Published in final edited form as:

J Neurosci Methods. 2007 September 15; 165(1): 38–48.

MatOFF: A Tool For Analyzing Behaviorally-Complex Neurophysiological Experiments

Aldo Genovesio and Andrew R. Mitz

Laboratory of Systems Neuroscience, National Institute of Mental Health

Abstract

The simple operant conditioning originally used in behavioral neurophysiology 30 years ago has given way to complex and sophisticated behavioral paradigms; so much so, that early general purpose programs for analyzing neurophysiological data are ill-suited for complex experiments. The trend has been to develop custom software for each class of experiment, but custom software can have serious drawbacks. We describe here a general purpose software tool for behavioral and electrophysiological studies, called MatOFF, that is especially suited for processing neurophysiological data gathered during the execution of complex behaviors. Written in the MATLAB programming language, MatOFF solves the problem of handling complex analysis requirements in a unique and powerful way. While other neurophysiological programs are either a loose collection of tools or append MATLAB as a post-processing step, MatOFF is an integrated environment that supports MATLAB scripting within the event search engine safely isolated in programming sandbox. The results from scripting are stored separately, but in parallel with the raw data, and thus available to all subsequent MatOFF analysis and display processing. An example from a recently published experiment shows how all the features of MatOFF work together to analyze complex experiments and mine neurophysiological data in efficient ways.

Keywords

neurophysiology; behavior; single-unit; data mining; primate; software

Introduction

Behavioral neurophysiology, the technique of recording signals from the central nervous system in alert animals, owes its success to technological advances that have been driven by the conceptual ones. These experiments require collecting a mix of time-stamped behavioral events, analog signals reflecting behavior, and time stamps of single units detected through electrodes, while a trained animal performs one or more strategically-designed tasks. Computer programs to collect and analyze data from these experiments have undergone a co-evolution with the conceptual underpinnings that have motivated the introduction of more complex and dynamic tasks (Sclabassi and Harper, 1973). Early analysis programs simply aligned trial-to-trial data to a specific behavioral event, often a sensory stimulus. Subsequent programs were well suited to experimental paradigms that focused on stationary processes recorded from animals executing highly stereotyped behaviors in experiments designed to compare a modest number of conditions, e.g., movement to one of several targets, in a blocked trial or mixed-

Correspondence should be directed to: Dr. Andrew R. Mitz, National Institutes of Health, Bldg 49/Rm B1C72 MSC 4401, Bethesda, Maryland 20892-4401, USA, phone: 301 402-5573, fax: 301 402-5441, email: arm@nih.gov

Publisher's Disclaimer: This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final citable form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

trial design. For a given condition in these experiments, behavioral performance and single-unit activity in any one trial mimicked all other trials for that same condition. Analyzing these experiments required separating out all trials for a given condition (e.g., movement goal) from the trials of the other conditions. Newer behavioral paradigms – developed to test more subtle complexities of mammalian brain function – have placed new demands on the analysis tools. Modest refinements to earlier software were sufficient to accommodate simpler non-stationary paradigms, e.g., associative learning, embedded sequence learning, reversal learning, motor skill acquisition, and pharmacological wash-in and wash-out experiments. These non-stationary paradigms tracked the time course of processes or events that evolved over many trials.

More radical software changes are required to study behavioral paradigms where the state of the subject, reward schedule, sequence of movements or other behavioral factors depend unpredictably on the animal's behavioral choices over many task trials. Some laboratories that employ highly non-stationary behavioral paradigms have abandoned general purpose analysis programs in favor of custom programs written for each new experiment. They typically use high-level general purpose programming languages that can allow relatively efficient re-use of previous code, like C, Basic, MATLAB® (The MathWorks, Inc.), or LabVIEW® (National Instruments). However, many components of the code must be re-written or extensively modified for each new analysis of data.

Rather than abandon general purpose neurophysiological analysis tools, the present paper describes a general purpose neurophysiological analysis tool, MatOFF that supports a very wide range of analysis requirements. On the low end, non-programmers can rapidly generate simple peri-event rasters and histograms for a single neuron using its graphical user interface (GUI) and built-in functions. As a subset of the built-in functions, MatOFF is a valuable tool for analyzing a purely behavioral experiment or data from acute electrophysiology. On the high end, architectural features permit the more sophisticated user to sift through a population of units recorded under complex, non-stationary behavioral conditions by mixing batch, macro, and high-level language scripting from within the framework of a single tool. Results can either be plotted in traditional ways (rasters, histograms, X-Y displays, etc.) or passed to other (e.g., statistical, graphics) tools for further processing. The examples of MatOFF presented here are from a recent demonstration of strategy selection in prefrontal cortex, a paradigm not readily amenable to analysis by the current generation of general purpose neurophysiology analysis programs. MatOFF was developed and is maintained by the Laboratory of Systems Neuroscience in the National Institute of Mental Health, Bethesda, Maryland (<http://dally.nimh.nih.gov/matoff/matoff.html>). Except for the (extensive) documentation, all components are written in MATLAB and the program is distributed freely with its source code.

The major advantages of general purpose neurophysiological analysis tools over custom programming are twofold. First, code reuse is maximized. Software components that manage data, file names, cell identifiers, and components that warehouse interim values for later processing, as well as components for delivering results (plots, statistics, output data files) do not have to be re-written or re-integrated with new software each time a new analysis program is written. Second, the likelihood of software bugs in the analysis software is reduced each time a new experiment is analyzed, rather than increased. More general purpose analysis tools are tested with each new experiment and carry the benefit of prior testing from past experiments. The reliability of code is inherently less assured when it is rewritten or changed with each new experiment.

Levels of data analysis and management

For the most complex experiments, selecting trials based on intra-trial event codes is not sufficient (see below). MatOFF has a hierarchy of tools that aid not only trial selection and

data realignment, but also aid in the selection of units for batch processing and population analyses. A data analysis for neurophysiological recording must provide tools to move smoothly from the single unit domain to the cell population domain, without obfuscating or abstracting the data beyond recognition at any point along the analysis axis. Not all levels must be visible simultaneously, but problems arising at one level must be traceable to the others. MatOFF encourages data visibility through its layered design.

Methods

A concise event search engine

As behavioral tasks and their analyses become more complex, the ability of a data analysis tool to efficiently locate, accumulate, and display the experimental trials that define an experimental condition becomes increasingly important. MatOFF was designed to provide efficient within-trial event searching through an event sequence matching engine that operates from a simple, concise search command. Each possible event in a trial is assigned an arbitrary event number (event code) by the acquisition system or data conversion program. The search criteria are defined by a sequential list of bracked event code groups, called the event code sequence. Event codes are logically ORed within brackets, and a logically ANDed between brackets. For example, the event code sequence: [5][14][8,10][51] is satisfied by the following two possible “embedded” event code sequences in the data: 5, 14, 8, 51 and 5, 14, 10, 51. Intervening events disqualify the sequence, unless those events are specifically excluded by an “ignore” list.

State-based execution engine (description-execution)

MatOFF operates in two steps: state description and execution. Most user interaction with MatOFF is to establish a state set for processing. Once this set is established, an execute command is given to process the data as described by the state set. MatOFF compares the new requirements to the previous state and makes changes as necessary. This approach has a number of major advantages over other, more immediate interactive approaches. First, most interactions with MatOFF are nearly instantaneous; long delays are relegated to the execution step. We find this division of time more consistent with the way users like to work. Second, the user is not burdened with mastering the sequence of command processing. MatOFF decides what processing steps to execute during the execution step. Third, having a separate execution step simplified the implementation of a command interpreter that seamlessly mixes GUI and command line commands. Fourth, the state system, or part of it, can be saved, restored, and edited in human-readable form as a group of commands.

The command interface is a fully mixed GUI/command line environment (Figs 1, 2). Nearly every GUI action has a command line equivalent and vice-versa. Many GUI actions generate the command lines and most commands have an equally-visible action in the GUI. This duality is achieved by keeping only a single copy of the system state, mostly in the GUI structures. Scripting is accomplished simply by writing a list of commands stored in a text file. By nesting and allowing some simple macro features, scripting can be made very general (see below). There has been no effort to make the execution engine look or feel like a programming language. There are no assignment or loop control statements.

In its simplest form, the description-execution model can be viewed as an interactive process:

- Describe (user operations)
 - select file and unit
 - set search requirements
 - describe desired graphical layout

choose data to export

Execute (computer operations)

locate trials

organize results

plot

export

This interactive process is typically used to explore the data and fine tune output formats using commands described below and in Supplementary Tables 1 and 2 online

Scripting and Batch file Layers

Scripting and batch files are layered over the basic description-execution model. Once there is a meaningful description for each experimental condition, the descriptions can be saved as one long list of commands called a “protocol”. Scripting is accomplished in MatOFF by simply executing a protocol:

Run protocol (script file)

Describe

Execute

Describe

Execute

Batch operation adds one additional level of execution. It runs protocol files and assigns string variables that are used as parameters in the protocol files:

Run batch file

Run protocol <string 1> <string 2> <string 3> ...

Run protocol <string 1> <string 2> <string 3> ...

Commands in the protocol use place holders that undergo simple macro expansion when they are encountered. For example, if string 1 is “102”, the command **sequence [10][12][%1]** will expand to **sequence [10][12][102]**. One more level of macro expansion can occur when a protocol file calls another protocol file. The layering of execution works well for analysis of neurophysiological experiments when batch files are used to gather population data for one type of analysis. Each batch file is used for a different question about the population. Each line of the batch file lists a single unit by file and name, as well as any characteristics specific to that unit (e.g., which trials are valid, which stimulus was used, the cytoarchitectonic location of the unit). The same protocol file is used each line of the batch file. That protocol file has a series of description-execution command sequences for all standard conditions (e.g., target directions), and generates either a series of plots or a text spread sheet of results. Other approaches to parsing analysis problems with MatOFF have been used, as well.

Command language

The present version of the software has 116 commands, most with multiple parameters or other variants. For descriptive purposes, the commands can be divided into the following categories:

Data conversion to convert data from different sources into MatOFF files

Data preprocessing to provide data filtering during conversion

File, unit, and channel selection to choose the data set for processing

Search criteria for accepting or rejecting behavioral trials

Options for ordering the selected trials

Graphics layout for rasters, histograms, analog data, and time scale control

Graphics layout for handling groups of graphs

Data dumps, statistics, and other output file commands

Process control for plotting and statistics

Process control for batch file and scripting

Process control for debugging and special purpose

User interface control

Utility commands for error handling, logging, directory navigation, etc.

Supplementary Table 1 online provides a list of the most often used MatOFF commands and their purposes. Supplementary Table 2 online lists the graphics display control commands.

MatOFF file optimizations, the MatOFF database

MatOFF requires that data be converted from its original format to an optimized internal MatOFF data format. (See Supplementary Note 2 online for detailed descriptions of the data files.) MatOFF has a built-in conversion tool for files created by the NIMH Cortex experimental control and data acquisition program. Cortex is free, widely used in behavioral neurophysiology, and has a simple file format. The Cortex file includes three data types: single unit (spike) data, analog data, and event data, stored in trial-by-trial records. The MatOFF conversion program not only converts these files, but also provides a range of remapping and filtering options. To simplify the task of converting other data file types (Fig 3), we have adopted the strategy of converting other file formats to the Cortex format first. MatOFF is distributed with a stand-alone tool for converting Plexon, Inc. files “.plx” into Cortex format, and we are currently developing a NeuroShare interface for converting many popular data acquisition formats to the Cortex file format (Fig 4).

The design of MatOFF data files provides both flexibility and efficiency. Cortex and MatOFF are trial oriented. Like many data acquisition systems, Cortex can record a single unit across multiple data files, and in some cases a single unit may be stored discontinuously across a group of files. Also, like many acquisition systems cortex data records may be of varying lengths. For efficiency, MatOFF creates a separate file for each data type. It then indexes all the behavioral trials in a file that holds pointers to each of the other files. The index file has a fixed record length for rapid random access to any trial in the experiment. To assure data integrity, each trial of data in a MatOFF data file has a header that must match the index entry each time it is accessed. A “unit definition file” lists which of the behavioral trials belongs to each single unit. This definition file manages discontinuous recording and recordings across multiple input files. When a file and single unit spike channel are requested by the user, MatOFF reads the entire index file entry for that spike channel into memory. (At present, MatOFF works on only one spike channel at a time.) Accessing any specific data type for any selected trial is rapid, because only the essential data type (i.e., analog, single unit, event) is accessed and records are accessed through in-memory absolute file pointers to structures within the data files. Profiling studies of MatOFF show that the most demanding task is searching through complex event sequences for matching event patterns. The file organization of MatOFF optimizes the search process by isolating event data from other data types, and feeding the event search engine through rapid access of each trial, even when trials are not contiguous.

One additional performance issue was addressed early in the development of MatOFF. Opening and reading very large data files created serious operating system performance problems under some versions of Windows. We did not want to have limits on the data file sizes, so MatOFF has its own internal file caching for the data files. Facility for advanced trial analysis: the MATLAB sandbox

What sets MatOFF apart from other neurophysiological analysis tools is its integration with MATLAB. Other neurophysiological tools, like NeuroExplorer® (Nex Technologies, Littleton, MA), use MATLAB for a post-processing step *after* the data search is complete. MatOFF integrates user-written MATLAB scripts *into the search engine* using a sandbox to protect the original data and program execution. A sandbox is a program execution environment for running untrusted programs without risk to the main software or data. A script is a file of MATLAB commands that does not begin with the FUNCTION keyword. This sandbox provides flexibility without compromising software stability or data integrity, and has the unique power to evaluate complex experimental conditions from within the data analysis program.

In understanding the role of the MATLAB sandbox, it helps to understand the original motivation for integrating MATLAB scripts within the MatOFF search engine. A given trial of recorded data will have single unit spike times, analog samples, and behavioral events. In simpler experiments all practical information about a trial can be encoded as event codes; but, in complex experimental designs, the significance or categorization of a trial can depend upon its relationship to a complex series of events and experimental conditions occurring in many other trials. The MATLAB sandbox was originally conceived as a tool for examining the past and future histories surrounding a trial, and to use that information to establish new classifications for the trial. For that reason the MATLAB scripts are called *history scripts*. Although they are used for many functions beyond evaluating historical records of events before or after the current trial, history scripts have been most essential in teasing out complex behavioral histories. Each time an event sequence matches the sequence criteria (determined by the events codes within a trial) the execution branches to a script file of MATLAB commands (the “history script”). The script has access to all data for all trials of a single unit, as well as other key information about the current trial (trial number of the matching trial, sequence that matched, etc.) through a group of functions and predefined variables. The history script is written to evaluate this information and make decisions about the current trial.

The history script obtains information about the experimental data from a set of functions and variables available to the script. A second set of functions allow the script to permanently store results. The script can define any number of “classes” (numbered 1...n) and store a separate value “class value” in that class for each trial (Fig 5). For example, the script could establish class 1 as the number of rewarded trials that occurred before the current trial by counting the number of prior rewarded trials and placing that value into class 1 for the current trial. A history script can query class values just like experimental data, so any class values determined earlier can be used for determining new ones. Thus, trials can be refined or re-classified in a meta-analysis based on earlier classifications. Class values can be the result of detailed single unit data or analog analysis, as well as events. For example, we developed a saccade analysis script that produces class values of saccade vector amplitudes and angles for each trial, as well as the start and ending times of the saccade (Supplementary Note 3 online).

All class values for all trials (called the “history data”) can be saved with the original data in the MatOFF data file structure. MatOFF stores data in a group of files with a common root name (e.g., data.analog, data.pulse, data.event), called a *MatOFF database*. The class values are stored using the same root name, but in files with “.history” and “.hindex” suffixes (Figs 3, 5). This arrangement permanently associates the history data with the original data, but does

not mix the two directly. This parallel, but separate data management is the data management component of the scripting sandbox. Each time a MatOFF database is opened, MatOFF fetches the history data along with the raw data. All further analyses of these trials have access to both the original trial data as well as to the derived trial information from previous history scripts (Fig 6). The power of this approach is immense. It allows the user to peel away the complexities of an experiment in layers and draw on all prior work at any time.

Once created by a history script, history data can be applied to further analysis in multiple ways (Fig 6). History data can be treated as a spread-sheet or database. The data can be saved to disk using a history script and retrieved by another analysis program (e.g., statistical package or spreadsheet program) for further analysis. MatOFF can use class values for organizing and plotting data. History scripts can determine which trials that match the search sequence will be included or eliminated from further analysis. Data alignment for peri-event displays can use the times stored in a class. This allows alignment of data to any measurable quantity or event, and can even include synchronized data stored by some other device during the course of an experiment. A history class can be used to mark events on time displays. Two history classes can be used to truncate analog displays to eliminate all but the key time window of each analog trace. A pair of history classes can likewise mark a time window in each trial to indicate which time bins should be included in the single unit spike histogram.

Results

MatOFF as a productive tool

MatOFF comes from a heritage of productive tools. Since its introduction in May, 2000 MatOFF has been used as the primary analysis tool for seven full-length peer-reviewed research papers from this laboratory and is used on a regular basis in other neurophysiological laboratories. Before that, PCOFF, a 16-bit DOS-based C language program with a text-based GUI and monochrome graphics contributed to 17 full-length peer-reviewed research papers. PCOFF used the description-execution model and supported scripting (protocol files). PCOFF was introduced around 1990. Prior to that, KOFF, a PDP-11 program written in Pascal, supplied much of the same basic functionality. The last version of KOFF was completed in 1985 and had 60 commands, including protocol files without macro expansion. A complete count of papers published with data analyzed by KOFF has not been tabulated, but KOFF contributed to at least as many papers as its successors.

MatOFF's evolution has been layered. The basic functions and features survived the evolutionary changes as new facilities were added. Preserving its primitive built-in functions makes MatOFF well suited to "simpler" experiments. For example, MatOFF is an excellent tool for evaluating the behavioral data of psychophysics experiments. On the flip side, MatOFF can be used for acute or *in-vitro* electrophysiological studies where the only "behaviors" are the experimental manipulations.

By providing a tool that can be used efficiently by programmers and non-programmers, alike, MatOFF has been an attractive solution for laboratories that see a regular turn-over of PHD students and post-doctoral scientists who arrive with a wide range of prior computational experience. Those without programming expertise can begin working independently with MatOFF with about an hour of training. Daily print-outs of behavioral and other data can be automated easily, so preliminary experimental results are available at the end of each recording day. As the user gets more sophisticated with MatOFF features, he or she can develop more complex analyses and displays. Typically, the analytical needs drive the learning process. History scripting supports very sophisticated analyses from within MatOFF. The MatOFF database does not limit the complexity or size of an analysis; we have used a database with over 1,500 units without problem. Still, it is common to export results from MatOFF to other

software tools, including statistical packages, spreadsheets, and databases, by judiciously parsing the analysis among the various tools. As with all programmable tools, the key to productivity is to determine how to best parse the problem.

The MATLAB sandbox as an enabling technology

With the introduction of history scripts, many tasks that were once difficult with MatOFF became easy to program. For example, the ability to measure the time between any two events under MatOFF was restricted to events in the search string. A 20-line MATLAB script generalized this ability to any events. Likewise, any class values can now be used to order rasters, vastly expanding MatOFF's capabilities. A 17-line script is now used to export analog data to almost any data format. A ~250-line script now analyzes saccade start time, end time, start and end positions, and peak velocity in a series of history classes, making new classes that can be used for marking, sorting, or exporting data. Being able to add functionality with so little regard for data file structure, unit selection, trial search strategy, trial selection, or graphical display, makes history scripting a rapid and reliable method for adding functionality as needed. Scripts can be experiment specific, or they can be general purpose. The next section describes the analysis of a demanding behavioral task, accomplished without the need to venture outside of the data analysis tool.

Use of history scripting for trial classification in a strategy task

As discussed earlier, it is sometimes impossible and often undesirable to numerically encode every possible behavioral event during data acquisition that may be required for later analysis. Furthermore, unforeseen analyses may require information beyond that encoded in each trial during the experiment. Some analyses require information about the previous trial, sometimes many previous trials. We use history scripts for both unforeseen analysis requirements and for dealing with complex intertrial events that might be unwieldy to encode. In a recent experiment (Genovesio *et al.*, 2005) we used history scripts to efficiently classify each trial into one or more critical categories. Despite using a relatively rich coding scheme during the experiment, only a minority of event codes was essential; all other event codes could have been eliminated from the original data file. The essential codes included time markers (times of stimulus presentation, eye movements, etc.) and a few other events (type of stimulus, response executed, response requested, delivery of reward). In the end, we used history scripting to identify, categorize and simplify the analysis. We have since started using history scripting to look at old experiments and test hypotheses not originally anticipated.

Here we provide an example of a complex experiment analyzed using history scripting. Two rhesus monkeys were trained to perform a saccade task (strategy task) as previously reported (Genovesio *et al.*, 2005). Briefly, in this task the monkeys first fixated a central spot; then, three potential eye-movement targets appeared: up, right and left from the center (Fig 7A). One second later a visual instruction stimulus appeared at the fixation point for a variable amount of time. Its disappearance triggered a saccadic eye movement to one of the three potential targets. The correct target for a given trial depended on executing the correct one of three strategies. A fluid reward was given depending upon both the strategy and the monkeys' proper execution.

In part of this task (Fig 7B), monkeys were required to select between a *repeat-stay* and a *change-shift* strategy when the previous trial ended with a juice reward. The repeat-stay strategy required the monkeys to choose the previous target again whenever the stimulus repeated from the previous trial (called "repeat trials"). The change-shift strategy required them to choose either of the other two other targets when the stimulus changed from the previous trial (called "change trials"). Fig 7B shows an example. The stimulus repeated from the previous trial where a rightward movement was rewarded, hence the repeat-strategy required selecting the same

target again. When the stimulus changed to one of the other two stimuli in the set, either the top or left target would be treated as a correct action, according to the change-shift strategy. A correct repeat-stay choice always produced a reward, but a correct change-shift choice did so for only one of the two possibilities, which was selected by the computer in advance of each change trial. That is, correct application of the strategy did not always lead to a reward. (Incorrect application of the strategy was always unrewarded.) If a monkey correctly shifted its response, but did not receive a reward, the following trial was guaranteed to be a *second-chance* trial. In a second-chance trial the instruction stimulus (picture) is repeated from the previous unrewarded trial and the monkey is required to choose the third possible direction (the one not chosen the past two trials). Correctly executed second-chance trials were always rewarded. This complex set of rules is actually the most efficient rule set that can be used for visuomotor associative learning when implemented with forced retrials. It is a rule set most humans and at least some monkeys adopt spontaneously during this type of learning set behavior (Wise and Murray, 1999).

History based selection of the strategies—Identifying a trial associated with one of the above strategies cannot be done simply on the basis of the events within a trial. The information from the previous trial is sufficient while the monkey completes trials without any errors. For example, a repeat-stay trial is a type of trial in which a stimulus presented in the previous trial is presented again. Repeat-stay trials can be distinguished from change-shift trials because the stimulus changes in change-shift trials. Distinguishing repeat-stay trials from second-chance trials is more subtle. With repeat-stay trials the expected response matches the previous response; in second-chance trials the requested response never matches the previous response. Thus, unambiguously identifying a repeat-stay trial requires knowing: if the stimulus has changed, if the requested response has changed, and if the prior trial was executed without error. When the monkey stopped working for a few trials and then resumed, recorded event codes were insufficient for identifying the trial type. In these and many other cases history scripting was essential for classifying trials. With the flexibility of history scripting we were able to look backwards in time to examine any number of intervening trials. Adding a history class is so easy that we implemented multiple classification schemes for the trials. Having multiple classifications allowed us to employ a “late binding” for analysis. In more than one occasion we created our standard, as well as “more conservative”, classifications in anticipation of reviews’ objections to an analysis decision. Fig 7C shows an example of a strategy cell. This cell increased its activity after stimulus presentation only in the change-shift trials but not in the repeat-stay trials

The experiment

History scripting proved essential to uncovering a property of prefrontal cortex activity that was unanticipated (Genovesio *et al.*, 2006). At the start of each repeat-stay or change-shift trial the monkey must remember which was the target (goal) selected in the previous trial. Both that previous goal and the upcoming instruction stimulus are essential for choosing the correct next goal. Figure 7D shows the same cell showed in Fig 7C, but this time with the same trials sorted according to the previous goals. This cell is both strategy and previous-goal selective. It shows a preference for trials in which either the top or the right goals had been selected previously. Second-chance trials are quite different. By the time the monkey is awaiting the instruction signal, the decision about the next goal has already been made. That decision was made at the end of the previous trial, and is thus something that must be held in memory between trials. To study how the brain distinguishes between storing the representation of an accomplished goal and storing the representation of a planned, but not yet executed, goal, we used history scripting to extract specific instances of repeat-stay, and matching instances of change-shift trials and second-chance trials.

Creating history classes—To study the neuronal representation of the previous goals we needed to classify all trials for all single units according to the goal selected in the previous trial (Fig 8). At the same time, each trial also had to be classified by other key features (e.g., the current strategy, the instruction stimulus), and the number of intervening error trials (0~3). We applied a history script that searched backwards one to four trials to find the previously selected goal and stored the results in a class as information about the current trial. That script also saved a separate class value indicating how many error trials intervened, if any. In the majority of the analysis trials following an incorrect choice or a break of fixation were excluded. We also developed classes for more exploratory measures, like the number of times prior to the current trial that the current instruction stimulus was paired with the expected response. In addition to creating classes for behavioral events, we added classes that held the duration and selected goal for each trial, calculated in a history script by analyzing event codes and the X and Y eye position data.

History scripting supports both bottom-up and top-down data analysis—History scripting lends itself to the natural flow of data analysis. Although an experiment may have a top-down design, analyzing data often requires a bottom-up approach. Even with explicit experimental questions in mind, it was easiest to construct trial types and subtypes before attacking the specific questions (Fig 8). Top-down goals helped define the intermediate results that would be the most useful. Separate history scripts were used to work at each hierarchical level, from defining trial types to defining cell populations. Each level drew from classifications extracted at the previous level.

Once the basic classes were established, further analysis was remarkably easy (Fig 9). Multistage or “layered” application of history scripting simplifies bottom-up analysis. For example, using the strategy class and the correct trial class we could select only the repeat-stay or change-shift trials rejecting the second-chance trials. For another analysis we needed to sort the correct trials according to the previous choice. We had to separate trials leading to a success from those that lead to a failure. (Although it is conceivable to design a data collection system that could buffer data enough to encode future events, we know of no existing system with this facility. Certainly, in our case, it was not possible to acquire event encodes describing the success or failure of future trials.) History scripting greatly eased the problem of selecting trials that lead to a correct repeat-stay or change-shift trial. By using the information about the previous trials in conjunction with the information about the correctness of the current trial, we assigned the value of the previous choice to the previous trial in one of two (success – failure) classes. Thus, the history for a given trial contained information about the choice and whether it leads to a successful subsequent trial.

History scripting was also essential for studying the evolution (time course) of single-unit activity associated with the representation of future goals in second-chance trials. This activity starts towards the end of the previous trial, which was a not-rewarded, but strategically correct, change-shift trial. To evaluate this time course we created a new class and assigned the value of the current goal to this class in the previous trial. This classification scheme provided an easy dichotomy between not-rewarded change-shift trials that lead to successful second chance trials and those that did not. For convenience, the same script created a class that stored the future goal, although that information could be gleaned from other trial data. In the next cycle of analysis we used these new classes to sort the not-rewarded change-shift trials by the future goal.

Discussion

One unexpected benefit of MatOFF and its history scripting is the promotion of teamwork. A single MatOFF database can be shared (copied), and any copy of a database can be updated

by sharing history scripts. Once a database is updated, any investigator can query and make use of all available classes. In practice, an initial series of history scripts are used to define classes that are essential to all subsequent analysis. This generates a common database. That database is distributed to the investigators, who then work independently. By agreement, each investigator is assigned a different range of class numbers for new classes. New classes developed by each investigator are distributed to other investigators as history scripts. Each investigator can pick and choose which of the distributed scripts might be useful. Any time along the way a “universal” database using all the history scripts can be created and re-distributed.

There are many approaches to building data analysis tools and some remarkably powerful tools are available both commercially and as freeware. The best tools are always developed in the context of specific problems. MatOFF is the current incarnation of a tool that traces its origins to the father of primate behavioral neurophysiology, Edward V. Evarts (Cowan *et al.*, 2000), and has co-evolved these past three decades with the explosive expansion of the field. From its inception, the progenitor to MatOFF aimed at concise, flexible and efficient classification of behavioral trials. MatOFF inherits these qualities and extends the power of trial-by-trial analysis by integrating the MATLAB language internally to its search engine, rather than as a post-processor. The examples used are among the most complex neurophysiological behaviors published in the field, and they highlight the benefits of MatOFF as a tool. Conceptually, the sandbox integration of a high level language into the search component of existing tools is an approach that all tool designers should consider, especially those who write new programs for each new experiment. Standardization, especially for data exchange and reuse, is an important goal of experimental software development and management. Various standards and organizational schemes have been developed to make data accessible through a common interface (Kötter, 2001; Bradley *et al.*, 2005, and many others). Some schemes employ a relational database (Bradley *et al.*, 2005), others use XML (<http://neurodatabase.org>), and others rely on a Microsoft Windows dynamic link library (DLL). The MatOFF internal file structure has some of the features of a relational database, but deviates from this architecture to make event sequence searches both fast and flexible. A description of the file structure is provided with the MatOFF distribution and as Supplementary Note 2 online. For data sharing with MatOFF, we have adopted the Neuroshare standard for importing data (<http://neuroshare.sourceforge.net>), which uses a standardized DLL interface. As of this writing, a Cortex Neuroshare DLL is being tested and the ability to read files via Neuroshare is in development. MatOFF has several facilities to generate text files. Many of our results are output as text files in a columnar format, then imported to a spread sheet (Microsoft Excel) or relational database (Microsoft Access) for further analysis.

Despite all its power, MatOFF continues to be the tool of choice in our laboratory for the beginner or non-programmer. MatOFF is easiest to learn by demonstration, but two well-documented examples are available. One example is in the *ezstart* directory of the distribution and described in Supplementary Note 1 online. With a single command it will create a plot from a sample Cortex data file, or from the user’s own data file. The other is an example of history scripting. It is available on the MatOFF Web site as *history-scripting-example.zip* and in the MatOFF distribution under *example_files*. Both examples show how to go from a Cortex file to a plot in relatively few commands. When learning by demonstration, the GUI is preferred. Although potentially daunting at first, the GUI is ideal for a quick look at data and for testing “what if” scenarios. We find new users can be trained to use the GUI quite rapidly. Intermediate users graduate to batch files and text output once they are familiar with GUI operation. With the advent of history scripting, MatOFF power users can now apply MATLAB programming skills without having to abandon their most productive tool.

Supplementary Material

Refer to Web version on PubMed Central for supplementary material.

Acknowledgements

The authors wish to thank Dr. Paul Cisek for his comments on an early version of the manuscript. This research was supported by the Intramural Program of the NIH, National Institute of Mental Health. The authors also wish to acknowledge the work of Dr. Karl Arrington for his pioneering work on the KOFF data analysis program.

References

- Bradley DC, Mascaro M, Santhakumar SA. Relational database for trial-based behavioral experiments. *J Neurosci Methods* 2005;141:75–82. [PubMed: 15585290]
- Cowan WM, Harter DH, Kandel ER. The Emergence of Modern Neuroscience: Some Implications for Neurology and Psychiatry. *Ann Rev Neurosci* 2000;23:343–391. [PubMed: 10845068]
- Genovesio A, Brasted PJ, Mitz AR, Wise SP. Prefrontal cortex activity related to abstract response strategies. *Neuron* 2005;47:307–320. [PubMed: 16039571]
- Genovesio A, Brasted PG, Wise SP. Representation of previous and future and previous spatial goals by separate neural populations in prefrontal cortex. *J Neurosci* 2006;26:7305–16. [PubMed: 16822988]
- Kötter R. Neuroscience databases – tools for exploring brain structure-function relationships. *Phil Trans Roy Soc Lond B* 2001;356(1412)
- Sclabassi RJ, Harper RM. Laboratory computers in neurophysiology. *Proc IEEE* 1973;61:1602–14.
- Wise SP, Murray EA. Role of the hippocampal system in conditional motor learning: mapping antecedents to action. *Hippocampus* 1999;9:101–17. [PubMed: 10226772]

Sequence Menu

Channel selection Display window
 Search sequence Trial ordering

File Menu

File Script file
 Unit Wizards

Layout Menus

Raster layout Histogram layout
 Analog layout Text layout

Fig 1. MatOFF sequence menu, file menu, and layout menus

Nearly every menu entry has a matching MatOFF command-line command and vice versa, so command-line and menu entries are interchangeable. Command-line commands immediately update the menus, so the system state is always visible. **Sequence Menu.** This menu selects single unit spike and analog channels, search criteria, and the temporal organization of plots. It includes a sub-sampling of trials both before (“Source Trials”) and after (“Trials”) the search for matching sequences. **File Menu.** This menu selects the data source and an optional scripting file. It also is used to launch some simple wizards. **Layout Menus.** This compact/overlapping collection of menus manages nearly every detail for plotting behavioral, single unit spike, and analog data. Each plot type can be turned on and off, repositioned, annotated, and adjusted in various ways. Up to 10 “layouts” can be defined at any one time.

Fig 2. Example MatOFF commands

MatOFF has 116 basic commands, not counting parameters and variants. The commands listed here are organized in functional groups (Configuration and environment, Unit selection, Search criteria, and Raster display, centering, shift, time sort), but the order is irrelevant because of MatOFF's state-based execution engine. Note the second command, "ALDO". It is an example call to a text file of MatOFF commands. Except for that command, each of the commands used in this example could have been called with a two- to four-letter case-insensitive abbreviation.

Fig 3. Converting Cortex data files into a MatOFF Database

Files in the Cortex data file format are collected and translated to a MatOFF Database by the “:makdat” command, under the direction of a List file. The List file flexibly assigns one or more data files to a unit name, and controls filtering options that are built into the conversion process. The resulting MatOFF Database is efficient for searching and retrieval, and has the capacity to manage data from thousands of single units. The Database is a group of files with a common root file name.

Fig 4. Data conversion for files collected by various data acquisition systems

MatOFF converts Cortex data files to the MatOFF Database structure (Fig 3). Files in the Plexon, Inc. (.PLX) and Alpha-Omega (.MAP) formats convert readily to the Cortex file format using available tools. NeuroShare (<http://www.NeuroShare.org>) support for Cortex is in development, so files from any vendor with a Neuroshare DLL (dynamic link library) can be used with MatOFF.

Fig 5. History Scripts are MATLAB commands embedded in MatOFF

The “History script” is integrated in between the MatOFF sequence matching engine (Scan Sequences) and the accumulation of matching trials (Accumulate). The execution phase steps are shown in chronological order from top to bottom on the left. The history script has access to all data from the current unit as well as history data (class values) saved from any previous history script. The history script can store information as new or changed class values, or it can save information to files using standard MATLAB file commands. The box on the lower right shows the organization of class values. A given class can have a scalar value associated with any of the trials in the unit.

Fig 6. How MatOFF uses class values

Class values generated by history scripts can be used in a number of ways. They are available for subsequent history scripts, they can also be used to mark events on rasters, order rasters, and control the time range of XY plots. Other uses for class values are in development.

Fig 7. Task design and example neural activity for a strategy cell

Selecting matching instances of strategy-based behaviors required a complex scheme of classification based on the animal's behavioral history

A. Sequence of task events. Gray rectangles represent the video screen, white squares show the three potential response targets (not to scale), the central white dot illustrates the fixation point, and the converging dashed lines indicate gaze angle. Disappearance of the instruction stimulus was the trigger stimulus, after which the monkey made a saccade (solid arrow) and maintained fixation at the chosen target. The target squares then filled with white, and reinforcement (dotted arrow) was delivered when appropriate

B. Strategy task. Responses shown by the thick arrows in the middle column represent correct application of the repeat-stay (pink background) or change shift (blue background) strategies. + indicates a rewarded response; – an unrewarded response. If unrewarded, the monkey then got a second chance to respond and received reinforcement for choosing the saccade direction made least recently

C. Example strategy cell with a preference for change-shift trials. Each dot corresponds to the time of a neural action potential. This cell had a much greater activity for change trials than for repeat trials while the instruction stimulus was being presented

D. The same cell showed selectivity for the previous goal. First chance trials are sorted into raster displays according to the previous goal, indicated by a black arrow. Responses during

the fixation period and after the appearance of the instruction stimulus were diminished when the previous goal was to the left.

Fig 8. Trial classification for late binding

History scripts can be used to develop multiple trial classification schemes before settling on a specific analysis (“late binding” approach). The upper left box is a batch file that executes “Protocol1” on the data from each unit of a population. Protocol1 (adjacent box) is a file of MatOFF commands that applies several different search criteria, and calls the Classification History Script to classify each trial of the unit. There are no practical limits to the number of units, trials, or classification schemes that can be applied at this level of processing. All results are stored as class values for each trial, and these values are available for the subsequent “specific analysis” (Fig 9).

Fig 9. Specific analysis makes use of late binding

A complex experiment may require many specific analyses to test hypotheses, illustrate population behavior, select example cases, etc. Each specific analysis can be carried out for all units by using a batch file in concert with a protocol file, Protocol2 in this figure, that calls a an “Analysis History Script”. Processing steps are the same as Fig 8. The History Script uses both acquired data and previously calculated class values from the Trial Classification (Fig 8). The output from the specific analysis can be standard MatOFF plots (lower left), a text output file for other programs to use (lower right), or a unique display programmed in the history script.