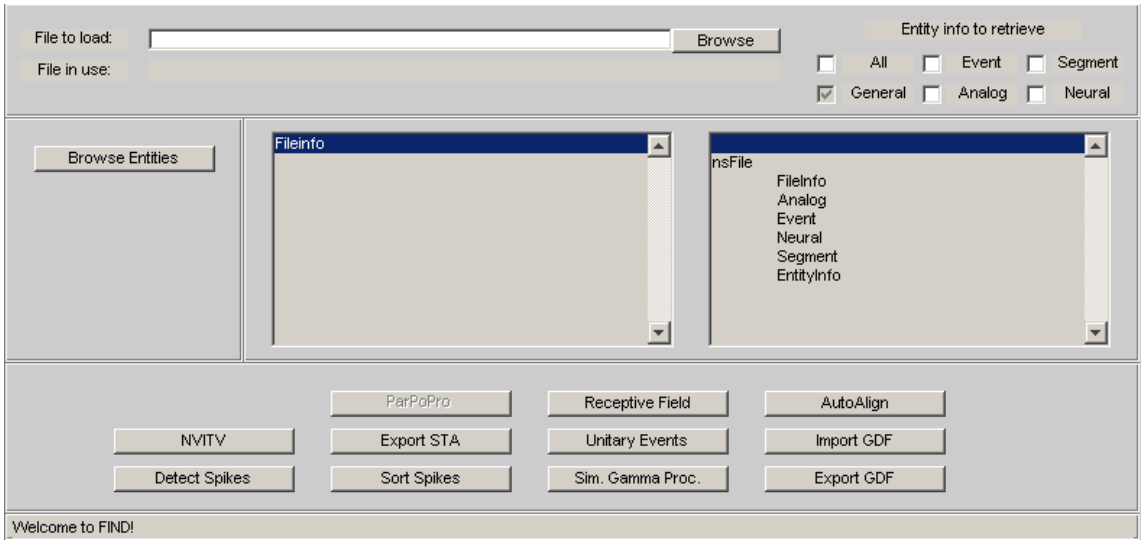
```
close all; clear all; clear global; clear functions;
```

Start the toolbox, initialize Variables and Display Parameters:

Show the main Graphical User Interface: Here most of the methods available in FIND can be called

```
FIND_GUI;
```

```
FIND init: Creating global nsFile variable in base workspace.
```



AutoAlign Example: load prepared dataset

Loads the prepared data for the rate function alignment example.

To avoid hassle in sending large spike2 files we have prepared a dataset, as it would be generated with the detectSpikes function. The analog raw data was discarded afterwards. For a tutorial on the FIND_loader see: http://find.bccn.uni-freiburg.de/?n=Docu.Tutorial

```
load([pwd filesep 'ExampleData' filesep 'DataAutoAlign.mat']);
```

Display the Data Structure

nsFile is the top level, there all information available is organized and stored.

```
whos('nsFile');
disp(nsFile);
disp(nsFile.Neural);
disp(nsFile.Event);
```

```
  Name        Size              Bytes  Class     Attributes

  nsFile      1x1             146044  struct    global

    FileInfo: [1x1 struct]
      Analog: [1x1 struct]
       Event: [1x8 struct]
      Neural: [1x1 struct]
     Segment: [1x1 struct]
```

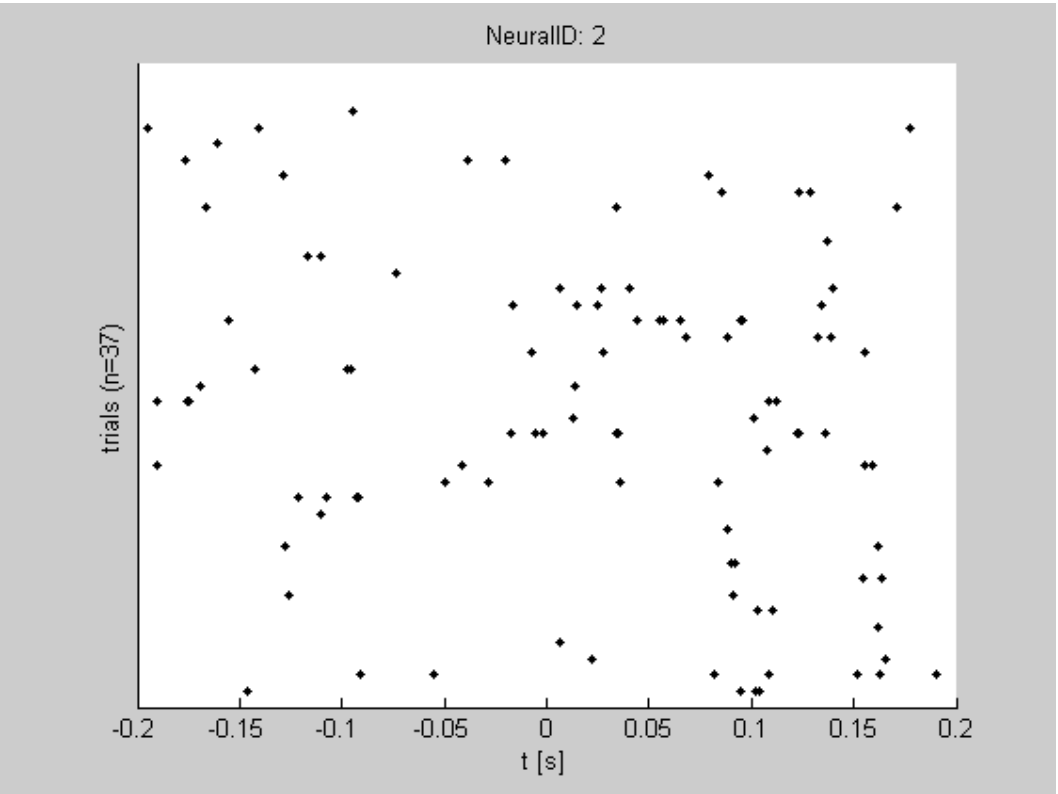
```
        Data: {1x8 cell}
    EntityID: 8

1x8 struct array with fields:
    TimeStamp
    Data
    DataSize
    Info
```

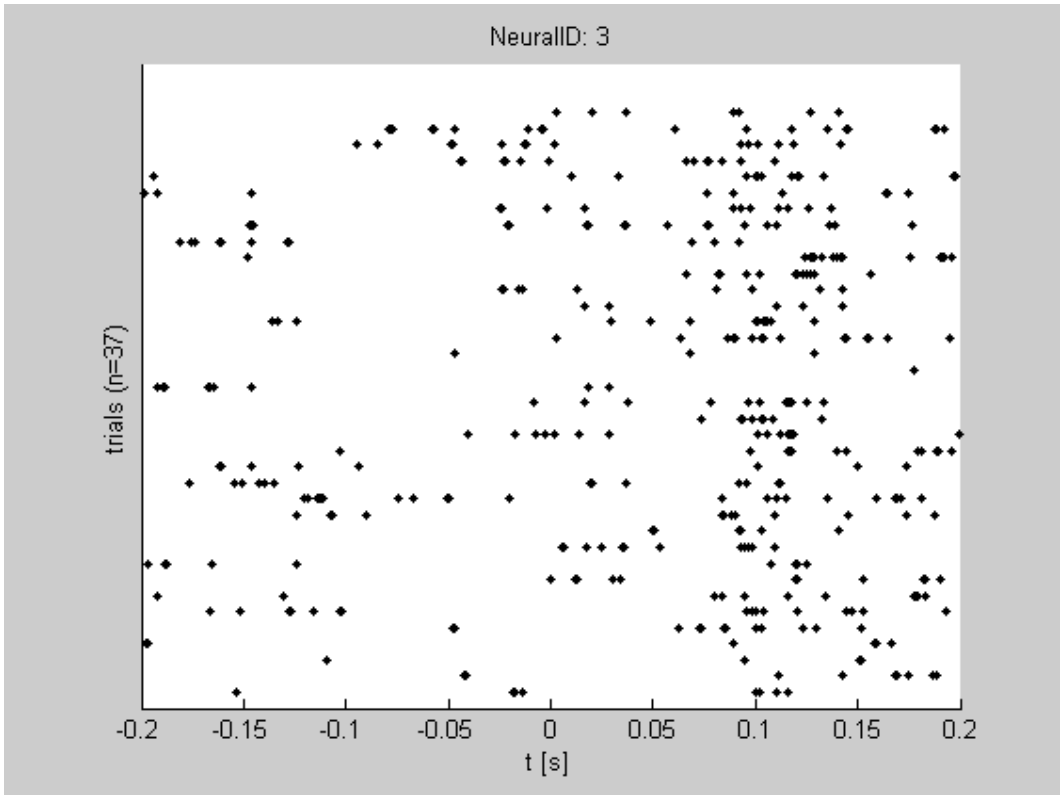
Raster Plots to illustate the Spike-Data used for the rateAlignment

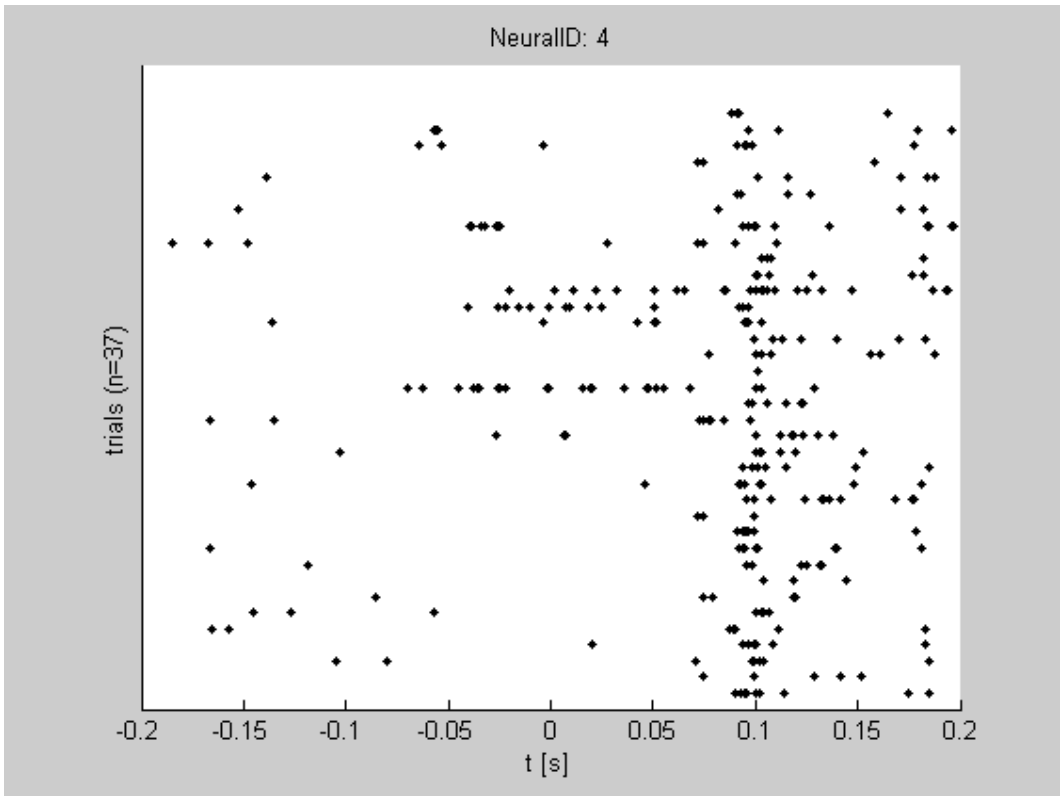
show raster plots for 3 Units

```
hh=rasterPlot('neuralIDs',[2 3 4],'eventID',1,'preTrigger',.2,'postTrigger',.2);
```

```
TRIALSPACING =

    0.1000

elc =

     1
```



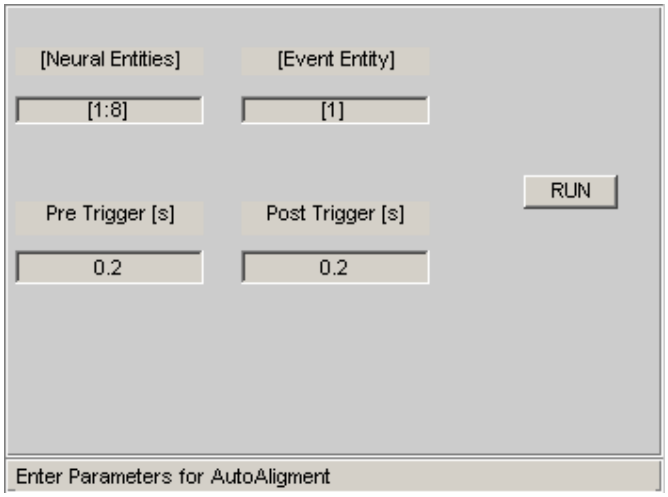

NeuralID: 3
trials (n=37)
-0.2
-0.15
-0.1
-0.05
0
0.05
0.1
0.15
0.2
t [s]


NeuralID: 4
trials (n=37)
-0.2
-0.15
-0.1
-0.05
0
0.05
0.1
0.15
0.2
t [s]

Show the graphical user interface for the autoAlign Tool

the alignement can also be exectued from the GUI - it will run the autoAlign function by generating the Parameter Value Pair calls.

```
autoAlignGUI;
```

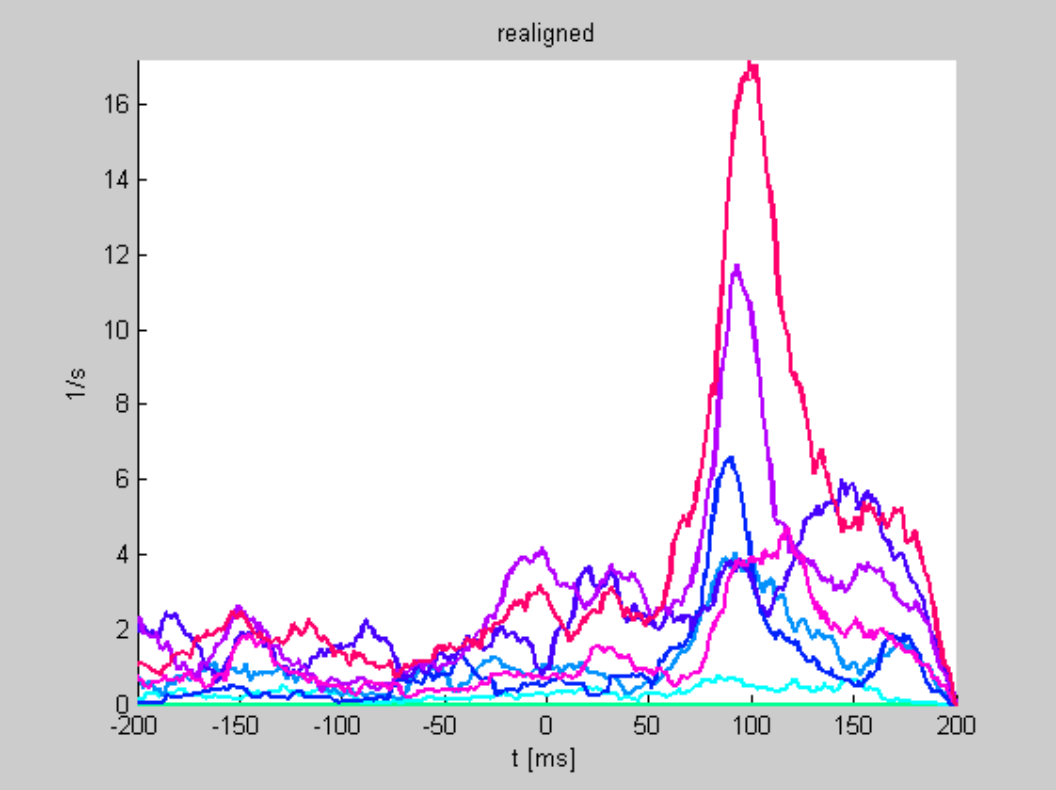


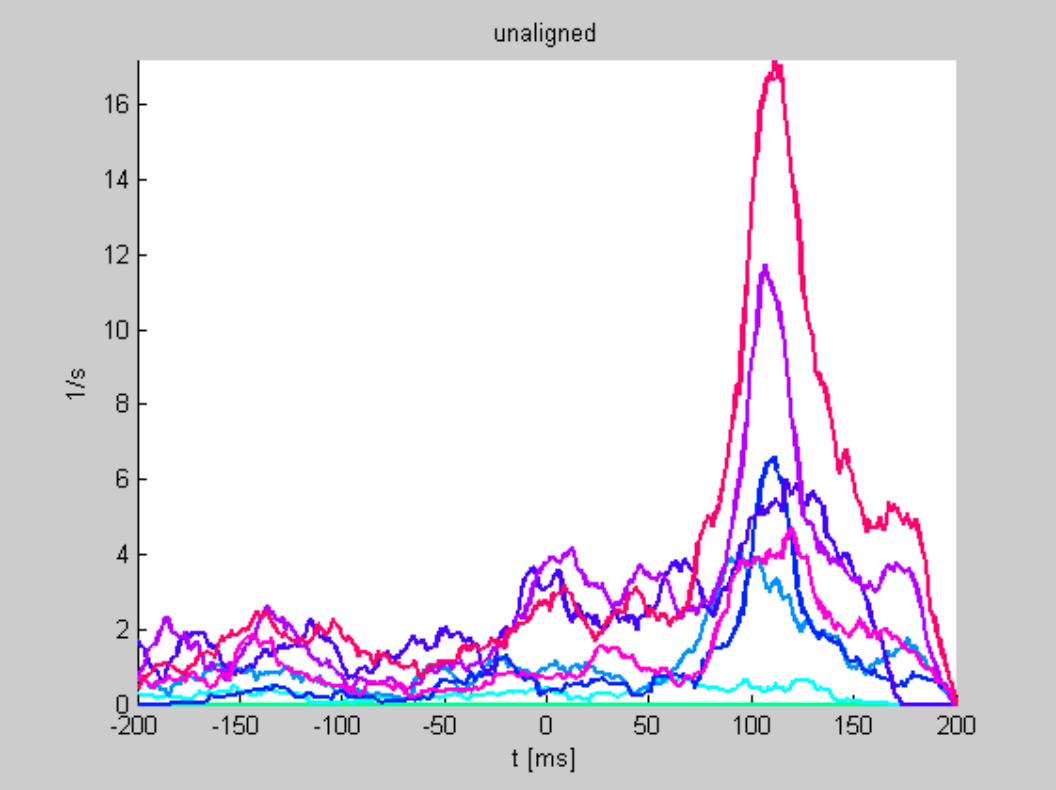
Run the Autoalign Procedure from the script

Illustrate the synonymous call of the core function autoAlign Could have been done using the GUI, which will essentially create this call.

```
shifts=autoAlign('neuralIDs',[2:8],'eventID',1,'preTrigger',.2,'postTrigger',.2);
```

```
TIME_BIN_MS =

     1


elc =

     1

========== Iteration: 1 ====================
------- LATENCIES -------
540  510  493  539  502  511  502
------- SHIFTS -------
-26   4  21 -25  12   3  12
============================
========== Iteration: 2 ====================
------- LATENCIES -------
513  516  515  517  513  516  515
------- SHIFTS -------
2 -1  0 -2  2 -1  0
============================
```





Report the time-shifts obtained by realingment

```
for ii=1:7
    disp(['Channel ' num2str(ii+1) ' was shifted by: ' 09 num2str(shifts(ii)) 09
'ms.']);
```

```
end
```

```
Channel 2 was shifted by:     -24     ms.
Channel 3 was shifted by:     3       ms.
Channel 4 was shifted by:     21      ms.
Channel 5 was shifted by:     -27     ms.
Channel 6 was shifted by:     14      ms.
Channel 7 was shifted by:     2       ms.
Channel 8 was shifted by:     12      ms.
```

End of AutoAlign Example.

```
disp('Alignement example finished.')
```

```
Alignement example finished.
```

Start the TimeWarp Example:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

disp('Example Analysis b): The bee data');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% BeeExampleAnalysis
%
% This matlab script is a supplement to the follwing article
%
%   Meier R, Egert U, Aertsen A, Nawrot M (2008)
%   FIND - a unified framework for neural data analysis
%
% to appear in the journal "Neural Networks". This script performs all
% steps of analysis needed to acomplish the results in section
% "Odor-response dynamics of spiking irregularity in a mushroom-body
% extrinsic neuron of the honeybee" and finally constructs Figure 3 of the
% manuscript. The provided data set comprises spike times of one
% extracellular recorded unit from the alpha lobe of the mushroom body in the
% honeybee brain in response to the repeated stimulation (N=66 trials) with
% one and the same odorant (peppermint). This data comprises is a subset of a
% the full data set with repeated stimulations of various odors. More
% detailed information on the experimental setup is provided in the
% mansucript.
%
% The experiment was designed by Martin Strube, Randolf Menzel
% and Martin Nawrot. Recordings were performed by Martin Strube in the lab of
% Randolf Menzel, Institute of Biology - Neurobiology, Freie Universitaet
% Berlin, Germany: http://www.neurobiologie.fu-berlin.de/
%
% References
%
% The analysis has been previously published in abstract form including an
% application to data from the auditory nerve of the locust:
%
% [1] Nawrot MP, Benda J (2006) Two methods for time-resolved inter-spike
% interval analysis. Berlin Neuroforum 2006: Abstr. No. 62
%
% Details on the method used to estimate the coefficient of variation and
% for correction of the sample bias in the time-resolved interval
% distribution please refer to:
%
% [2] Nawrot MP, Boucsein C, Rodriguez-Molina V, Riehle A, Aertsen A, Rotter S (2007)
% Measurement of variability dynamics in cortical spike trains. J Neurosci Meth,
% doi:10.1016/j.jneumeth.2007.10.013
%
%
% FIND function calls
%
% - makeKernel
% - optimizeKernelWidth
% - unWarpTime
% - warpTime
%
% MATLAB toolbox function calls
%
% - gamfit in STATISTICS Toolbox
%   -> used for fitting a gamma distribution to the empiric
%      inter-spike interval distribution
%
%
% Analysis and figure for MS on FIND toolbox
% To Do : check whether inclusion of double spikes in CV is correct!
```

```
%
% History
%
% (2) Feb 20, 2008 Martin Nawrot & Ralph Meier
%     - Adapted for publication as supplement to the Article
%       Meier et al. 2008 to appear in Neural Networks
%     - tested with MATLAB 7.1
%     - data set now available in nsFile struct
% (1) Oct 23, 2007 corrected : CV = 1/sqrt(alpha)
% (0) Oct 10, 2007 nawrot@neurobiologie.fu-berlin.de
%
% (-1) June 2005 Martin Nawrot
% based on original script: 'LocustBeeExampleMethodA' for the Berlin NeuroForum (BNF)
% Meeting 2006 in Liebenwalde, Germany ([1]) Figures A-D
%
%  nawrot@neurobiologie.fu-berlin.de
```

Example Analysis b): The bee data

Clean up, Define Parameters and obtain data

```
clear

% Paramter for Visualization
PlotIntervalMS = [-200 800];

% ----------------------------------------
% I DATA and paramter of analysis
% ----------------------------------------

WindowWidthOT = 5;
KernelSigmaMS = []; % ~0.5 - 2
DataExampleFile = 'BeeExampleData'; % data in sec with 20kHz resolution
DataDirectory=[pwd filesep 'ExampleData' filesep];
% - define
%   + pre and post trigger interval for selection of spike times
%   + time resolution for binary representation of spike trains in 0/1
%     matrix
PreTimeMS=500; %
PostTimeMS=1000; %1000;%PostTimeS*1000;
TimeResolutionMS=0.5;%1; % for analysis (unwarping etc)
offsetMS = 100; % allow for kernel width
SelectTimesMS = [150 600];
SampleBinWidthOT=0.1; % for ISI histogram on operational time
StimulusDurationMS=3000;

% ----------------------------------------
% II READ DATA
% ----------------------------------------

% - load spikes, trials in succession
%   + spike times are stored in seconds
disp(['read ',DataExampleFile])
load([DataDirectory,DataExampleFile]);
s=nsFile.Neural.Data{1}*1000;   % now in ms
TriggerEvent=nsFile.Event.TimeStamp*1000;
% - assign spike times to individual trials
N=length(TriggerEvent);
for tri=1:N
    S{tri}=s(find(s > TriggerEvent(tri)-PreTimeMS & s <=
TriggerEvent(tri)+PostTimeMS))-TriggerEvent(tri);
end

% - Fill spike matrix with original spike times & transform spikes
%    to operational time
%   + construct binary spike matrix with N single trial spike trains
TMS=PreTimeMS+PostTimeMS;
PreTime=PreTimeMS/TimeResolutionMS;
T=TMS/TimeResolutionMS;
edges=(0:TimeResolutionMS:TMS);
SM=zeros(T,N);
for tri=1:N
    idx = S{tri}+PreTimeMS;
    tmp=histc(idx,edges);   % allows for more than one spike per bin, i.e. count>1
    SM(:,tri)=tmp(1:end-1);
end;

% ----------------------------------------
% III Convolute spike train to estimate RATE
% ----------------------------------------
sv=sum(SM,2);   % pooled spike train
```

read BeeExampleData

determine optimal kernel width

---------- 1. determine optimal kernel width on the basis of the pooled spike train + choose form of convolution kernel, cf. FIND function 'makeKernel' for implemented alternatives

```
KernelForm='TRI';   % triangular kernel shape
biasflag='low';     % cf. FIND function 'optimizeKernelWidth'
offset=offsetMS/TimeResolutionMS;           % in time bins
vec=[zeros(offset,1);sv;zeros(offset,1)];   % zero padding
TimeResolutionS=TimeResolutionMS/1000;
disp('calling optimizeKernelWidth')
%   + call FIND function 'optimizeKernelWidth'
%   which in turn calls FIND function 'makeKernel'
%   + The paramter ObservationInterval defines the relevant part of the input
%   spike train. The paramter offset determines the maximal half-width of
%   the kernel that can be tested withour side-effects.
[BestKernelSigma,TestedKernelSigmas] = optimizeKernelWidth('InputSpikeMatrix',vec, ...
    'TimeUnits',TimeResolutionS,'ObservationInterval',[offset length(vec)-offset], ...
    'KernelForm',KernelForm,'biasflag',biasflag);
```

```
calling optimizeKernelWidth
```

estimate trial-averaged rate

---------- 2. estimate trial-averaged rate + use the determined optimal kernel width for rate estimate + use asymmetric alpha kernel for final rate estimate

```
kernel_form='ALP';  % 'TRI' etc ..., cf. FIND function 'makeKernel'

[k,norm,median_idx]=makeKernel('form',kernel_form, ...
    'sigma',BestKernelSigma,'TimeStampResolution',TimeResolutionS);
% + perform convolution, use Matlab's 'filter' or 'conv'
% -> with 'filter' : padd sufficient number of zeros at end of input vector
% -> for 'filter' the convention is causal. The filter shape is prepared for
% use with 'filter' or 'conv' since MATLAB conv is using filter. Note: older
% versions of Matlab had different definitions for zero padding in filter
% and conv.
r=filter(k,1,[sv;zeros(length(k),1)])*norm/N;    % estimate rate, normalize to units
1/s and devide by number of trials
r=r(median_idx+1:end-(length(k)-median_idx)); % now length(r) = length(vec) and the
center of
% -> note: mass of the rate funtion is now unchanged with respect to the center of
% mass of the original spike train.
% -> we ignore the edge effects of the convolution. this is fine because we
% will not use the edge data for time warp

% + define time axis in ms for plotting
tkMS=(1:length(k))*TimeResolutionMS;
```

integrate rate and perform time transformation

---------- 3. integrate rate and perform time transformation + integrate rate funtion

```
ir=csum(r)/1000*TimeResolutionMS;
T_=ir(end)-ir(1);           % total operational time T_
t_=ir(1):T_/(T-1):ir(end);  % define operational time t_
dt2ot = T_/T;
%    + rescaled time axis (inverse transformation)
%      will be used to display analysis results in real time
t_warp = warpTime('duration',1,'amplitude',r/1000*TimeResolutionMS,'opevents',t_);
t_warp=t_warp*TimeResolutionMS;
t_warp=t_warp-PreTimeMS;
%    + subtract trigger time (start of stimulus)
ir_offset=ir(PreTimeMS/TimeResolutionMS);
ir=ir-ir_offset;
t_=t_-ir_offset;
%   + define time axes interval for plotting
tmp=find(t_warp<PlotIntervalMS(1));
PlotIntervalOT(1)=[t_(tmp(end))];
```

```
tmp=find(t_warp<PlotIntervalMS(2));
PlotIntervalOT(2)=[t_(tmp(end))];
clear tmp
%   + time vector for spike trains (experimental time)
ts=(1:T);
tsMS=ts*TimeResolutionMS-PreTimeMS;
```

IV Unwarp spike trains and estimate CV

---------- IV Unwarp spike trains and estimate CV ----------

```
USM=unWarpTime('amplitude',r,'events',SM);
```

collect ISIs and estimate CV^2

---------- 2. collect ISIs and estimate CV^2 + in operational time + from gamma fit of ISI distribution

```
NumberOfSpikes=sum(SM(:));
% changed Oct 2007: make sure that window is of even length
window=floor(WindowWidthOT/dt2ot/2)*2;

Npre=sum(sum(SM(1:PreTime,:)));
Npost=NumberOfSpikes-Npre;

l=size(USM,1);
[u,v]=find(USM);        % single spike entries
[u2,v2]=find(USM>1);    % more than one spike per bin

disp('estimate cv ...')
clear cv2
clear ISI
all_isi=[];
% + loop across windows in time (sliding window)
for ni=1:l-window;
    isi=[];
    % + determine indices of all spikes within current window
    idx=find(u>=ni & u<ni+window);    % only spikes within observation interval
    % + determine indices of zero intervals (more than one spike per time
    %   bin). If the time resolution of spike matrix is used small enough
    %   (i.e. <= 1ms) then there should be no count > 1 in experimental
    %   data. However, in point process simulations, e.g. of a Poisson
    %   process the probability of counting more than one spike per bin
    %   is P>0 for non-zero intensity.
    tmp=find(u2>=ni & u2<ni+window);
    clear cv_tr cv2_tr;
    % + loop across trials
    for ti=1:N;
        idx2=find(v(idx)==ti);
        tmp2=find(v(tmp)==ti);
        if length(idx2)>1;              % min 2 spikes needed for at least one isi
            isi_tr=diff(u(idx(idx2)));  % isis within observation interval in trial ti
            % deprived
            % if find(USM(u(idx(idx2)))>1);
            %     disp('help');
            %     return
            % end
            % + collect isis from all trials; for 'double spikes' include
            % interval of length zero
            %            isi=[isi;isi_tr;zeros(sum(USM(u2(tmp(tmp2)),v2(tmp(tmp2)))-
1),1)];
            isi=[isi;isi_tr];
            % + compute per trial cv and cv^2
            if length(idx2)>2   % require at least 2 intervals to compute variance
                cv2_tr(ti) = var(isi_tr)/(mean(isi_tr)^2);
                cv_tr(ti)=sqrt(cv2_tr(ti));
            else
                cv2_tr(ti)=NaN;
                cv_tr(ti)=NaN;
            end
        else
            cv2_tr(ti)=NaN;
            cv_tr(ti)=NaN;
        end;
    end;
    % + determine indices of zero intervals (more than one spike per time
    %   bin). If the time resolution of spike matrix is used small enough
    %   (i.e. <= 1ms) then there should be no count > 1 in experimental
```

```
        %   data. However, in point process simulations, e.g. of a Poisson
        %   process the probability of counting more than one spike per bin
        %   is P>0 for non-zero intensity.
        %tmp=find(u2>=ni & u2<ni+window);
        % CHECK THIS FOR CORRECTNESS - Feb 2007 !!!
        %isi=[isi;ones(sum(USM(u2(tmp),v2(tmp))-1),1)];
        % + fit gamma distribution to empiric distribution of isi
        param = gamfit(isi);
        alpha(ni)=param(1);
        % + keep isis for this window
        ISI{ni}=isi;
        % + normaliz isis by mean interval
        ISIot{ni}=ISI{ni}/mean(ISI{ni});
        % + pool isis across windows
        all_isi=[all_isi;isi];
        % + compute CV on operational time
        if ~isempty(ISIot{ni})
            cv2(ni)=var(ISIot{ni})./(mean(ISIot{ni})^2);
        else
            cv2(ni)=NaN;
        end

        % + compute average per trial CV and averaged across trials
        cv2_trial(ni) = mean(cv2_tr(find(~isnan(cv2_tr))));
        cv_trial(ni) = mean(cv_tr(find(~isnan(cv_tr))));
    end
    MeanInterval=mean(all_isi);

    % deprived because we now use rewarped time scale
    % % - time vector for cv2
    % tcv=tOT;
    % tcv=tcv(ceil(window/2):ceil(window/2)+length(cv2));
```

estimate cv ...

Construct time-resolved ISI distribution

---------- V Construct time-resolved ISI distribution ----------

```
    disp('construct time resolved isi distribution')
    bin=SampleBinWidthOT;%0.05;
    maxbin=WindowWidthOT; % no longer intervals can be observed !
    if mod(WindowWidthOT,bin)==0
        maxbin=maxbin-bin; % otherwise: zero correction factor !
    end
    binvec=0:bin:maxbin;
    correction_factor=1./(WindowWidthOT - binvec);
    clear H H_C
    for ni=1:1-window;
        % A: histogram methods
        if~isempty(ISI{ni});
            H(:,ni)=histc(ISIot{ni},binvec)';
        else
            H(:,ni)=zeros(length(binvec),1);
        end
        % B : kernel convolution: to be implemented ...
    end

    % + Compute corrected distribution (see [1]):
    % -> multiply with dafctor 1 / (T - x)
    %    where : T = window width / x = ISI length on operational time
    %    -> this estimation is noisy for large intervals, could possibly
    %       be improved by weighing individual intervals and performing a
    %       kernel estimation
    H_C=H .* repmat(correction_factor',1,size(H,2));


    % + Normalize histograms
    nH=H./repmat(sum(H,1),size(H,1),1);
    nH_C=H_C./repmat(sum(H_C,1),size(H_C,1),1);

    MarginalDistribution=sum(H,2);
    MarginalDistribution_C=sum(H_C,2);

    MaxOT=max(ir);
    MinOT=min(ir);
```

construct time resolved isi distribution

Plot the Figure shown in the Manuscript :

============================== FIGURE ==================== ---------- ---------- 7 panels

```
 a : original single trial spike trains in experimental time
 b : estimated rate function
 c : integrated rate function (time transformation)
 d : unwarped spike trains in operational time
 e : estimated CV
 f : estimated gamma order alpha
 g : reconstructed time-resolved interval histogram in experimental time
```

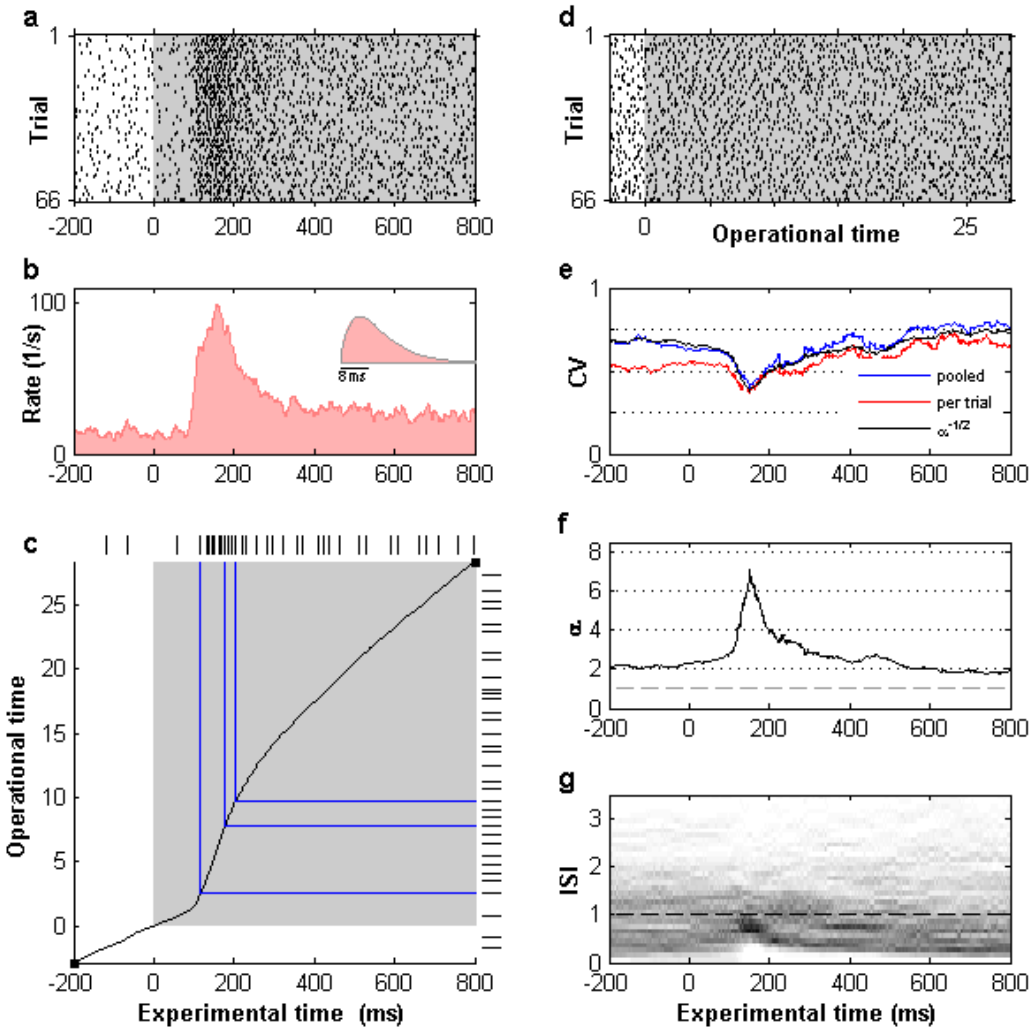
```
% - color definitions
grey1=[.6 .6 .6];
grey2=[.7 .7 .7];
rose =[1 .7 .7];
bleue=[.8 .8 .8];

% - open figure and set figur properties
figure
px=17; % width in cm
py=17; % heigth in cm
fac=py/px;
pos=[0 8 px py];
paperpos=[2 1.5 px py];

set(gcf,'papertype','a4letter') % A4 paper type for Europe
set(gcf,'color','w')
set(gcf,'units','cent')
set(gcf,'paperunits','cent')
set(gcf,'pos',pos)
set(gcf,'paperpos',paperpos)

% - define axes sizes
xwidth1cm=6;
xwidth1=xwidth1cm/px;
xwidth2cm=3;
xwidth2=xwidth2cm/px;

ywidth1cm=2.5;
ywidth1=ywidth1cm/py;
ywidth2cm=0.5;
ywidth2=ywidth2cm/py;
ywidth3cm=2;
ywidth3=ywidth3cm/py;

ax0cm=1.5;
ax0=ax0cm/px;
ay0cm=1;
ay0=ay0cm/py;

dxcm=2;
dx=dxcm/px;
dycm=1.3;
dy=dycm/py;

% - define axes positions
apos=[ax0 1-(ywidth1+dy) xwidth1 ywidth1];
bpos=[ax0 1-2*(ywidth1+dy) xwidth1 ywidth1];
bpos2=[ax0+xwidth1*(2/3) 1-2*(ywidth1+dy)+ywidth1*(2/3-1/7) xwidth1/3 ywidth1/3];

dpos=[ax0+xwidth1+dx 1-(ywidth1+dy) xwidth1 ywidth1];
epos=[ax0+xwidth1+dx 1-2*(ywidth1+dy) xwidth1 ywidth1];
fpos=[ax0+xwidth1+dx 1-3*(ywidth1+dy) xwidth1 ywidth1];
gpos=[ax0+xwidth1+dx 1-4*(ywidth1+dy) xwidth1 ywidth1];

cpos=[ax0 1-4*(ywidth1+dy) xwidth1 xwidth1*px/py];
cpos2=[ax0 1-4*(ywidth1+dy)+xwidth1*px/py xwidth1 ywidth2];
cpos3=[ax0+xwidth1 1-4*(ywidth1+dy) ywidth2*py/px xwidth1*px/py];

% ==== Start with Panels ====

% A - ORIGINAL SPIKE TRAIN --------------------
[oidx,oj]=find(SM);
oi=oidx*TimeResolutionMS-PreTimeMS;
axes
axlim=[PlotIntervalMS];
aylim=[0 min(N+0.5,100.5)];
set(gca,'linewidth',.5)
```

```
set(gca,'fontsize',10)
set(gca,'pos',apos)
set(gca,'vis','on')
set(gca,'box','on')
set(gca,'ydir','reverse')
set(gca,'tickdir','out')
set(gca,'xlim',axlim)
set(gca,'ylim',aylim)
set(gca,'ytick',[1 N]);
ylabel('Trial','fonts',10,'fontw','b','vert','top')
hold on
% + gray patch to indicate stimulus
if exist('StimulusDurationMS')
    ps=patch([0 StimulusDurationMS StimulusDurationMS 0],[aylim(1) aylim
aylim(2)],'c');
    set(ps,'facec',bleue,'edgec','none')
end
% + raster diagram, represent each spike by a vertical tick
line([oi oi]',[oj-.35 oj+.35]','color','k','linew',.25);
set(gca,'layer','top')

% B - RATE ESTIMATE --------------------
axes
set(gca,'color','none')
set(gca,'linewidth',.5)
set(gca,'fontsize',10)
set(gca,'pos',bpos)
set(gca,'box','on')
set(gca,'tickdir','in')
set(gca,'xlim',axlim)
set(gca,'ylim',[0 ceil(max(r)/50)*50+10])
set(gca,'ytick',[0 100]);
ylabel('Rate (1/s)','fonts',10,'fontw','b','vert','top')
hold on
pa=patch([tsMS(1) tsMS tsMS(end)],[0;r;0],'c');
set(pa,'facec',rose,'edgec',[1 .5 .5])
set(gca,'layer','top')

% + inset shows kernel form and scale bar in ms
axes
set(gca,'pos',[bpos2])
set(gca,'vis','off')
set(gca,'xlim',[0 BestKernelSigma*1000*5])
hold on
pb=patch([tkMS(1) tkMS tkMS(end)],[0,k,0]+.1*max(k),'c');
set(pb,'facec',rose,'edgec',[.6 .6 .6]);
line([0 8],[0 0],'linew',1,'color','k');
text(4,0,'8 ms','vert','top','horiz','center','fonts',6)

% C - WARP --------------------
idx1=max(find(tsMS<axlim(1)))+1;
idx2=max(find(tsMS<axlim(2)));
cylim=[ir(idx1) ir(idx2)];
axes
set(gca,'color','none')
set(gca,'linewidth',.5)
set(gca,'fontsize',10)
set(gca,'pos',cpos)
set(gca,'box','off')
set(gca,'tickdir','in')
set(gca,'xlim',axlim)
set(gca,'ylim',cylim)
xlabel('Experimental time  (ms)','fonts',10,'fontw','b')
ylabel('Operational time','fonts',10,'fontw','b')
hold on
% + gray patch indicates stimulus presentation
if exist('StimulusDurationMS')
    tmp=max(find(t_warp<StimulusDurationMS));
    StimulusDurationOT=t_(tmp);
    ps=patch([0 StimulusDurationMS StimulusDurationMS 0],[0 0 StimulusDurationOT
StimulusDurationOT],'c');
    set(ps,'facec',bleue,'edgec','none')
end
% + integrated rate function
plot(tsMS,ir,'color','k','linew',1);
% + mark end points
plot(axlim(1),cylim(1),'s','markerfacec','k','markeredgec','k','markers',4)
plot(axlim(2),cylim(2),'s','markerfacec','k','markeredgec','k','markers',4)

% + plot 3 blue lines to indicate transformation of 3 example spikes
idx=find(oj==1);
jdx=ceil(oidx(idx(7)));
line([oi(idx(7)) oi(idx(7)) axlim(2)],[cylim(2) ir(jdx) ir(jdx)])
jdx=ceil(oidx(idx(14)));
line([oi(idx(14)) oi(idx(14)) axlim(2)],[cylim(2) ir(jdx) ir(jdx)])
jdx=ceil(oidx(idx(17)));
line([oi(idx(17)) oi(idx(17)) axlim(2)],[cylim(2) ir(jdx) ir(jdx)])

% + horizontal spike train in real time
```

```
axes
set(gca,'color','none')
set(gca,'linewidth',.5)
set(gca,'fontsize',10)
set(gca,'pos',cpos2)
set(gca,'vis','off')
set(gca,'xlim',axlim)
set(gca,'ylim',[.5 1.5]);
hold
line([oi(idx) oi(idx)]',[1-.25 1+.25]','color','k','linew',.25);

% vertical spike train in operational time
axes
set(gca,'color','none')
set(gca,'linewidth',.5)
set(gca,'fontsize',10)
set(gca,'pos',cpos3)
set(gca,'vis','off')
set(gca,'xlim',[.5 1.5])
set(gca,'ylim',cylim);
hold on
sot=(u(idx)/T)*(max(ir)-min(ir))+ir(1);
line([1-.25 1+.25]',[ir(oidx(idx)) ir(oidx(idx))]','color','k','linew',.25);

% D - Unwarped Spike Trains on Operational Time --------------------
dxlim=PlotIntervalOT;
dylim=[0 min(N+0.5,100.5)];
axes
set(gca,'linewidth',.5)
set(gca,'fontsize',10)
set(gca,'pos',dpos)
set(gca,'ydir','reverse')
set(gca,'vis','on')
set(gca,'box','on')
set(gca,'tickdir','out')
set(gca,'xlim',dxlim)
set(gca,'ylim',dylim)
set(gca,'xtick',[-5:5:30]);
set(gca,'xtickl',{'' '0','','','','','25',''});
set(gca,'ytick',[1 N]);
xlabel('Operational time','fonts',10,'fontw','b','vert','bottom')
ylabel('Trial','fonts',10,'fontw','b','vert','top')
hold on
% + gray patch indicates stimulus presentation
if exist('StimulusDurationMS')
    tmp=max(find(t_warp<StimulusDurationMS));
    StimulusDurationOT=t_(tmp);
    ps=patch([0 StimulusDurationOT StimulusDurationOT 0],[dylim(1) dylim
dylim(2)],'c');
    set(ps,'facec',bleue,'edgec','none')
end
line([dxlim,dxlim(2)],[0,aylim],'linew',.5,'color','k')
% + define spikes on operational time
uot=u*dt2ot-ir_offset;
% + raster plot spike trains in operational time
line([uot uot]',[v-.35 v+.35]','color','k','linew',.25);
set(gca,'layer','top')

% E - CV time-resolved ---------------------------------
exlim=axlim;
axes
set(gca,'linewidth',.5)
set(gca,'fontsize',10)
set(gca,'pos',epos)
set(gca,'vis','on')
set(gca,'box','on')
set(gca,'tickdir','in')
set(gca,'ylim',[0 1])
set(gca,'ytick',[0:.25:1])
set(gca,'ytickl',{'0','','','','1'})
set(gca,'xlim',exlim)
set(gca,'ygrid','on')
ylabel('CV','fonts',10,'fontw','b','vert','middle')
hold on
w_half=window/2;
p1=plot(t_warp(w_half+1:end-w_half),sqrt(cv2),'linew',.5,'color','b');
p2=plot(t_warp(w_half+1:end-w_half),cv_trial,'linew',.5,'color','r');
p3=plot(t_warp(w_half+1:end-w_half),1./sqrt(alpha),'linew',.5,'color','k');
lg=legend([p1,p2,p3],{'pooled','per trial','\alpha^{-1/2}'});
set(lg,'fonts',7);
set(lg,'location','southeast');
set(lg,'ycolor','w')
set(lg,'xcolor','w')

% F - gamma order alpha --------------------------------
fxlim=axlim;
fylim=[0 ceil(max(alpha))+.5];
axes
set(gca,'linewidth',.5)
```

```
set(gca,'fontsize',10)
set(gca,'pos',fpos)
set(gca,'vis','on')
set(gca,'box','on')
set(gca,'tickdir','in')
set(gca,'xlim',fxlim)
set(gca,'ylim',fylim)
set(gca,'ygrid','on')
ylabel('\alpha','fonts',10,'fontw','b','vert','base')
hold on
line(axlim,[1 1],'color',[.6 .6 .6],'linew',1,'linest','--')
plot(t_warp(w_half+1:end-w_half),alpha,'color','k','linew',.5)

% G - distribution time-resolved ----------------------------------
gxlim=axlim;
gylim=[0 3.5]
axes
set(gca,'linewidth',.5)
set(gca,'fontsize',10)
set(gca,'pos',gpos)
set(gca,'vis','on')
set(gca,'box','on')
set(gca,'tickdir','in')
set(gca,'ylim',gylim)
set(gca,'xlim',gxlim)
xlabel('Experimental time (ms)','fonts',10,'fontw','b')
ylabel('ISI','fonts',10,'fontw','b','vert','base')
shading flat
hold on
colormap(flipud(gray));
% Note: pcolor is one of Matlab's 'handle with care' functions
pcolor(t_warp(w_half+1:end-w_half),binvec,nH);
shading flat
line(axlim,[1 1],'color','k','linew',1,'linest','--')
set(gca,'layer','top')
% careful: pcolor sets renderer to Zbuffer
set(gcf,'renderer','painters')

% + text axis for panel labeling
axes
set(gca,'pos',[0 0 1 1],'xlim',[0 1]','ylim',[0 1]);
set(gca,'vis','off')

text(apos(1)-ax0/2,apos(2)+apos(4),'a','fonts',12,'fontw','b','vert','bottom')
text(bpos(1)-ax0/2,bpos(2)+bpos(4),'b','fonts',12,'fontw','b','vert','bottom')
text(cpos(1)-ax0/2,cpos(2)+cpos(4),'c','fonts',12,'fontw','b','vert','bottom')
text(dpos(1)-ax0/2,dpos(2)+dpos(4),'d','fonts',12,'fontw','b','vert','bottom')
text(epos(1)-ax0/2,epos(2)+epos(4),'e','fonts',12,'fontw','b','vert','bottom')
text(fpos(1)-ax0/2,fpos(2)+fpos(4),'f','fonts',12,'fontw','b','vert','bottom')
text(gpos(1)-ax0/2,gpos(2)+gpos(4),'g','fonts',12,'fontw','b','vert','bottom')


return
```

```
Current plot held

gylim =

         0    3.5000
```



Finished Analysis -

Cleaning up.

```
close all; clear all;


% + for print to file options see 'print' command
% print -depsc2 BeeExampleAnalysis.eps
% print -dpdf BeeExampleAnalysis.pdf
```