Plotting and Analysis for Neural Database-Oriented Research Applications (PANDORA) Toolbox — User's and Programmer's Manual

Cengiz Günay

Document Revision , December 10, 2017

Contents

1 Introduction

1.1 What is the PANDORA Toolbox?

The PANDORA Toolbox is a software package which consists of a collection of MATLAB object-oriented classes and script functions for creating, analyzing and visualizing databases based on data from electrophysiological neuron simulations and recordings.

1.2 Why did you make it?

Motivations to create this software were:

- Analyze data generated by brute-force and other parameter search methods.
- Analyze subsets of parameter spaces and special cases.
- Evaluate robustness of model neurons.
- Find functional roles of specific conductances.

1.3 How is it implemented?

A custom database management system (DBMS) is written from scratch in the MATLAB language. The toolbox design follows object-oriented programming principles. It uses functions from the statistics and signal processing toolboxes of MATLAB, but they are not strictly necessary. It does not use MATLAB's database (DB) toolbox.[1]

1.4 How can I use it?

The PANDORA Toolbox uses an object-oriented approach to provide maximal flexibility for interactive use on the MATLAB command-line.[2] Objects can be created, modified, analyzed, and visualized interactively in few steps. It is straightforward to save and load binary representations of these objects into files. Scripts can be made to programmatically repeat these procedures. Existing object classes are designed with the prospect of future extension, to accommodate new types of data and analyses.

1.5 Who is it made for?

PANDORA Toolbox is customized for neuroscientific research. However, the concepts of a complex dataset, extraction of multiple observations from each item of the dataset, and analysis of multi-dimensional parameter spaces are universal. In its current form the database and dataset classes can be used for data other than electrophysiologic sources. As this toolbox is designed for flexible extensibility, one can add extensions that deal with different types of data and analyses.

[1] At the time of initial design, the author did not have access to the DB toolbox. Future versions may support the DB toolbox.

[2] This version of the toolbox does not yet have a general graphical user interface (GUI). The author prefers to have a flexible command-line interface than to maintain a limited GUI. However, once commonly used functions can be conveniently placed within a GUI, it will be added to the toolbox.

1.6 Finding your way around

The source code uses MATLAB's documentation system, therefore all methods and classes are documented. To get help about all classes, issue the

```
>> help classes
```

at the MATLAB prompt. This should give you an overview of available classes. Then, to learn about a specific class, ask for the documentation for the constructor method. For instance, for the `trace` class, issuing

```
>> help trace
```

gives you the documentation for the constructor together with an overview for the class. Sometimes, if there are multiple methods with the same name under different classes, you may get the wrong documentation. In that case, you can specify the class from which to take the method by prepending the class name to the method, such as in

```
>> help trace/spikes
```

In order to learn all methods available for a class, you can use MATLAB's `methods` command. For the `trace` class, do

```
>> methods(trace)
```

However, some documentation may be outdated or simply wrong. Please report these to the author via e-mail to `cgunay AT emory.edu`.

1.7 Overview of this document

Next, Section 2 guides the reader through the installation of the package and other dependencies. You can skip this section if you already have a running software environment. Section 3 introduces the essential components of the software and talks about their design decisions. You can also skip this part if you're not interested in the guts of the system and you are in favor of a quick start. The recipes in Section 5 provide a tutorial for some common tasks. It may be easier for some readers to follow these recipes to jump-start using the software. However, it is recommended that you familiarize yourself with the basic organization of the classes before proceeding into more complex tasks. Section 6 takes the tutorial approach to describe common visualization tasks. Finally, Section A points to the list of individual methods provided by the software. These methods are documented in detail using the MATLAB online help system.

2 Installation

Download the latest package file from:
`http://userwww.service.emory.edu/~cgunay/pandora`.
Unpack the archive anywhere in your system, using

```
$ tar xzf pandora-xyz.tar.gz
```

and follow the instructions in the README file.

Basically it involves pointing your MATLAB installation to look at the `pandora/` subdirectory for loading the PANDORA files. This can be achieved by adding this directory to your MATLAB search path using the `addpath` Matlab command. To avoid its repeated application for each new session, you can have a startup script, `startup.m`, in the directory that you run MATLAB with the following commands:

```
%--- startup.m for matlab
addpath /my/download/directory/pandora-1.0b/pandora
%--- end startup.m
```

This will be loaded everytime you run MATLAB from this directory. in UN*X systems, this can be improved further by placing the command in the file `$HOME/matlab/startup.m`, `which` is executed no matter from where MATLAB is called, especially if you are running Matlab from different or unknown places each time. In Windows, place the file under `My Documents/MATLAB`, or add the directory to the search path using the *File->Set Path* menu option.

Part I
Software Architecture

3 Toolbox Components

An overview of the toolbox functionality is shown in Figure 1. In the figure, boxes represent objects that can be created with the toolbox. Flow starts from the dataset object on the top left which represents the collection of raw data files. The raw data is loaded using information in the dataset to create intermediate objects that, for instance, contain data traces. These objects define electrophysiological measurements to be entered into the data matrix of the database object on the top right. The database object allows filtering and querying to refine its contents. From the database object, one can always go back to the dataset and find the raw data that results from a query. The arrows going to bottom objects and corresponding plots show the types of possible analyses that can be done on a database object. These analyses are typically for displaying statistical information. The red arrow is a speacial analysis for searching and matching rows between different databases. The match is done by taking a row from a database created with data from real neurons and finding best matching model neurons from a simulation database.

The objects in the figure are instances of classes that define their properties in the object-oriented framework. Each class comes with a hierarchy of of subclasses that specialize to specific functions. Subsequent sections describe each of these class hierarchies that make up the main components of the toolbox.

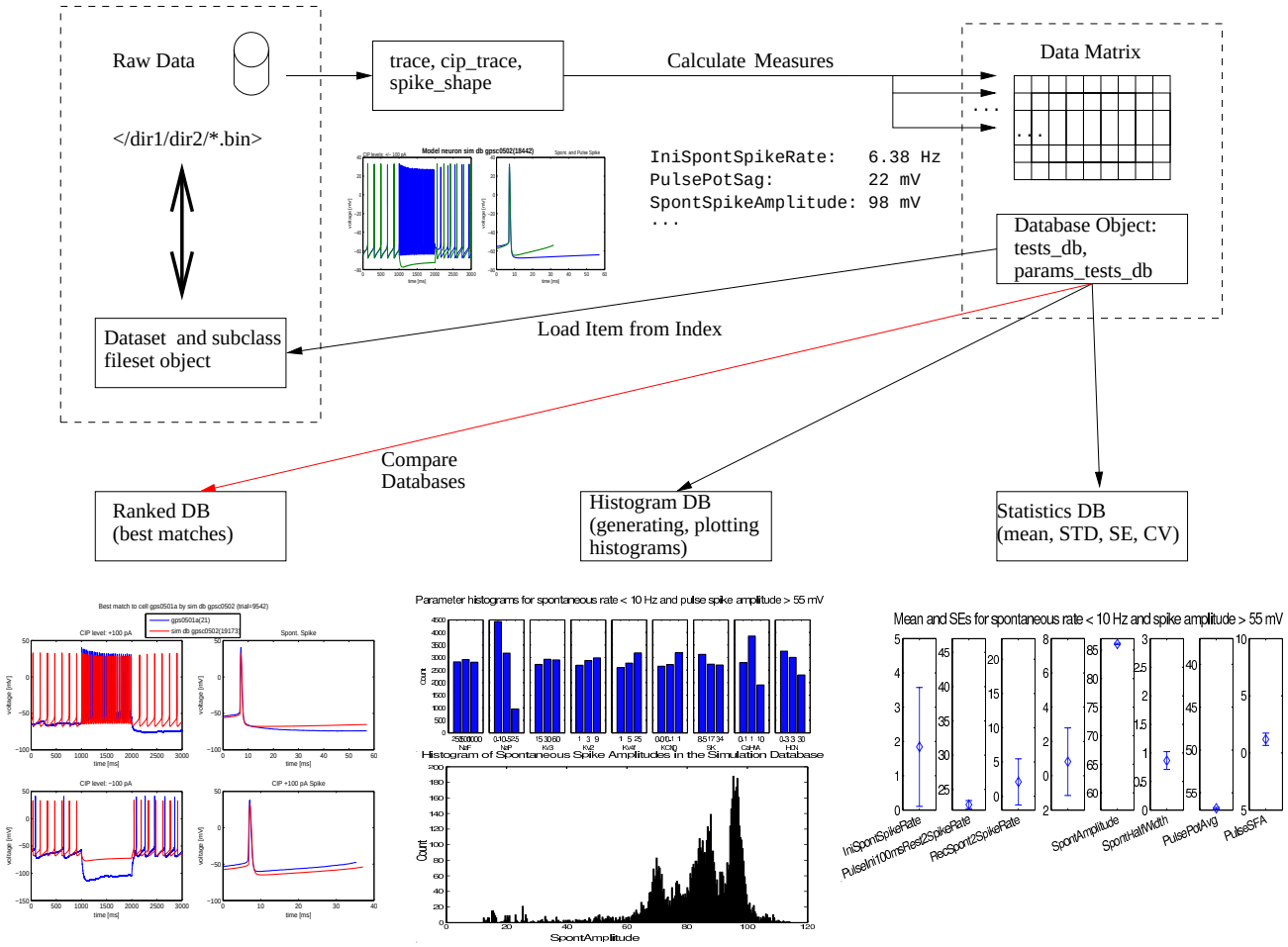


Figure 1: Schematic overview of the database software.

3.1 Databases hold all the information

The database object is at the center of this toolbox (see Figure 1). It holds a data matrix with rows as observations and columns as attributes. The rows would normally correspond to results from individual data traces, or simply neurons. The columns hold values of separate measurements, statistical data, or parameter values.

A database object can be created from any of the classes in the hierarchy of Figure 2. The top-level database class is `tests_db` which contains a two-dimensional data matrix of real numbers and some metadata. The metadata consists of column labels (e.g., measure names), a dataset label, and data properties (e.g., time resolution). The subclasses are specialized for different tasks.

If the database object is created using a dataset object, this maintains a connection from the elements of the database (e.g., neurons) to the raw data. This allows raw data associated with database contents to be visualized during analysis. However, a database can be created from any data matrix given in the proper format.

Some specialized subclasses of `tests_db` are as follows:

`params_tests_db` The first `num_params` columns are reserved for parameters that were changed between different rows. It contains methods that treat these columns specially. Parameters can be simulation parameters, or pharmacological applications to experiments.

`tests_3D_db` Contains a three-dimensional data matrix that has additional dimension for pages of information. This is mainly used to look at change in measurements with a parameter using the `invarParam` method of `params_tests_db`. Three dimensional databases can be useful for other purposes as well.

`stats_db` Contains few rows that describe the statistics obtained possibly from another database. It can contain the mean and standard deviation or error, or in some cases, the minimal and maximal values of columns in a database. It contains special plotting functions. There are methods that use the statistics collected by this class.

`ranked_db` Contains distances that resulted from a comparison of a database with a criterion. Its rows are ranked and sorted according to this distance value. Each row would point to a row in dex into the original database. Contains methods to generate reports from information about matching neurons.

`spikes_db` Contains results from individual spike shapes of a `trace` object. It can be obtained using the `trace/analyzeSpikesInPeriod` method.

`histogram_db` Each row corresponds to a histogram bin. Contains plotting methods.

`corrcoefs_db` Each row corresponds to a correlation coefficient. Contains plotting methods.

`cluster_db` Each row corresponds to a cluster centroid. Contains plotting methods.

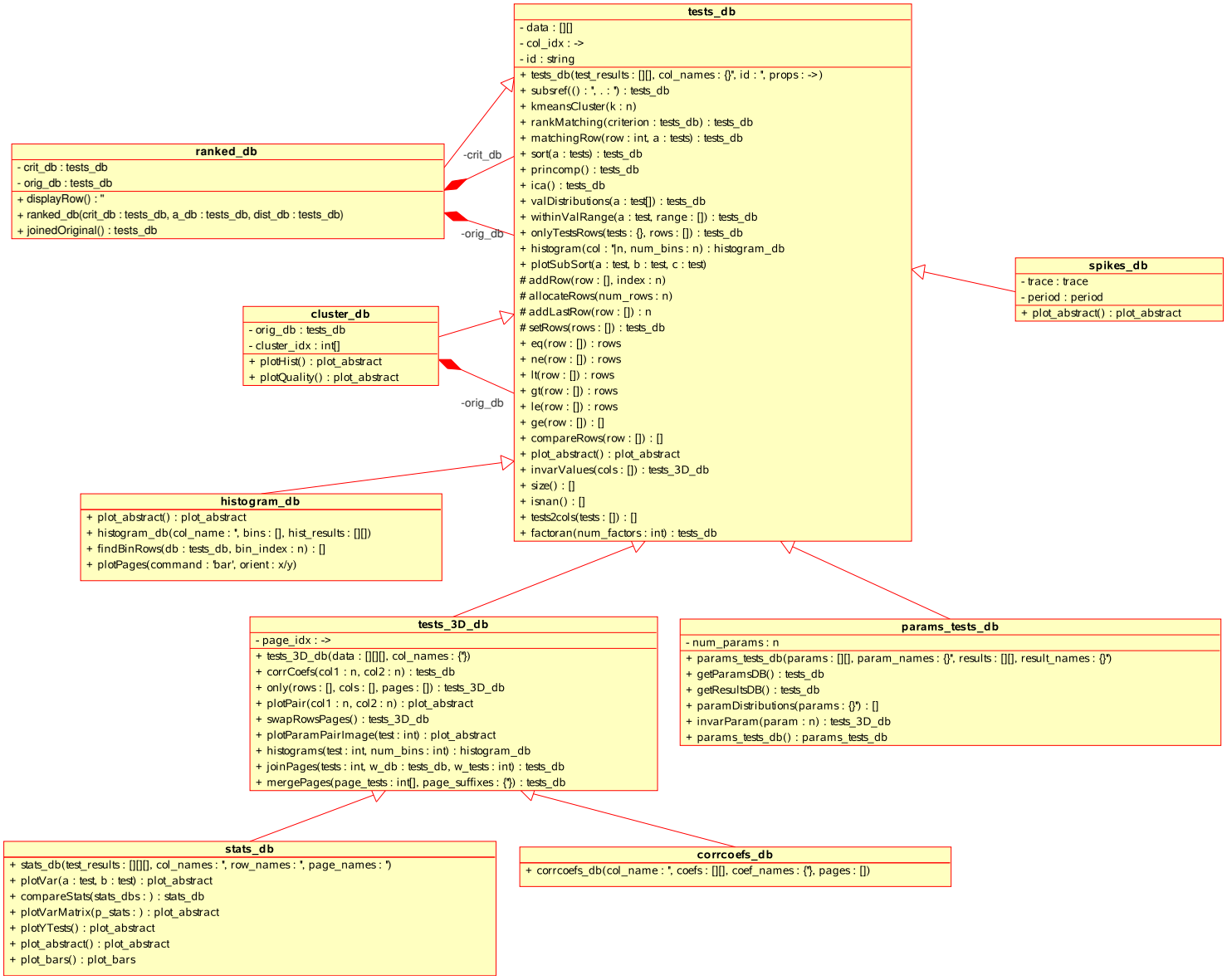


Figure 2: Database class hierarchy.

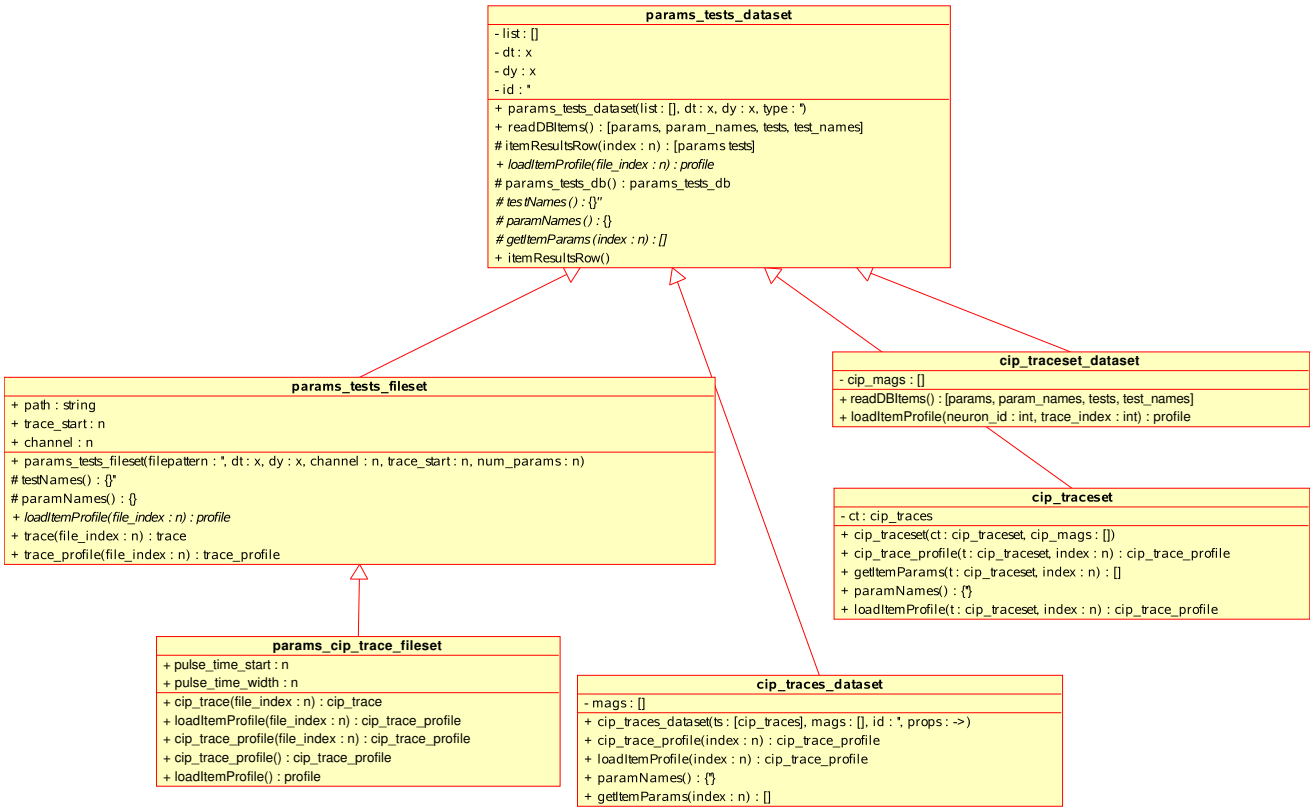


Figure 3: Dataset class hierarchy.

3.2 Datasets create the databases

The dataset object is responsible for creating the database objects (see Figure 1). It defines where the raw data is stored and what parameters are used to load and analyze it. It knows that raw data has parameters associated which individual raw data traces and how and which measures will be generated. This information is used to automatically generate a database from the dataset. It also allows reaching back the raw data from rows of an analyzed database.

Figure 3 shows the hierarchy for the dataset classes. The top-level dataset class is `params_tests_dataset` which is an incomplete class. That is, this class defines general utilities that can work for a variety of dataset subclasses, but one cannot make a object from the `params_tests_dataset` class directly. Instead, one of its subclasses must be chosen and used. Some of these specialized subclasses are as follows:

`params_tests_fileset` This class assumes each raw data item resides in a file and all of these files are in the same directory. The parameter names and values are obtained from each file name itself. This class is mostly useful for simulation filesets.

`params_cip_trace_fileset` This is a subclass of `params_tests_fileset`, therefore it inherits the notion of one file per data item. The files must conform to the current-pulse injection experiments and have a starting time and duration for the pulses. The pulse magnitude is read from the `pAcip` parameter. This class is mostly useful for simulation filesets.

`physiol_cip_traceset` This is a subclass of `params_tests_dataset`. It is designed to load a set of physiology traces from a single file generated by the PCDX stimulation and acquisition software.

`physiol_cip_traceset_fileset` This is a subclass of `params_tests_dataset`. It is designed to load traces from multiple PCDX data files. It uses the `physiol_cip_traceset` class for this purpose.

`cip_traces_dataset, cip_traceset, cip_traceset_dataset` These are obsolete classes that allow loading physiology traces from older MATLAB formatted objects.

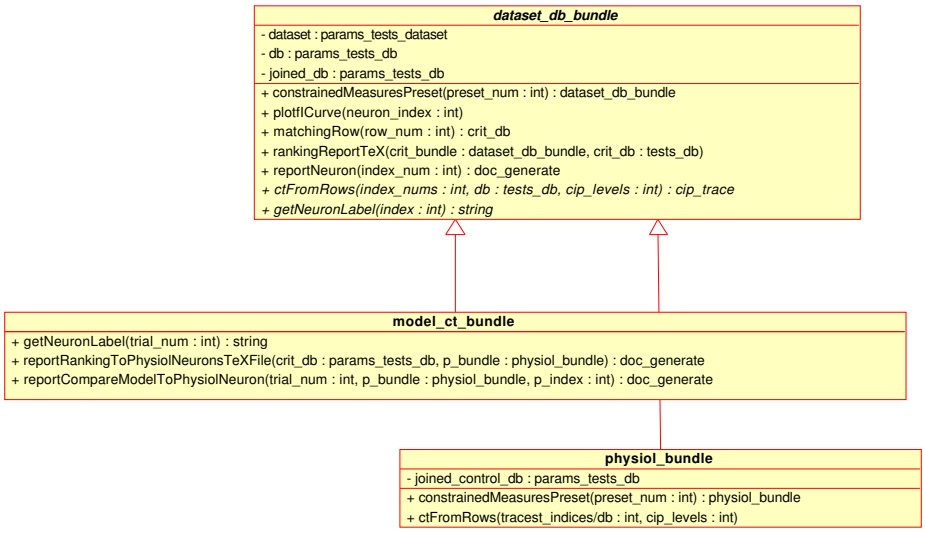


Figure 4: Bundle class hierarchy.

3.3 Bundling the database and dataset together

Since dataset and database objects are related and work together for some operations, it is convenient to have another object that bundles them together. There are several analysis routines that start from the database, retrieve raw data traces and other related information from the dataset and create a result. For instance, matching neurons from one database to another requires first comparing the measurements to find match candidates, and then comparing raw traces to visually represent the match quality.

The top-level `dataset_db_bundle` class in Figure 4 fulfills this purpose by bundling a `dataset` with the raw database, `db`, created from it, and with the reduced database, `joined_db`, that contains a one-row-per-neuron representation. Although being a virtual class that cannot be instantiated, it contains general methods and prototype methods that must be implemented in subclasses. This way, it provides guidelines for defining subclasses. Its two subclasses provide specialize methods for model and physiology databases, respectively.

`model_ct_bundle` Contains methods to name and visualize neurons in the model database. It has methods to compare real neurons to model neurons to find best matching candidates.

`physiol_bundle` Contains methods to name and visualize neurons in the physiology database. It contains a new attribute, `joined_control_db`, that holds only the neurons recorded without any pharmacological treatments.

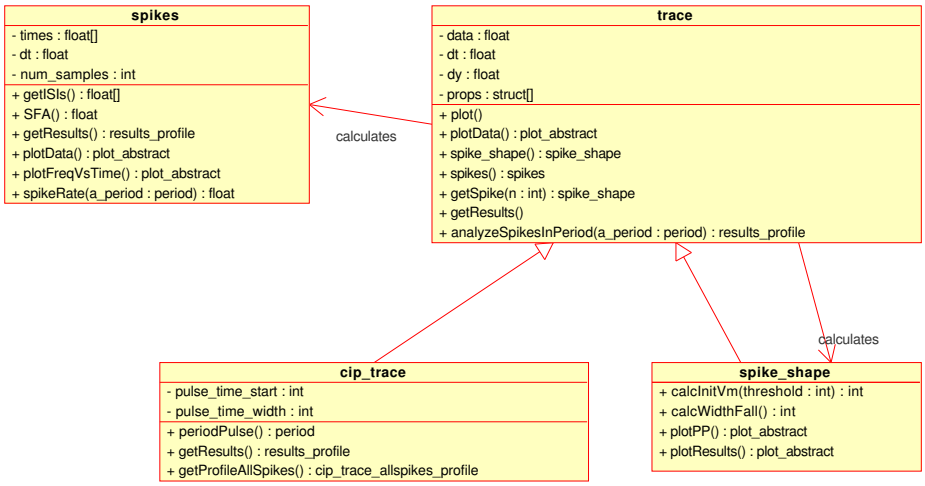


Figure 5: Data wrapper class hierarchy.

3.4 Wrapper classes hold raw data

Wrapper classes are designed to hold data and provide simple methods that operate on them. They can either hold raw data, or intermediate processed forms of data being byproducts of analysis routines. In the overall schema of Figure 1, the raw traces obtained from the dataset object are kept in data wrapper objects.

Figure 5 shows the hierarchy for the data wrapper classes. The most basic data wrapper class in this toolbox is the `trace` class, which holds raw voltage or current traces. The `spikes` object contains the spike times obtained by analyzing a `trace` object.

A data wrapper class does more than just holding the data. It defines a set of operations in terms of method functions that can work on the data held by the class. As a rule of thumb, if one needs to add some new functionality into the toolbox, it should be added as a method into a class holding the data on which to operate.

Some of the data wrapper classes are as follows:

`trace` Generic object that holds a vector of data that changes over time. It has a time resolution and y-axis resolution. Contains simple analysis routines such as finding average values within different periods, or finding spikes given a threshold.

`cip_trace` A subclass of `trace` class for current-injection recording protocols. It defines an initial spontaneous period, followed by a current-injection period, and final recovery period. It contains period-specific analyses that apply to the experimental protocol.

`spike_shape` A subclass of `trace` that holds the shape of a single spike. It contains spike shape measurements.

`spikes` A generic class to hold the event times for spikes. It contains methods for making measurements based on spike times, such as rate and ISI calculations.

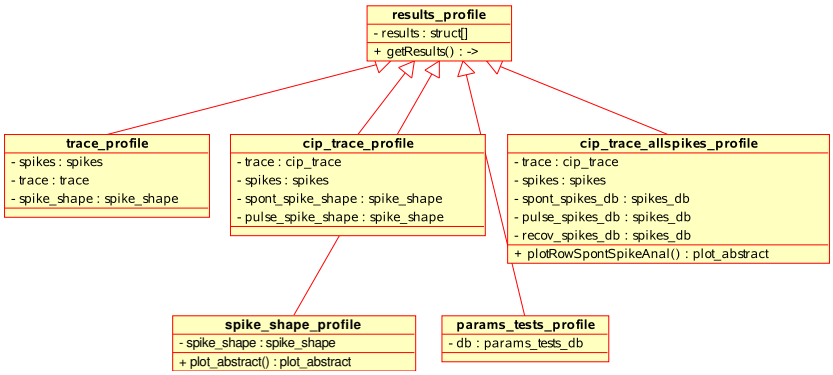


Figure 6: Profile class hierarchy.

3.5 Profiles hold results of measurements

Profile classes are designed to hold results of analysis and measurements on the data wrapper or database objects. The data and results are separated into different classes for added flexibility of saving data and results separately. Yet, the profiles normally keep a copy of the data wrapper object from which they obtained the measurements. The intention is to save the measurement results for possible visualization or later inspection, without having to repeat the analyses.

In Figure 6, the top-level `results_profile` class contains a simple MATLAB structure variable, `results`, that holds a set of name-value pairs. These are names of measurements and their corresponding values. Most of the subclasses are simplistic, and they exist only for organizational reasons. Some of them may implement specialized plotting methods that make use of the saved measurements. These subclasses can be briefly described as follows:

`trace_profile` Holds measurements from a `trace` object. It contains the `trace` object and the `spikes` found in it, and averaged `spike_shape` object.

`cip_trace_profile` Holds measurements from a `cip_trace` object with a current-injection period. It contains the original `cip_trace` object and the `spikes` found in it. In addition, it holds averaged `spike_shape` objects from the spontaneous and current-injection periods.

`cip_trace_allspikes_profile` Extended version of `cip_trace_profile`. Instead of single averaged spike shapes, it contains spike databases from the spontaneous, current-injection and recovery periods. These databases only retain measurements made from individual spikes, but not their shapes.

`spike_shape_profile` Holds measurements made from a `spike_shape` object.

`params_tests_profile` Holds analysis results from a `params_tests_db` object.

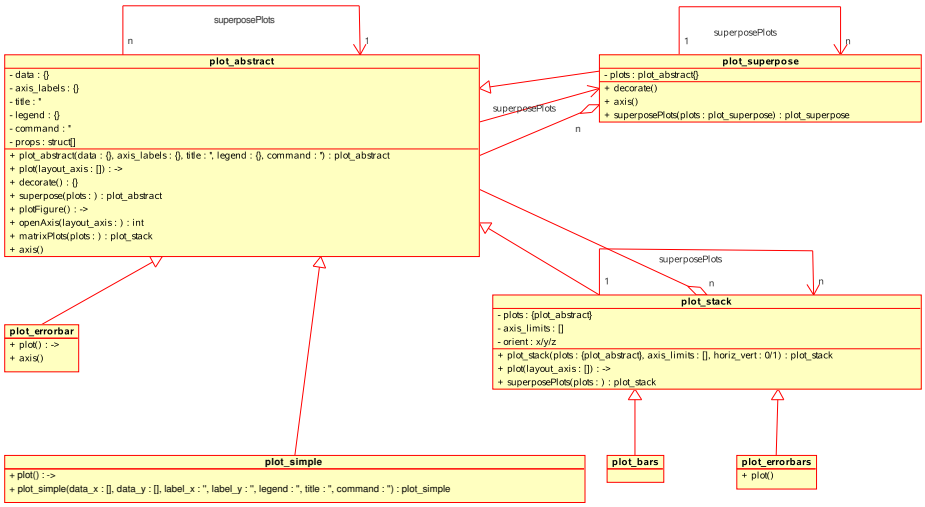


Figure 7: Plot classes hierarchy.

3.6 Integrated plotting for easy visualization

To integrate visualization into each class, common MATLAB plotting features are implemented in the supporting classes seen in Figure 7. These bring an object-oriented approach to plot generation in MATLAB. Plots can be generated as objects, saved, modified and included as subplots in larger plots.

The main plotting classes are `plot_abstract`, `plot_superpose`, and `plot_stack`. The most general plotting template class, and the top-level class in the hierarchy, is `plot_abstract`, which plots an axis using a single MATLAB command, like `plot` or `bar`. Multiple `plot_abstract` objects that use the same command can be superposed and still act as a single `plot_abstract` object. If they require different plotting commands (e.g., mixing `plot` and `text` labels), a `plot_superpose` object must be used that is composed of an array of `plot_abstract` objects. Multiple `plot_abstract` objects or any of the subclass objects can be composed together in a horizontal or vertical stack using the `plot_stack` class. Since `plot_stack` is itself a subclass of `plot_abstract`, it can be stacked as well. This allows creating virtually any complex structured figure using the three classes. Each of these classes have several properties that control the layout and details of placement and looks.

The rest of the classes in the hierarchy create typical types of plots for convenience:

`plot_bars` Multi-axis bar plot with extending errorbars using a combination of the `bar`, `errorbar`, and `text` commands.

`plot_errorbar` Single-axis errorbar plot using the `errorbar` command.

`plot_errorbars` Multi-axis errorbar plot using the `errorbar` command.

`plot_simple` Simplified single-axis, single command plot.

3.7 Miscellaneous classes

These are miscellaneous classes that do not fit into any of the above categories:

`period` Defines a period composed of a start and end time for operations on traces, etc.

`script_array` Defines a looping construct that can be extended. It defines an initialization routine, a job that needs to be repeated, and a finalization routine.

`script_array_for_cluster` Subclass of `script_array`, it can submit the array job to run in parallel on a computing cluster that supports the Sun Grid Engine (SGE) commands.

`script_factory` Factory class to generate an enumerated array of script files to be distributed on several machines and run in parallel. It also defines a final function to gather results. It is recommended to use `script_array_for_cluster` instead.

4 Programming Conventions

4.1 Using property structures for passing optional arguments to methods

For flexibility in passing optional arguments to methods, this toolbox adopted using property structures. A MATLAB structure, mostly called `props`, is passed to a method as the last argument:

```
>> props.optionalParam = 1
>> myFunc('hello', props)
```

Each method defines a list of accepted arguments that can be defined as fields in the structure, but should be able to execute without them by substituting defaults. Using a property structure is advantageous over using the `varargin` keyword for variable number of arguments, because properties allow adding and deleting arguments in methods without causing changes to the contents of the method. Since arguments are addressed by names rather than positional arguments, missing arguments do not affect the other arguments.

Most objects keep a property structure that define custom attributes passed at time of construction. These classes define a `setProp` method to modify properties after being created.

4.2 Overloaded operators for transparent access to object contents

The simplistic implementation of object-oriented programming features in Matlab impose several strict limitations. MATLAB's powerful and flexible operator overloading feature helps overcome these limitations.

PANDORA Toolbox uses MATLAB operator overloading to facilitate manipulation of local and parent object fields. In MATLAB, object fields can only be

accessed from the object's private methods. This means one cannot access the object fields using the dot operator. To give an example, the `trace` object has a `dt` field for time resolution. The following command fails:

```
>> mytrace.dt = 1e-4;
??? Object fields can only be accessed within methods.
```

Everytime object contents need to be addressed, a method must be called. The recommended way to do this is by defining separate getter/setter methods for each field of the object. For instance, writing `getDt` and `setDt` methods for accessing the `dt` field. This creates a lot of burden for the programmer not just creating a class, but also maintaining it later. Although this probably was intended for strictness in building object-oriented constructs, it is highly inconvenient for command-line manipulations. Therefore our toolbox objects offer generic `get` and `set` methods that can read or write the value of any of its fields:

```
>> mytrace = set(mytrace, 'dt', 1e-4)
>> get(mytrace, 'dt')
ans = 1e-04
```

These methods are almost identical across different classes. In addition to this, defining the special `subsref` method for objects allow overloading the dot (`.`), parenthesis (`()`), and curly brace (`{}`) operators. Most[3] of the objects in the toolbox allows using the dot operator to read or write to fields. Overloading these operators also help with the limitation of accessing parent object fields, a problem not found in other object-oriented languages such as JAVA. For example without any overloading, from the subclass class `cip_trace` one needs to first address the parent class name, and then `dt`:

```
>> myciptrace.trace.dt
ans = 1e-4
```

After defining the overloaded operator that call parent methods, one get reach `dt` directly:

```
>> myciptrace.dt
ans = 1e-4
```

Some classes overload indexing operators to allow accessing special functions. For instance the main database class, `tests_db`, overloads parenthesized indexing to access cells in the database matrix. Some classes define the special `subsasgn` method to overload the assignment operations when the object is on the left-hand-side of the operation. This allows the command:

```
>> mytrace.dt = 1e-4;
```

which would otherwise need to be done the following way:

```
>> mytrace = set(mytrace, 'dt', 1e-4);
```

4.3 Troubleshooting errors

For debugging problems with methods, one can turn on the verbosity of information display during execution with:

```
>> warning on verbose
>> warning on backtrace
```

[3] May not be implemented for all objects.

4.4 Creating a new class

To get the benefit of overloading, the top-level class must have the generic `subsref` and `subsasgn` methods. These methods can be copied from any of the other top-level classes. Any subclasses should have the generic `get` and `set` methods in place.

Part II
User's Manual

5 Recipes for Common Tasks

5.1 Create a database from arbitrary data

A database can be created directly from a data matrix.

```
>> cap = [1 2 3 1];
>> res = [5 6 4 5];
>> a_db = tests_db([cap' res'], {'Capacitance', 'Resistance'}, {}, ...
'Canton S - fake');
```

5.2 Loading a database

A secondary way would be to create a database from a dataset. First a dataset object must be created that point to the data sources. There are different dataset classes that allow using different data sources. For instance, physiology and simulation data require different operations. In physiology data, one can record information about the treatments and other conditions, whereas in simulations one can keep track of changing parameters.

Once the dataset object is obtained, the database object can be created with

```
>> mydb = params_tests_db(dataset)
```

which initiates the loading of files. This operation is the same no matter what type of dataset or fileset object is used. The following commands reduce the verbosity of output during this long process:

```
>> warning off verbose
>> warning off backtrace
>> warning off calcInitVm:info
```

5.2.1 Creating a dataset for physiology data

Physiology data can be obtained from multiple sources.

Loading data by specifying tracesets in a text file The preferred way to load physiology traces is to first create a text file where each line specifies traces to load from a single data source (e.g., a PCDX file). The format of this text file is explained in the help of the `physiol_cip_traceset_fileset` class. The physiology fileset can be created from the text file with a command such as follows:

```
>> phys_fileset =
     physiol_cip_traceset_fileset('cell_traces.txt', 1e-4, 1e-3,
                                  struct('profile_method_name',
                                         'getProfileAllSpikes',
                                         'offset_y', -9,
                                         'cip_list',
                                         [-200 -100:20:100 200 300]))
```

This command reads the `cell_traces.txt` file and records the tracesets to read from each file. The structure passed indicates to use the `getProfileAllSpikes` method to calculate the measurements on the traces.

The fileset can then be used to generate the database, as shown above, using its `params_tests_db` method. The fileset object holds within, a separate `physiol_cip_traceset` object for each line in the text file.

The `cip_list` optional parameter must be used with caution. To determine actual CIP-levels, the current channel of the trace is analyzed. `cip_list` entries are used to discretize the noisy current channel. Current levels will coerce to the nearest entry from `cip_list`. In the above example, all current levels below -200 pA will be assumed to be -200 pA. The default `cip_list` resides in the `physiol_cip_traceset/CIPform` method.

Loading data from existing `cip_traces` objects The now obsolete `cip_traces` Matlab objects have been used to hold some earlier physiological data. Each object holds a set of traces with varying CIP levels applied to the cell. The following command creates a dataset object from a cell array `ct_list` of `cip_traces` objects by choosing only the traces with 100 pA and -100 pA CIP levels

```
>> phys_dataset =
  cip_traceset_dataset(ct_list, [100, -100], 1e-3,
                       'dataset gpd 0411-21',
                       struct('offsetPotential', -9)
```

5.2.2 Creating a dataset for simulation data

First a dataset or a fileset must be created. An example to load GENESIS `.bin` files would be

```
>> fileset =
  params_cip_trace_fileset('/home/cengiz/data/*.bin',
                           1e-4, 1e-3,
                           20001, 10000,
                           'sim dataset gpsc0501',
                           struct('trace_time_start', 10001,
                                  'type', 'sim',
                                  'scale_y', 1e3))
```

The explanation of arguments can be obtained by issuing a

```
>> help params_cip_trace_fileset
```

in MATLAB. In this example, all GENESIS files were created with the same characteristics: $dt = 10^{-4}$, $dV = 10^{-3}$, pulse during samples $[20001, 30000]$. Optional properties

(the last argument) indicates that the first 10000 samples should be discarded and that the data should be prescaled to yield the dV indicated. Note that, using an absolute path to refer to data files ensures that they can be reached from different directories after the fileset object is saved.

Loading heterogeneous set of simulation files Sometimes not all data files in a simulation set would have the same length, or CIP start time. The brute-force simulation set is such an example, where the spontaneous trace and different CIP level traces are in different files. I have a special superclass that contains multiple fileset objects to automatically handle this kind of data. It resides not in the general distribution directory, but in my personal directory `/djlab/shared/matlab/classes/cengiz`. This class is an example of how to create composite fileset objects. An instance of this class can be created with:

```
>> m_filesetall =
     multi_fileset_gpsim_cns2005('.../data', '.../paramRanges.txt',
                                 '.../all_0.par', 'sim db gpsc0502')
```

which will find all the files in the given directory and put them in separate pre-defined fileset objects according to their `_pAcip_` suffixes. Parameter range definition and value files are used to read parameter values for each simulation. A single database object can be loaded using the `params_tests_db` method on the `m_filesetall` object above. The help on this method explains how to load only certain filesets at a time. This helps to load different filesets in parallel, since databases can be concatanated easily afterwards.

5.3 Finding constrained subsets in a database

Once a database with more-than-sufficient number of measures is available, subsets of this database can be extracted easily for other tasks. New databases can be formed by filtering rows, columns or pages of an existing database. For choosing any of these dimensions, the user can specify an array of indices, or a logical array. For instance,

```
>> db2 = db1(1:10, :);
```

creates a database object `db2` by the first ten rows of `db1` and all its columns. For three-dimensional databases, a third parameter can be specified, as in

```
>> db2 = db1(1:10, :, [1 3]);
```

which will take only the first and third page from `db1`.

For measures, columns can also be specified as a single string value, or a cell array of strings, as in

```
>> db2 = db1(1:10, 'pAcip');
```

which chooses only the `pAcip` column of the first 10 rows of `db1` or

```
>> db2 = db1(1:10, {'pAcip', 'IniSpontSpikeRate'});
```

which chooses two columns. Finally, composite queries can be formed when cell arrays are used for addressing:

```
>> db2 = db1(1:10, {1:10, 'IniSpontSpikeRate', 234});
```

which will select the first ten measures, the spontaneous spike rate, and the measure number 234.

Rows of the database signify neurons or simulation runs. Therefore it is important to find subset of neurons that match a certain criteria. This can be done specifying a list of rows that is the result of a logical operation. A logical operation finds rows that satisfy constraints on a single parameter or measure of a database. For instance,

```
>> rows = db1(:, 'IniSpontSpikeRate') > 10;
```

gives a logical array that contains a `true` value for all rows in `db1` that has spontaneous firing faster than 10 Hz. If this is used as the row specifier in a subset operation, a new database with only these neurons can be obtained by

```
>> db2 = db1(rows, :);
```

If we want more constrains it is straightforward to use any logical operation such as AND (&), OR (|) and NOT(~) on these logical arrays such as

```
>> db2 = db1((rows | rows2) & rows3, :);
```

which says choose all rows from `db1` where either the tests `rows` and `rows2` are satisfied and while `rows3` is always satisfied. All these operations can be specified in-place such as in

```
>> db2 = db1(db1(:, 'IniSpontSpikeRate') > 10 &
             db1(:, 'IniSpontSpikeRate') <= 20, :);
```

which will create a database of neurons that spontaneously fire between 10–20 Hz.

5.3.1 Complex Queries

Complex queries can be constructed using results of queries in nested fashion. The following example shows an example of finding all neurons that match any of the neurons in another database and then find the ones that match certain criteria:

```
>> sub_phys_es2 = phys_joined_db(phys_joined_db(:, 'NeuronId') == es2(:, 'NeuronId'), :);
>> displayRows(sub_phys_es2(sub_phys_es2(:, 'Apamin') > 0, 'NeuronId'))
```

Here, the first query returns all rows that match the `NeuronId`s from the `es3` database. The inner term in the second query finds among these neurons the ones for which apamin blocker data is present. The final enclosing block uses these rows to get a subset of the `phys_joined_db` with only the `NeuronId` column. This type query has equivalent computational power to using nested SELECT statements in the SQL language.

5.4 Sorting the database according to a measure

First, all rows where the desired measure value is `NaN` should be eliminated:

```
>> ampDecayTau_nonNaN_db =
     dball(~isnan(dball(:, 'PulseSpikeAmpDecayTau')), :)
```

This finds all rows in `dball` that the `PulseSpikeAmpDecayTau` measure is not `NaN` and creates a new DB object `ampDecayTau_nonNaN_db`, which includes these rows with all measures from `dball`. Notice that the newly created DB contains less rows than the original DB. The number of rows in the new DB can be obtained by just typing the name of the DB and pressing enter at the MATLAB prompt.

Then, one can sort the new database using:

```
>> ampDecayTau_sorted_db = sortrows(ampDecayTau_nonNaN_db,
                                    'PulseSpikeAmpDecayTau')
```

This will create DB which is sorted with increasing values of the `PulseSpikeAmpDecayTau` measure. Displaying the first few rows gives lowest values:

```
>> displayRows(ampDecayTau_sorted_db, 1:3)
ans =
    'NaF'                    [      1000]    [     250]    [     250]
    'NaP'                    [    0.5000]    [  0.5000]    [  2.5000]
    'Kv3'                    [        60]    [      15]    [      30]
    'Kv2'                    [         9]    [       3]    [       3]
    'Kv4f'                   [         5]    [       1]    [      25]
    'KCNQ'                   [    0.1000]    [  0.0100]    [  0.1000]
    'SK'                     [    8.5000]    [      34]    [  8.5000]
    'CaHVA'                  [        10]    [  0.1000]    [      10]
    'HCN'                    [        30]    [  0.3000]    [      30]
    'pAcip'                  [       100]    [     100]    [     100]
    'IniSpontISICV'          [3.9448e-04]    [  0.0051]    [  0.0452]
    'IniSpontPotAvg'         [  -64.9161]    [-52.6859]    [-41.5876]
    'IniSpontSpikeRate'      [   14.0014]    [ 18.0018]    [ 81.0081]
    'PulseISICV'             [    0.0226]    [       0]    [  0.0366]
    'PulseIni100msISICV'     [    0.0541]    [       0]    [  0.0814]
              [1x27 char]    [   28.8953]    [       0]    [ 88.9086]
              [1x27 char]    [   26.6785]    [       0]    [ 86.7052]
              [1x22 char]    [        30]    [      20]    [     100]
              [1x25 char]    [   29.4118]    [166.6667]    [ 96.0512]
    'PulsePotAvg'            [  -61.0044]    [-32.7775]    [-34.1518]
    'PulsePotMin'            [       NaN]    [     NaN]    [     NaN]
    'PulsePotSag'            [       NaN]    [     NaN]    [     NaN]
    'PulseSFA'               [    1.1254]    [     NaN]    [  1.3571]
    'PulseSpikeAmpDecayDelta'[    4.2764]    [  9.2041]    [ 17.8389]
    'PulseSpikeAmpDecayTau'  [    0.2000]    [  0.2000]    [  0.3000]
    'PulseSpikeRate'         [   28.0028]    [  2.0002]    [ 89.0089]
...
```

Displaying the last few rows gives the highest values:

```
>> displayRows(ampDecayTau_sorted_db, 13879:13881)
ans =
    'NaF'                    [     250]    [     250]    [      1000]
    'NaP'                    [  2.5000]    [  2.5000]    [    2.5000]
    'Kv3'                    [      15]    [      15]    [        60]
    'Kv2'                    [       3]    [       3]    [         9]
    'Kv4f'                   [       5]    [       5]    [        25]
    'KCNQ'                   [  0.1000]    [  0.1000]    [    0.0100]
    'SK'                     [  8.5000]    [  8.5000]    [        17]
    'CaHVA'                  [      10]    [      10]    [        10]
    'HCN'                    [      30]    [       3]    [        30]
    'pAcip'                  [    -100]    [    -100]    [       100]
    'IniSpontISICV'          [  0.0027]    [  0.0027]    [9.1376e-04]
    'IniSpontPotAvg'         [-28.5685]    [-28.5687]    [  -67.7567]
```

```
    'IniSpontSpikeRate'       [ 69.0069]    [ 69.0069]    [   14.0014]
    'PulseISICV'              [  0.0046]    [  0.0046]    [    0.0091]
    'PulseIni100msISICV'      [  0.0080]    [  0.0080]    [         0]
              [1x27 char]     [ 71.1269]    [ 71.1269]    [   24.4499]
              [1x27 char]     [ 71.1427]    [ 71.1427]    [   26.6785]
              [1x22 char]     [      80]    [      80]    [        20]
              [1x25 char]     [ 70.6357]    [ 70.6357]    [   25.4453]
    'PulsePotAvg'             [-30.1705]    [-30.1718]    [  -65.2721]
    'PulsePotMin'             [     NaN]    [     NaN]    [       NaN]
    'PulsePotSag'             [     NaN]    [     NaN]    [       NaN]
    'PulseSFA'                [  0.9792]    [  0.9792]    [    1.0407]
    'PulseSpikeAmpDecayDelta'[ -1.2791]    [ -1.2868]    [    1.2201]
    'PulseSpikeAmpDecayTau'  [999.6000]    [999.6000]    [      1000]
    'PulseSpikeRate'          [ 72.0072]    [ 72.0072]    [   25.0025]
...
```

5.5 Preprocessing a raw (physiology) database by elimination and averaging

Mostly, raw physiology databases are subject to redundancies and unwanted recordings. We usually apply the following steps before we start analyzing a raw physiology database. Similar steps may be employed for simulation databases, too.

5.5.1 Limiting range of bias currents

Recordings with high bias current are undesirable. We commonly filter-out high bias currents with:

```
>> db_bias_small =
     phys_dball(phys_dball(:, 'pAbias') > -30 &
                phys_dball(:, 'pAbias') < 30, :)
```

which will limit the bias current, i_b, to $-30\,\mathrm{pA} < i_b < 30\,\mathrm{pA}$.

5.5.2 Choosing few current levels

To get a profile for a neuron, usually both hyperpolarizing and depolarizing CIP-levels need to be included. Moreover, to capture the spiking frequency vs. current response of the neuron, multiple depolarizing CIP-level information may need to be included.

There are two counterparts to selecting which CIP-levels to include in a DB. First, one can select what CIP-levels are available in the raw data and what discretization levels should be used while loading the database. This is done with the `cip_list` optional parameter described in Section 5.2.1. GP recordings prior to mid-2005 have current channel data which are too noisy to be quantized to levels of 10 pA. Instead, at least a step size of 20 pA needs to be used. Later recordings have both better recordings, and feature 20 pA steps in the experimental protocol anyway.

Second, after the database is loaded, one can filter-out unwanted CIP-level traces:

```
>> phys_dball_limitedcip =
     phys_dball_big(phys_dball_big(:, 'pAcip') == -100 |
                    phys_dball_big(:, 'pAcip') == 0 |
                    phys_dball_big(:, 'pAcip') == 50 |
                    phys_dball_big(:, 'pAcip') == 100 |
                    phys_dball_big(:, 'pAcip') == 200, :)
```

This operation can be simplified to take advantage of complex query form:

```
>> phys_dball_limitedcip =
     phys_dball_big(phys_dball_big(:, 'pAcip') == [-100; 0; 50; 100; 200])
```

which will choose rows with current levels matching any of the given values.

5.5.3 Adding new columns calculated from existing measures

Some measures can be deduced from measures collected from raw data. These do not need to be calculated at time of loading the raw data, but rather can be added to the database later. Some measures must be added later because they may be composed of measurements from multiple traces or averages. Here is an example for adding a new measure:

```
>> phys_db_limitedcip_addedcols =
     addColumn(phys_db_limitedcip, 'PulsePotSagDivMin',
               phys_db_limitedcip(:, 'PulsePotSag').data ./
               phys_db_limitedcip(:, 'PulsePotMin').data)
```

5.5.4 Averaging multiple traces of same neuron with same CIP-level and (pharmacological) parameters

In making a one-row-per-CIP-level database, it is essential to include all available information from the raw database. Especially in physiology datasets, there may be multiple traces of a neuron where the same CIP-level and the same pharmacological conditions were applied. These rows can be averaged to obtain a single row for each CIP-level of a neuron.

Before doing this, the parameters of the raw database should only include parameters that uniquely distinguish neurons. The averaging operation tries to find each distinct set of parameters and then averages all rows that has this combination. For example, the `NeuronId` and `pAcip` parameters need to be distinct. However, the `pAbias` parameter does not need to be distinct for each neuron. The non-unique parameters need to be filtered-out before the averaging process. The following shows all the parameters of a raw physiology database:

```
>> phys_db
params_tests_db, tracesets from .../cip_traces_all_axoclamp.txt
ans =
    num_params: 16
         props: [0x0 struct]
      tests_db: [1x1 tests_db]
Optional properties of params_tests_db:
ans =
0x0 struct array with no fields.
```

```
tests_db, tracesets from .../cip_traces_all_axoclamp.txt
1527 rows in database with 182 columns, and 1 pages.
Column names:
    [  1]    'pulseOn'
    [  2]    'pulseOff'
    [  3]    'traceEnd'
    [  4]    'pAcip'
    [  5]    'pAbias'
    [  6]    'Cadmium'
    [  7]    'PicroTx'
    [  8]    'Apamin'
    [  9]    'Glycine'
    [ 10]    'KynAcid'
    [ 11]    'TTX'
    [ 12]    'XE991'
    [ 13]    'drug_4AP'
    [ 14]    'EBIO'
    [ 15]    'NeuronId'
    [ 16]    'TracesetIndex'
...
```

One can choose which parameters need to be distinct for each row by specifying as the second argument in the call to the `meanDuplicateRows` method, whereas the third argument specifies the measures to be averaged:

```
>> phys_mean_db = meanDuplicateRows(phys_db, [4 6:15], [17:161])
params_tests_db, averaged tracesets from .../cip_traces_all.txt
ans =
    num_params: 13
         props: [0x0 struct]
      tests_db: [1x1 tests_db]
Optional properties of params_tests_db:
ans =
0x0 struct array with no fields.
tests_db, averaged tracesets from .../cip_traces_all.txt
690 rows in database with 158 columns, and 1 pages.
Column names:
    [  1]    'pAcip'
    [  2]    'Cadmium'
    [  3]    'PicroTx'
    [  4]    'Apamin'
    [  5]    'Glycine'
    [  6]    'KynAcid'
    [  7]    'TTX'
    [  8]    'XE991'
    [  9]    'drug_4AP'
    [ 10]    'EBIO'
    [ 11]    'NeuronId'
    [ 12]    'NumDuplicates'
    [ 13]    'RowIndex'
    [ 14]    'IniSpontISICV'
```

```
...
```

This command ignores the pulse time information and the bias current, but includes all the pharmacological parameters, in distinguising the unique traces. It also eliminates some measures, such as the `ItemIndex`, from averaging.

5.6 Making a database by merging multiple rows from another database

A simple example for making a new database out of multiple rows in an existing database is combining multiple traces from the same neuron with different current pulse injection (CIP) levels. The initial database contains a row for each CIP level with redundant information, such as spontaneous period measurements.

5.6.1 Making a one-row-per-neuron DB from multiple CIP-level rows

Measures of same cell obtained with multiple CIP-levels can be merged to make a single row. Note that, different pharmacological conditions applied to one cell must be kept in a different rows. The following command selects measures from each of the $\{-100, 0, 40, 100, 200\}$ CIP levels to be included in the merged database:

```
>> phys_joined_db =
     mergeMultipleCIPsInOne(phys_mean_db(:, [1:13 16:180]),
                            {'_H100pA', 13 + [5:14 19:24 (119 + spike_tests) 165],
                             '_0pA', 13 + [1:4 (27 + spike_tests)],
                             '_D40pA', 13 + [5:11 19:24 (73 + spike_tests) 165],
                             '_D100pA', 13 + [5:11 14:16 19:24 (73 + spike_tests)
                                              (119 + spike_tests) 165],
                             '_D200pA', 13 + [5:11 19:24 (73 + spike_tests) 165]},
                            'RowIndex_D200pA')
```

This command operates on the previously averaged database, `phys_mean_db`, where each CIP level only occurs once for each distinct pharmacological setting for each neuron. Note that, we filter-out columns 14 and 15 from the averaged DB while supplying the first argument to `mergeMultipleCIPsInOne`, which are artifacts of the averaging process and need not be included in the merged database. The second argument is a cell array of pairs of a suffix string and a corresponding list of measures for each of the CIP levels in `phys_mean_db`, in increasing order. The merged `phys_joined_db` looks like this:

```
params_tests_db, averaged tracesets from .../cip_traces_all_axoclamp.txt mult CIP
ans =
    num_params: 12
      tests_db: [1x1 tests_db]
tests_db, averaged tracesets from .../cip_traces_all_axoclamp.txt mult CIP
179 rows in database with 258 columns, and 1 pages.
Column names:
    [  1]    'Cadmium'
    [  2]    'PicroTx'
    [  3]    'Apamin'
    [  4]    'Glycine'
```

```
    [  5]    'KynAcid'
    [  6]    'TTX'
    [  7]    'XE991'
    [  8]    'drug_4AP'
    [  9]    'EBIO'
    [ 10]    'Gabazine'
    [ 11]    'NeuronId'
    [ 12]    'TracesetIndex'
    [ 13]    'PulseISICV_H100pA'
    [ 14]    'PulseIni100msISICV_H100pA'
    [ 15]    'PulseIni100msRest1SpikeRate_H100pA'
    [ 16]    'PulseIni100msRest2SpikeRate_H100pA'
    [ 17]    'PulseIni100msSpikeRate_H100pA'
    [ 18]    'PulseIni100msSpikeRateISI_H100pA'
    ...
    [255]    'PulseSpikeRiseTimeMean_D200pA'
    [256]    'PulseSpikeRiseTimeMode_D200pA'
    [257]    'PulseSpikeRiseTimeSTD_D200pA'
    [258]    'PulseSpontAmpRatio_D200pA'
```

Note how each measure suffix indicate the CIP-level it belongs.

5.6.2 Making a one-row-per-neuron DB from dual CIP-level rows

The following statement uses the `params_tests_db/getDualCIPdb` method to merge rows of depolarizing and hyperpolarizing CIP-levels:

```
>> sdball = getDualCIPdb(dball, depol_tests, hyper_tests,
                         '', 'Hyp100pA')
```

Here, the cell array variables `depol_tests` and `hyper_tests` hold the names of measures to be selected from depolarizing CIP and hyperpolarizing CIP, respectively. The last two arguments define the suffixes to be applied to distinguish the measures from each CIP. The original DB is

```
>> dball
dball
params_tests_db, sim dataset gpsc0501
ans =
    num_params: 10
         props: [0x0 struct]
      tests_db: [1x1 tests_db]
Optional properties of params_tests_db:
ans =
0x0 struct array with no fields.
tests_db, sim dataset gpsc0501
39366 rows in database with 62 columns, and 1 pages.
Column names:
    [ 1]    'NaF'
    [ 2]    'NaP'
    [ 3]    'Kv3'
    [ 4]    'Kv2'
```

```
[ 5]    'Kv4f'
[ 6]    'KCNQ'
[ 7]    'SK'
[ 8]    'CaHVA'
[ 9]    'HCN'
[10]    'pAcip'
[11]    'IniSpontISICV'
[12]    'IniSpontPotAvg'
[13]    'IniSpontSpikeRate'
[14]    'PulseISICV'
[15]    'PulseIni100msISICV'
[16]    'PulseIni100msRest1SpikeRate'
[17]    'PulseIni100msRest2SpikeRate'
[18]    'PulseIni100msSpikeRate'
[19]    'PulseIni100msSpikeRateISI'
[20]    'PulsePotAvg'
[21]    'PulsePotMin'
[22]    'PulsePotSag'
[23]    'PulseSFA'
[24]    'PulseSpikeAmpDecayDelta'
[25]    'PulseSpikeAmpDecayTau'
[26]    'PulseSpikeRate'
[27]    'PulseSpikeRateISI'
[28]    'RecIniSpontPotRatio'
[29]    'RecIniSpontRateRatio'
[30]    'RecSpont1SpikeRate'
[31]    'RecSpont2SpikeRate'
[32]    'RecSpontISICV'
[33]    'RecSpontPotAvg'
[34]    'RecSpontSpikeRate'
[35]    'SpontAmplitude'
[36]    'SpontBaseWidth'
[37]    'SpontDAHPMag'
[38]    'SpontFallTime'
[39]    'SpontHalfVm'
[40]    'SpontHalfWidth'
[41]    'SpontInitTime'
[42]    'SpontInitVm'
[43]    'SpontMaxAHP'
[44]    'SpontMinTime'
[45]    'SpontMinVm'
[46]    'SpontPeakVm'
[47]    'SpontRiseTime'
[48]    'PulseAmplitude'
[49]    'PulseBaseWidth'
[50]    'PulseDAHPMag'
[51]    'PulseFallTime'
[52]    'PulseHalfVm'
[53]    'PulseHalfWidth'
[54]    'PulseInitTime'
```

```
    [55]    'PulseInitVm'
    [56]    'PulseMaxAHP'
    [57]    'PulseMinTime'
    [58]    'PulseMinVm'
    [59]    'PulsePeakVm'
    [60]    'PulseRiseTime'
    [61]    'PulseSpontAmpRatio'
    [62]    'ItemIndex'
Optional properties of tests_db:
ans =
0x0 struct array with no fields.
```

After merging, it becomes

```
>> sdball
sdball
params_tests_db, sim dataset gpsc0501 dual cip
ans =
    num_params: 9
         props: [0x0 struct]
      tests_db: [1x1 tests_db]
Optional properties of params_tests_db:
ans =
0x0 struct array with no fields.
tests_db, sim dataset gpsc0501 dual cip
19683 rows in database with 50 columns, and 1 pages.
Column names:
    [ 1]    'NaF'
    [ 2]    'NaP'
    [ 3]    'Kv3'
    [ 4]    'Kv2'
    [ 5]    'Kv4f'
    [ 6]    'KCNQ'
    [ 7]    'SK'
    [ 8]    'CaHVA'
    [ 9]    'HCN'
    [10]    'RecIniSpontPotRatioHyp100pA'
    [11]    'RecIniSpontRateRatioHyp100pA'
    [12]    'RecSpont1SpikeRateHyp100pA'
    [13]    'RecSpont2SpikeRateHyp100pA'
    [14]    'RecSpontISICVHyp100pA'
    [15]    'RecSpontPotAvgHyp100pA'
    [16]    'ItemIndexHyp100pA'
    [17]    'IniSpontSpikeRate'
    [18]    'PulseIni100msSpikeRate'
    [19]    'PulseIni100msSpikeRateISI'
    [20]    'PulseIni100msISICV'
    [21]    'PulseIni100msRest1SpikeRate'
    [22]    'PulseIni100msRest2SpikeRate'
    [23]    'RecSpont1SpikeRate'
    [24]    'RecSpont2SpikeRate'
```

```
    [25]    'RecIniSpontRateRatio'
    [26]    'IniSpontISICV'
    [27]    'PulseISICV'
    [28]    'RecSpontISICV'
    [29]    'PulseSFA'
    [30]    'PulseSpikeAmpDecayTau'
    [31]    'PulseSpikeAmpDecayDelta'
    [32]    'IniSpontPotAvg'
    [33]    'PulsePotAvg'
    [34]    'RecSpontPotAvg'
    [35]    'RecIniSpontPotRatio'
    [36]    'SpontInitVm'
    [37]    'SpontAmplitude'
    [38]    'SpontMaxAHP'
    [39]    'SpontDAHPMag'
    [40]    'SpontRiseTime'
    [41]    'SpontFallTime'
    [42]    'SpontHalfWidth'
    [43]    'PulseInitVm'
    [44]    'PulseAmplitude'
    [45]    'PulseMaxAHP'
    [46]    'PulseDAHPMag'
    [47]    'PulseRiseTime'
    [48]    'PulseFallTime'
    [49]    'PulseHalfWidth'
    [50]    'ItemIndex'
Optional properties of tests_db:
ans =
0x0 struct array with no fields.
```

6 Visualization

By default, MATLAB prints figures in portrait orientation with a 8x6 aspect ratio, ignoring their size on the screen. The command;

```
>> orient tall
```

changes that behavior to print a full page in portrait orientation. The command;

```
>> orient landscape
```

does the same but rotates the figure to landscape orientation. If you want the printed figure to reflect its current screen size, issue the command;

```
>> set(figurenum, 'PaperPositionMode', 'auto')
```

Figures generated by the plotting system of the PANDORA Toolbox has a special resizing capability. Everytime the figure is resized, it will be drawn from scratch after calculating proper spacing between subplots according to font size. However, in some conditions this may cause other problems, such as crashing MATLAB in figure editing mode or causing loss of manual changes to the figure. To disable the auto-resize function, issue the following command after creating the figure;

```
>> set(figurenum, 'ResizeFcn', '')
```

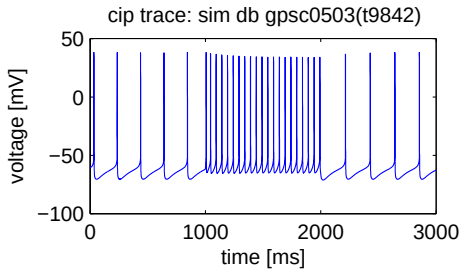


Figure 8: Example trace plot.

Finally, to print the figure, consult your Matlab manual or issue a command such as;

```
>> print -depsc2 figurename.eps
```

See below for specific types of figures you can create.

6.1 Visualizing traces

The `trace`, and its subclasses such as `cip_trace`, has the plot command overloaded to draw the raw trace in Figure 8:

```
>> a_ct = ctFromRows(mbundle, 9842, 100);
>> plot(a_ct)
```

Note that the plot in the figure has been created with a more precise control:

```
>> plotFigure(plotData(a_ct, '', struct('PaperPosition', [0 0 3 2])))
```

6.2 Displaying database contents

The `displayRows` method of `tests_db` can be used to display rows of a database:

```
>> displayRows(sdball, 1:3)
ans =
    'NaF'                   [     250]    [     250]    [      1000]
    'NaP'                   [  2.5000]    [  2.5000]    [    2.5000]
    'Kv3'                   [      15]    [      15]    [        60]
    'Kv2'                   [       3]    [       3]    [         9]
    'Kv4f'                  [       5]    [       5]    [        25]
    'KCNQ'                  [  0.1000]    [  0.1000]    [    0.0100]
    'SK'                    [  8.5000]    [  8.5000]    [        17]
    'CaHVA'                 [      10]    [      10]    [        10]
    'HCN'                   [      30]    [       3]    [        30]
    'pAcip'                 [    -100]    [    -100]    [       100]
    'IniSpontISICV'         [  0.0027]    [  0.0027]    [9.1376e-04]
...
```

Note that, in the output, database rows appear as columns, and database columns appear as rows. See in above Section 5.4 for more example outputs from `displayRows`.

`displayRows` returns a cell array of column names justapoxed to a matrix of values. This cell array is intended for display on the screen and for generating reports. The `displayRowsTEX` method uses output from `displayRows` to generate a LaTeX table that can be printed or converted to PDF:

```
>> tex_string =
     displayRowsTEX(a_db(:, rows),
                    'Selected rows indicating fast spiking neurons',
                    struct('height', '!'));
>> string2File(tex_string, 'fast_spiking.tex');
```

With this the LaTeX code to generate a table with the given caption is saved in a text file called `fast_spiking.tex`. This file can then be included from a regular LaTeX document to generate PDF output. See a LaTeX manual on how to do that.

An alternative to `displayRows` is using the `tests_db/rows2Struct` method:

```
>> s = rows2Struct(dball_full, 54023)
s =
                            NaF: 500
                            NaP: 2
                            Kv2: 1
                            Kv3: 10
                           Kv4f: 40
                           KCNQ: 2
                             SK: 4
                          CaHVA: 0.0300
                            HCN: 1
                          trial: 7739
                          pAcip: 100
                 IniSpontISICV: NaN
                IniSpontPotAvg: NaN
              IniSpontSpikeRate: 0
           IniSpontSpikeRateISI: 0
                     PulseISICV: 0.0265
              PulseIni100msISICV: 0.0584
    PulseIni100msRest1SpikeRate: 40.0089
    PulseIni100msRest2SpikeRate: 40.0178
         PulseIni100msSpikeRate: 50
      PulseIni100msSpikeRateISI: 43.5256
                  PulsePotAvg: -57.3737
                  PulsePotMin: NaN
                  PulsePotSag: NaN
                     PulseSFA: 1.1698
...
```

This method returns the database contents as a structure array. It is more natural for programming interfaces to use the database contents in a structure array than a cell array. The database columns become field names in the the structure. If multiple rows are requested, the displayed output would not contain the values. The desired row can be reached via indexing (e.g., `s(1)`). For instance, analysis in `cip_trace/getProfileAllSpikes` method is done using this method for getting statistics from the `spikes_db` databases.

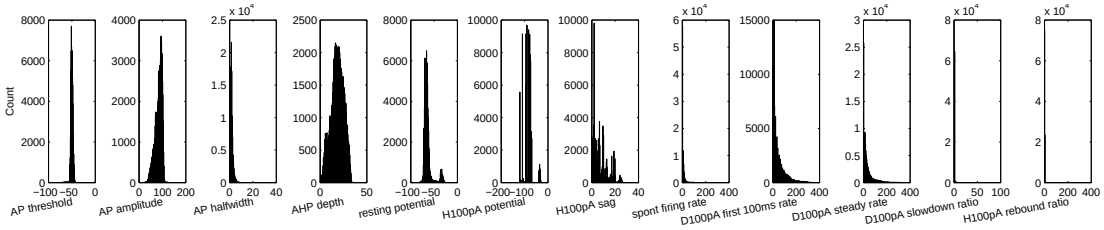


Figure 9: Example measure distribution plot.

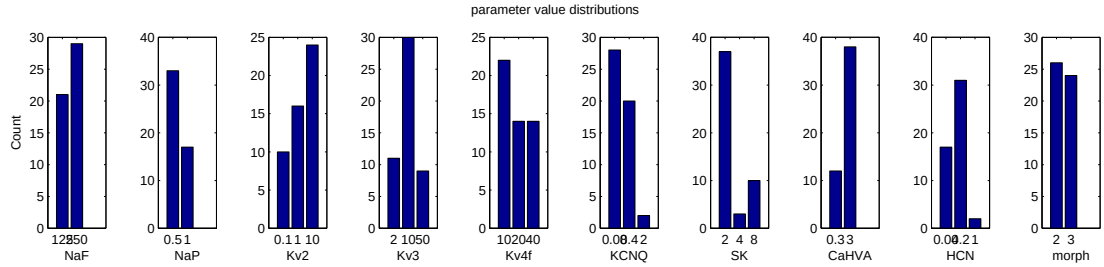


Figure 10: Example parameter distribution plot.

6.3 Plotting all measure histograms

For plotting all measure histograms in a DB, the following method of `tests_db` creates the horizontal stack plot in Figure 9:

```
>> mp = plotTestsHistsMatrix(renamed_model_db(:, 2:end), '',
                             struct('orient', 'x', 'quiet', 1, 'border', [0 0.05 0 0])))
>> plotFigure(mp);
```

6.4 Plotting all parameter histograms

For plotting all parameter histograms in a DB, the following method of `params_tests_db` creates the horizontal stack plot in Figure 10:

```
>> plotFigure(plotParamsHists(sdball));
```

6.5 Plotting database statistics

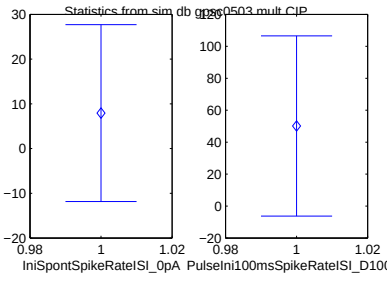
The `stats_db` object allows keeping statistical information obtained from a database. Statistics are calculated using one of the `tests_db` converter methods, such as `statsAll`, `statsMeanStd`, etc.:

```
>> my_stats = statsMeanStd(my_db(:, {'IniSpontSpikeRate', 'PulseSpikeRate'}));
```

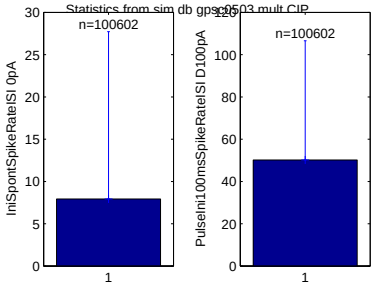
Then, the statistics can be plotted with diamonds indicating the mean and symmetric errorbars indicating upper and lower extensions (SE or Std):

```
>> plot(my_stats);
```

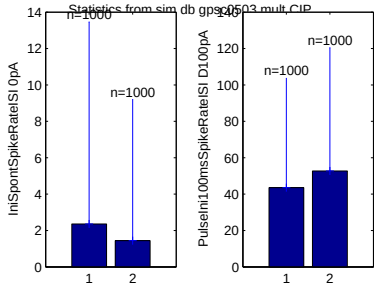
which is equivalent to:



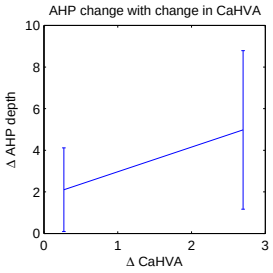
(a) Default visualization for `stats_db`.



(b) Optional `plot_bars` visualization for `stats_db`.



(c) Comparing multiple `stats_db`s with `plot_bars`.



(d) Plotting parameter-measure variation with `plotVar`.

Figure 11: Plotting statistics for two selected measures.

```
>> plotFigure(plot_abstract(my_stats));
```

An alternative plotting form is using filled bars with extending errorbars:

```
>> plotFigure(plot_bars(my_stats));
```

which are both seen in Figure 11 (a) and (b). The `plot_bars` method is better suited for comparing statistics from multiple `stats_db`s coming from different sources. In this case, we take the first 1000 rows from the DB as a subset, and compare it with the statistics from the second 1000 rows:

```
>> compared_two_subset_stats =
     compareStats(statsAll(mbundle.joined_db(1:1000,
                                             {'IniSpontSpikeRateISI_0pA',
                                              'PulseIni100msSpikeRateISI_D100pA'})),
                  statsAll(mbundle.joined_db(1001:2000,
                                             {'IniSpontSpikeRateISI_0pA',
                                              'PulseIni100msSpikeRateISI_D100pA'})));
```

The combined statistics object can then be fed into `plot_bars` as seen in Figure 11 (c):

```
>> plotFigure(plot_bars(compared_two_subset_stats));
```

6.6 Plotting parameter-measure variations

To plot the variation of a measure with a parameter the `plotVar` method of `stats_db` can be used to achieve Figure 11 (d):

```
>> plotFigure(plotVar(a_stats_db, 'CaHVA', 'AHP_depth', 'AHP change with change in CaHVA',
                      struct('quiet', 1, 'PaperPosition', [0 0 3 3])))
```

To plot all parameter-measure variations, the `plotVarMatrix` method of `stats_db` can be used (see Figure 12). `plotVarMatrix` requires the a `p_stats` array of `stats_db` objects that hold the mean and standard error information (or possibly other statistics) for each of the possible parameter-measure combinations. The `p_stats` array can be created using the `paramsTestsHistsStats` method, which in turn requires the `p_t3ds` array of 3-dimensional databases each of which contain effects of a parameter invariant of other parameters. The `p_t3ds` array can be created using the `params_tests_db/invarParams` method. The sequence of commands[4] is then becomes:

```
>> p_t3ds = invarParams(noNaNRows(sdball))
>> [pt_hists, p_stats] = paramsTestsHistsStats(p_t3ds)
>> ap = plotVarMatrix(p_stats)
>> plotFigure(ap)
```

This will create a matrix plot with as many columns as parameters and as many rows as measures in the `sdball` object. It may be difficult to read if `sdball` contains large number of measures. One can divide the measures into two plots with the following sequence of commands

[4]The `noNaNRows` function is required to filter out any rows containing `NaN` values in measurements before running statistic functions. Otherwise, the statistics functions (such as `mean` and `std`) will eliminate `NaN` values within each database column automatically, scrambling the row order and loosing the association with parameters.

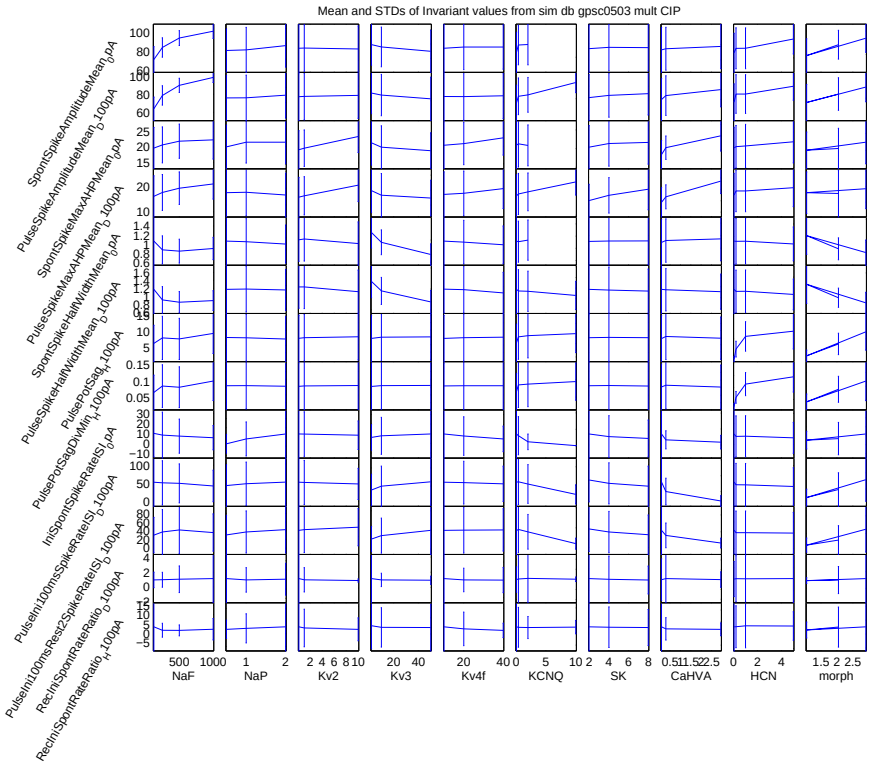


Figure 12: Example parameter-measure variation statistics plot.

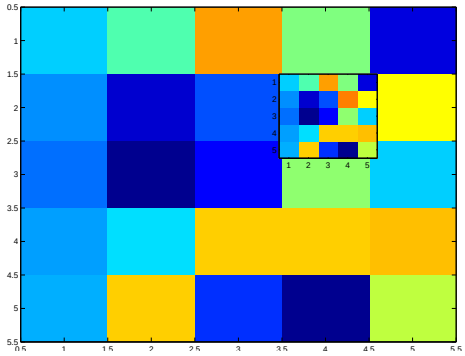


Figure 13: Creating insets in figures.

```
>> sdb1 = sdball(:, [1:9, 10:35])
>> sdb2 = sdball(:, [1:9, 36:49])
```

by choosing all parameters in both DBs, but only some measures for each. Then, the plots can be created for each DB by issuing

```
>> p1_t3ds = invarParams(sdb1)
>> p2_t3ds = invarParams(sdb2)
>> [pt1_hists, p1_stats] = paramsTestsHistsStats(p1_t3ds)
>> [pt2_hists, p2_stats] = paramsTestsHistsStats(p2_t3ds)
>> ap1 = plotVarMatrix(p1_stats)
>> plotFigure(ap1)
>> ap2 = plotVarMatrix(p2_stats)
>> plotFigure(ap2)
```

6.7 Insets

See Figure 13 which was created with the following set of commands.

```
>> im_p = plot_abstract({50 * rand(5)}, {}, '', {}, 'image');
>> plotFigure(plot_absolute([im_p, im_p], [0 0 1 1; 0.5 0.5 0.3 0.3]))
```

6.8 Generating a report comparing two databases

Most commonly a database of physiology neurons need to be compared to a database of a large body of simulation neurons and find best matches. One may need to see the match quality for a number of best matching candidates for each of the physiology neurons. Assuming multiple plots and tables are required to do a fair job of comparing a neuron to a thousands of simulation neurons, it becomes a difficult job to do this manually. An automatic report generation system has been built into the system for this purpose.

Currently a LaTeX document is created that needs to be included in either a proper

LaTeX document, or included in a LyX document.[5] The including document should provide the context in which the included part becomes meaningful. The report contains a set of tables and figures with proper captions. The table of contents, list of figures and list of tables facilities of LaTeX becomes useful to make the automatically generated document easily readable.

The report can be created with the command

```
>> tex_string = rankVsAllDB(sdb, phys_sdb,
                            fileset, phys_fileset);
```

where the simulation database `sdb` is searched for best matches to each row of physiology database `phys_sdb`. The dataset for each is provided for the report to contain raw data associated with best matches. The obtained report contained in `tex_string` can be saved as a ASCII LaTeX file with

```
>> string2File(tex_string, 'myreport.tex');
```

References

Appendices

A Function Reference

See Section 1.6 on how to get online help about the software within MATLAB.

A.1 Class `chans_db`

A.1.1 Constructor `chans_db/chans_db`

Summary: A database of channel activation and kinetics.

Usage:

```
a_chans_db = chans_db(data, col_names, channel_info, id, props)
```

Description: This is a subclass of tests_db. Channel tables can be imported from Genesis using the utils/chanTables2DB script.

Parameters:

`data`**:** Database contents.

`col_names`**:** The channel variable names.

[5] This document is prepared using the LyX document preparation system [**?**] which uses the LaTeX 2_ε typesetting language [**?**]. LyX is copyrighted by Matthias Ettrich and covered by the terms of the GNU General Public License (GPL), and LaTeX 2_ε is copyrighted by D. E. Knuth and the Free Software Foundation, Inc. and is covered by both the TeX copyright and the GNU GPL.

`channel_info`: Structure that holds scalar data elements such as Gbar.

`id`: An identifying string.

`props`: A structure with any optional properties.

Returns a structure object with the following fields:

tests_db, channel_info, props.

See also: `tests_db` (p. 256), `chanTables2DB` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2007/06/26

A.1.2 Method `chans_db/display`

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.1.3 Method `chans_db/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.1.4 Method `chans_db/plotAllInf`

Summary: Plots the steady-state (infinity) response of all channels.

Usage:

```
a_plot = plotAllInf(a_chans_db, title_str, props)
```

Parameters:

`a_chans_db`: a chans_db

`title_str`: Plot title.

`props`: A structure with any optional properties.
(rest passed to matrixPlots.)

Returns:

a_plot: A plot_abstract object that can be visualized.

See also: `trace` (p. 317), `trace/plot` (p. 328), `plot_abstract` (p. 180)

Author: Cengiz Gunay <cgunay@emory.edu>, 2007/03/05

A.1.5 Method `chans_db/plotAllVars`

Summary: Plot all channel variables by grouping activation and time constant curves per channel.

Usage:

```
a_plot = plotAllVars(a_chans_db, title_str, props)
```

Parameters:

`a_chans_db`: A chans_db describing channel variables.

`id`: String that identify the source of the tables structure.

`props`: A structure with any optional properties.
(rest passed to plot_abstract.)

Returns:

a_plot: A plot_abstract object that can be visualized.

See also: `trace` (p. 317), `trace/plot` (p. 328), `plot_abstract` (p. 180)

Author: Cengiz Gunay <cgunay@emory.edu>, 2007/03/05

A.1.6 Method `chans_db/plotGateVars`

Summary: Plot given channel gate variables of the same channel superposed.

Usage:

```
a_plot = plotGateVars(a_chans_db, chan_name, gate_subnames, title_str, props)
```

Parameters:

`a_chans_db`: A chans_db describing channel variables.

`chan_name`: Name of channel that make up the stem of variable names.

`gate_subnames`: Gate names of the channel.

`title_str`: (Optional) A string to be concatanated to the title.

`props`: A structure with any optional properties.
`usePowers`: Use the gate powers, Luke.
(rest passed to plot_abstract.)

Returns:

a_plot: A plot_abstract object that can be visualized.

See also: `trace` (p. 317), `trace/plot` (p. 328), `plot_abstract` (p. 180)

Author: Cengiz Gunay <cgunay@emory.edu>, 2007/07/01

A.1.7 Method `chans_db/plotInf`

Summary: Plot the product of minf variables and the gmax of the given channel.

Usage:

```
a_plot = plotInf(a_chans_db, chan_name, gate_subnames, title_str, props)
```

Parameters:

`a_chans_db`: A chans_db describing channel variables.

`chan_name`: Name of channel that make up the stem of variable names.

`gate_subnames`: Gate names of the channel.

`title_str`: (Optional) A string to be concatanated to the title.

`props`: A structure with any optional properties. (rest passed to plot_abstract.)

Returns:

a_plot: A plot_abstract object that can be visualized.

See also: `trace` (p. 317), `trace/plot` (p. 328), `plot_abstract` (p. 180)

Author: Cengiz Gunay <cgunay@emory.edu>, 2007/07/01

A.1.8 Method `chans_db/set`

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.2 Class `cip_trace`

A.2.1 Constructor `cip_trace/cip_trace`

Summary: A trace with a current injection pulse (CIP).

Usage:

```
obj = cip_trace(datasrc, dt, dy, pulse_time_start, pulse_time_width, id, props)
```

Parameters:

`datasrc`: A vector of data points containing the spike shape.

`dt`: Time resolution [s].

`dy`: y-axis resolution [ISI (V, A, etc.)]

`pulse_time_start, pulse_time_width:` Start and width of the pulse [dt]

`id:` Identification string.

`props:` A structure with any optional properties, such as:

`trace_time_start:` Samples in the beginning to discard [dt] (see trace for more)

Returns a structure object with the following fields:

trace, pulse_time_start, pulse_time_width, props.

See also: `trace` (p. 317), `spikes` (p. 227), `spike_shape` (p. 215), `period` (p. 162)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/07/30

A.2.2 Method `cip_trace/calcPulsePotAvg`

Summary: Calculates the average potential value of the CIP period of the cip_trace, t.

Usage:

```
avg_val = calcPulsePotAvg(t)
```

Parameters:

`t:` A cip_trace object.

Returns:

avg_val: The avg value [dy].

See also: `period` (p. 162), `trace` (p. 317), `trace/calcAvg` (p. 321)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/25

A.2.3 Method `cip_trace/calcPulsePotSag`

Summary: Calculates the minimal sag and sag amount of the CIP period of the cip_trace, t.

Usage:

```
[min_val, min_idx, sag_val] = calcPulsePotSag(t)
```

Description: The minimal sag is the minimal potential value of the first half of the CIP period. The sag amount is calculated by comparing this to the steady-state value at the end of the CIP period.

Parameters:

`t:` A cip_trace object.

Returns:

min_val: The min value [dy]. min_idx: The index of the min value [dt]. sag_val: The sag amount [dy].

See also: `period` (p. 162), `trace` (p. 317), `trace/calcMin` (p. 322)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/25

A.2.4 Method `cip_trace/calcRecSpontPotAvg`

Summary: Calculates the average potential value of the recovery period of the cip_trace, t.

Usage:

```
avg_val = calcRecSpontPotAvg(t)
```

Parameters:

`t:` A cip_trace object.

Returns:

avg_val: The avg value [dy].

See also: `period` (p. 162), `trace` (p. 317), `trace/calcAvg` (p. 321)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/25

A.2.5 Method `cip_trace/display`

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.2.6 Method `cip_trace/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.2.7 Method `cip_trace/getBurstResults`

Summary: Calculate test results related to Burst behavior.

Usage:

```
results = getRateResults(a_cip_trace, a_spikes)
```

Parameters:

`a_cip_trace`**:** A cip_trace object.

`a_spikes`**:** A spikes object.

Returns:

results: A structure associating test names with result values.

See also: `cip_trace` (p. 56), `spikes` (p. 227), `spike_shape` (p. 215)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/30, Tom Sangrey

---

A.2.8 Method `cip_trace/getCIPResults`

Summary: Calculate test results about CIP protocol.

Usage:

```
results = getCIPResults(a_cip_trace, a_spikes)
```

Parameters:

`a_cip_trace`**:** A cip_trace object.

`a_spikes`**:** A spikes object.

Returns:

results: A structure associating test names with result values.

See also: `cip_trace` (p. 56), `spikes` (p. 227), `spike_shape` (p. 215)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/30

A.2.9 Method `cip_trace/getProfileAllSpikes`

Summary: Creates a cip_trace_allspikes_profile object by collecting test results of a cip_trace, analyzing each individual spike.

Usage:

```
profile_obj = getProfileAllSpikes(a_cip_trace)
```

Description: Analyzes the spontaneous (periodIniSpont), pulse (periodPulse) and the recovery (periodRecSpont) periods separately and produces spike shape distribution results. Rate and CIP measurements are added to these.

Parameters:

`a_cip_trace`**:** A cip_trace object.

Returns:

profile_obj: A cip_trace_allspikes_profile object.

See also: `cip_trace` (p. 56), `cip_trace_allspikes_profile` (p. 70)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/04/26

A.2.10 Method `cip_trace/getPulseSpike`

Summary: Convert a spike in the CIP period to a spike_shape object.

Usage:

```
obj = getPulseSpike(trace, spikes, spike_num, props)
```

Description: Creates a spike_shape object from desired spike. Calls trace/getSpike method.

Parameters:

`trace`**:** A trace object.

`spikes`**:** (Optional) A spikes object obtained from trace, calculated automatically if given as [].

`spike_num`**:** The index of spike to extract.

`props`**:** A structure with any optional properties passed to getSpike.

Example:

```
Get 2nd pulse spike and plot it:
» plotFigure(plotResults(getPulseSpike(t, [], 2)))
```

See also: `spike_shape` (p. 215), `trace/getSpike` (p. 325)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/04/19

A.2.11 Method `cip_trace/getRateResults`

Summary: Calculate test results related to spike rate.

Usage:

```
results = getRateResults(a_cip_trace, a_spikes)
```

Parameters:

`a_cip_trace`: A cip_trace object.

`a_spikes`: A spikes object.

Returns:

results: A structure associating test names with result values.

See also: `cip_trace` (p. 56), `spikes` (p. 227), `spike_shape` (p. 215)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/30

A.2.12 Method `cip_trace/getRecSpontSpike`

Summary: Convert a spike in the CIP period to a spike_shape object.

Usage:

```
obj = getRecSpontSpike(trace, spikes, spike_num, props)
```

Description: Creates a spike_shape object from desired spike.

Parameters:

`trace`: A trace object.

`spikes`: A spikes object on trace.

`spike_num`: The index of spike to extract.

See also: `spike_shape` (p. 215)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/05/08

A.2.13 Method `cip_trace/getResults`

Summary: Calculate test results given a_spikes object.

Usage:

```
results = getResults(a_cip_trace, a_spikes)
```

Parameters:

`a_cip_trace`**:** A cip_trace object.

`a_spikes`**:** A spikes object.

Returns:

results: A structure associating test names with result values.

See also: `cip_trace` (p. 56), `spikes` (p. 227), `spike_shape` (p. 215)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.2.14 Method `cip_trace/measureNames`

Summary: Returns taxonomy of measurements collected by cip_trace.

Usage:

```
measures = measureNames(a_cip_trace)
```

Description: This is a static method, in the sense that it does not need the object passed as argument. Therefore it can be called directly by using the default constructor; e.g., measureNames(cip_trace). The measure names are required for merging columns of a database generated by profiling these objects.

Parameters:

`a_cip_trace`**:** A cip_trace object. It can be created by the the default constructor 'cip_trace'.

Returns:

measures: A structure with cell arrays of types of measures, and measure names inside.

See also: `getResults` (p. **??**), `getProfileAllSpikes` (p. **??**), `mergeMultipleCIPsInOne` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/07/30

A.2.15 Method `cip_trace/periodIniSpont`

Summary: Returns the initial spontaneous activity period of cip_trace, t.

Usage:

```
the_period = periodIniSpont(t)
```

Parameters:

`t`: A trace object.

Returns:

the_period: A period object.

See also: `period` (p. 162), `cip_trace` (p. 56), `trace` (p. 317)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/25

A.2.16 Method `cip_trace/periodPulse`

Summary: Returns the CIP period of cip_trace, t.

Usage:

```
the_period = periodPulse(t)
```

Parameters:

`t`: A trace object.

Returns:

the_period: A period object.

See also: `period` (p. 162), `cip_trace` (p. 56), `trace` (p. 317)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/25

A.2.17 Method `cip_trace/periodPulseHalf1`

Summary: Returns the first half of the CIP period of cip_trace, t.

Usage:

```
the_period = periodPulseHalf1(t)
```

Parameters:

`t`: A trace object.

Returns:

the_period: A period object.

See also: period (p. 162), cip_trace (p. 56), trace (p. 317)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/25

A.2.18 Method cip_trace/periodPulseIni100ms

Summary: Returns the first 100ms of the CIP period of cip_trace, t.

Usage:

```
the_period = periodPulseIni100ms(t)
```

Parameters:

t: A trace object.

Returns:

the_period: A period object.

See also: period (p. 162), cip_trace (p. 56), trace (p. 317)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/25

A.2.19 Method cip_trace/periodPulseIni100msRest1

Usage:

```
the_period = periodPulseIni50msRest1(t)
```

Parameters:

t: A trace object.

Returns:

the_period: A period object.

See also: period (p. 162), cip_trace (p. 56), trace (p. 317)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/25

A.2.20 Method `cip_trace/periodPulseIni100msRest2`

Usage:

```
the_period = periodPulseIni50msRest2(t)
```

Parameters:

`t`: A trace object.

Returns:

the_period: A period object.

See also: `period` (p. 162), `cip_trace` (p. 56), `trace` (p. 317)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/25

A.2.21 Method `cip_trace/periodPulseIni50ms`

Summary: Returns the first 50ms of the CIP period of cip_trace, t.

Usage:

```
the_period = periodPulseIni50ms(t)
```

Parameters:

`t`: A trace object.

Returns:

the_period: A period object.

See also: `period` (p. 162), `cip_trace` (p. 56), `trace` (p. 317)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/25

A.2.22 Method `cip_trace/periodPulseIni50msRest1`

Summary: Returns the first half of the rest after the first 50ms of the CIP period of cip_trace, t.

Usage:

```
the_period = periodPulseIni50msRest1(t)
```

Parameters:

`t`: A trace object.

Returns:

the_period: A period object.

See also: period (p. 162), cip_trace (p. 56), trace (p. 317)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/25

A.2.23 Method cip_trace/periodPulseIni50msRest2

Summary: Returns the second half of the rest after the first 50ms of the CIP period of cip_trace, t.

Usage:

```
the_period = periodPulseIni50msRest2(t)
```

Parameters:

t: A trace object.

Returns:

the_period: A period object.

See also: period (p. 162), cip_trace (p. 56), trace (p. 317)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/25

A.2.24 Method cip_trace/periodRecSpont

Summary: Returns the recovery spontaneous activity period of cip_trace, t.

Usage:

```
the_period = periodRecSpont(t)
```

Parameters:

t: A trace object.

Returns:

the_period: A period object.

See also: period (p. 162), cip_trace (p. 56), trace (p. 317)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/25

A.2.25 Method `cip_trace/periodRecSpont1`

Summary: Returns the first half of the recovery spontaneous activity period of cip_trace, t.

Usage:

```
the_period = periodRecSpont1(t)
```

Parameters:

`t:` A trace object.

Returns:

the_period: A period object.

See also: `period` (p. 162), `cip_trace` (p. 56), `trace` (p. 317)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/25

A.2.26 Method `cip_trace/periodRecSpont2`

Summary: Returns the second half of the recovery spontaneous activity period of cip_trace, t.

Usage:

```
the_period = periodRecSpont2(t)
```

Parameters:

`t:` A trace object.

Returns:

the_period: A period object.

See also: `period` (p. 162), `cip_trace` (p. 56), `trace` (p. 317)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/25

A.2.27 Method `cip_trace/periodRecSpontIniPeriod`

Usage:

```
the_period = periodRecSpont(t)
```

Parameters:

`t:` A trace object.

`iniPeriod`: the time following pulse offset that is kept, the rest of the time is ignored.

Returns:

the_period: A period object.

See also: `period` (p. 162), `cip_trace` (p. 56), `trace` (p. 317)

Author: Cengiz Gunay <cgunay@emory.edu>,Tom Sangrey 2006/01/26

A.2.28 Method `cip_trace/periodRecSpontRestPeriod`

Usage:

```
the_period = periodRecSpont(t)
```

Parameters:

`t`: A trace object.

`iniPeriod`: the time following pulse offset that is ignored. The rest of the time is kept

Returns:

the_period: A period object.

See also: `period` (p. 162), `cip_trace` (p. 56), `trace` (p. 317)

Author: Cengiz Gunay <cgunay@emory.edu>,Tom Sangrey 2006/01/26

A.2.29 Method `cip_trace/plotData`

Summary: Plots a trace by calling trace/plotData but also adds optionaldecorations.

Usage:

```
a_plot = plotData(t, title_str, props)
```

Description: If t is a vector of traces, returns a vector of plot objects.

Parameters:

`t`: A trace object.

`title_str`: (Optional) String to append to plot title.

`props`: A structure with any optional properties.

`stimBar`: If true, put a bar indicating the CIP duration. (rest passed to trace/plotData)

Returns:

a_plot: A plot_abstract object that can be visualized.

See also: `trace` (p. 317), `trace/plot` (p. 328), `plot_abstract` (p. 180)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/17

A.2.30 Method `cip_trace/plot_abstract`

Summary: Plots a trace by calling plotData.

Usage:

```
a_plot = plot_abstract(t, title_str, props)
```

Description: If t is a vector of traces, returns a vector of plot objects.

Parameters:

`t`: A trace object.

`title_str`: (Optional) String to append to plot title.

`props`: A structure with any optional properties.

`timeScale`: 's' for seconds, or 'ms' for milliseconds.
(rest passed to plot_abstract.)

Returns:

a_plot: A plot_abstract object that can be visualized.

See also: `trace` (p. 317), `trace/plot` (p. 328), `plot_abstract` (p. 180)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/17

A.2.31 Method `cip_trace/set`

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.2.32 Method `cip_trace/spikes`

Summary: Convert cip_trace to spikes object for spike timing calculations.

Usage:

```
obj = spikes(trace, plotit)
```

Description: Creates a spikes object by finding the spikes in the three separate periods, initial spontaneous activity period, CIP period, and final recovery period.

Parameters:

`trace`: A trace object.

`plotit`: If non-zero, a plot is generated for showing spikes found (optional).

See also: `spikes` (p. 227), `period` (p. 162)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/25

A.2.33 Method `cip_trace/subsref`

Summary: Defines generic indexing for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.3 Class `cip_trace_allspikes_profile`

A.3.1 Constructor `cip_trace_allspikes_profile/cip_trace_allspikes_profile`

Summary: Creates and collects test results of a cip_trace.

Usage:

```
obj = cip_trace_allspikes_profile(a_cip_trace, a_spikes, a_spont_spike_shape, results, id,
props)
```

Description: This is a subclass of results_profile. It is made to be used from subclass constructors.

Parameters:

`a_cip_trace`: A cip_trace object.

`a_spikes`: A spikes object.

`spont_spikes_db, pulse_spikes_db, recov_spikes_db`: tests_dbs with spontaneous, pulse and recovery period spike info.

`results_obj`: A results_profile object with test results.

`id`: Identification string.

`props`: A structure with any optional properties.

Returns a structure object with the following fields:

trace, spikes, spont_spikes_db, pulse_spikes_db, recov_spikes_db, props

See also: `cip_trace` (p. 56), `spikes` (p. 227), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/05/04

A.3.2 Method `cip_trace_allspikes_profile/display`

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.3.3 Method `cip_trace_allspikes_profile/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.3.4 Method `cip_trace_allspikes_profile/plotRowSpontSpikeAnal`

Summary: Creates a row of plots that show spontaneous spikes, starting from the whole trace, zooming into the individual spike.

Usage:

```
a_plot = plotRowSpontSpikeAnal(prof, title_str)
```

Parameters:

`prof`: A cip_trace_allspikes_profile object.

`title_str`: (Optional) String to append to plot title.

Returns:

a_plot: A plot_abstract object that can be visualized.

See also: `trace` (p. 317), `cip_trace` (p. 56), `spike_shape/plotCompareMethodsSimple` (p. 224), `plot_abstract` (p. 180)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/05/23

A.3.5 Method `cip_trace_allspikes_profile/set`

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.4 Class `cip_trace_profile`

A.4.1 Constructor `cip_trace_profile/cip_trace_profile`

Summary: Creates and collects test results of a cip_trace.

Description: The first usage is fully customizable to be used from subclass constructors. The second usage generates the spikes and spont_spike_shape objects, and collects some generic test results from them.

Parameters:

`data_src`**:** The trace column OR the filename.

`dt`**:** Time resolution [s]

`dy`**:** y-axis resolution [ISI (V, A, etc.)]

`pulse_time_start, pulse_time_width`**:** Start and width of the pulse [dt]

`id`**:** Identification string.

`props`**:** See trace object.

Returns a structure object with the following fields:

trace, spikes, spont_spike_shape, results, id, props.

See also: `cip_trace` (p. 56), `spikes` (p. 227), `spike_shape` (p. 215)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/25

A.4.2 Method `cip_trace_profile/display`

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.4.3 Method `cip_trace_profile/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.4.4 Method `cip_trace_profile/plot`

Summary: Plots a cip_trace_profile object.

Usage:

```
h = plot(t)
```

Description: Plots contents of this object.

Parameters:

`t`: A cip_trace_profile object.

Returns:

h: Plot handle(s).

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/15

A.4.5 Method `cip_trace_profile/set`

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.4.6 Method `cip_trace_profile/subsref`

Summary: Defines generic indexing for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.5 Class `cip_traces_dataset`

A.5.1 Constructor `cip_traces_dataset/cip_traces_dataset`

Summary: Dataset of cip_traces objects, each with varying cip magnitudes.

Usage:

```
obj = cip_traces_dataset(ts, cipmag, id, props)
```

Description: This is a subclass of params_tests_fileset.

Parameters:

`ts`: A cell array of cip_traces objects.

`cipmag`: A single cip magnitude to trace take from objects.

`id`: An identification string for the whole dataset.

`props`: A structure with any optional properties passed to cip_trace_profile.

Returns a structure object with the following fields:

params_tests_dataset, cipmag, props (see above).

See also: `cip_traces` (p. **??**), `params_tests_fileset` (p. 157), `params_tests_db` (p. 138)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/30

A.5.2 Method `cip_traces_dataset/cip_trace_profile`

Summary: Loads a raw cip_trace_profile given a index to this dataset.

Usage:

```
a_cip_trace_profile = cip_trace_profile(dataset, index)
```

Parameters:

`dataset`: A params_tests_dataset.

`index`: Index of file in dataset.

Returns:

a_cip_trace_profile: A cip_trace_profile object.

See also: `cip_trace_profile` (p. 72), `params_tests_dataset` (p. 132)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.5.3 Method `cip_traces_dataset/display`

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.5.4 Method `cip_traces_dataset/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.5.5 Method `cip_traces_dataset/getItemParams`

Usage:

```
params_row = getParams(dataset, index)
```

Parameters:

`dataset`: A params_tests_dataset.

`index`: Index of item in dataset.

Returns:

params_row: Parameter values in the same order of paramNames

See also: `itemResultsRow` (p. **??**), `params_tests_dataset` (p. 132), `paramNames` (p. **??**), `testNames` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/12/06

A.5.6 Method `cip_traces_dataset/loadItemProfile`

Summary: Loads a profile object from a raw data item in the dataset.

Usage:

```
a_profile = loadItemProfile(dataset, index)
```

Description: Subclasses should overload this function to load the specific profile object they desire. The profile class should define a getResults method which is used in the itemResultsRow method.

Parameters:

`dataset`: A params_tests_dataset.

`index`: Index of item in dataset.

Returns:

a_profile: A profile object that implements the getResults method.

See also: `itemResultsRow` (p. **??**), `params_tests_dataset` (p. 132), `paramNames` (p. **??**), `testNames` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.5.7 Method `cip_traces_dataset/paramNames`

Summary: Returns the only parameter, 'pAcip,' for this fileset.

Usage:

```
param_names = paramNames(fileset)
```

Description: Looks at the filename of the first file to find the parameter names.

Parameters:

`fileset`: A params_tests_fileset.

Returns:

params_names: Cell array with ordered parameter names.

See also: `params_tests_fileset` (p. 157), `paramNames` (p. **??**), `testNames` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/12/06

A.5.8 Method `cip_traces_dataset/set`

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.5.9 Method `cip_traces_dataset/subsref`

Summary: Defines generic indexing for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.6 Class `cip_traceset`

A.6.1 Constructor `cip_traceset/cip_traceset`

Summary: A traceset with varying cip magnitudes from a single cip_traces object.

Usage:

```
obj = cip_traceset(ct, cip_mags, dy, props)
```

Description: This is a subclass of params_tests_fileset. This traceset can create a mini-database form a single cip_traces object. The list contains cip_mags. cip_traceset_dataset should be used to load multiple cip_traceset objects.

Parameters:

`ct`: A cip_traces object.

`cip_mags`: An array of cip magnitudes to select from the object.

`dy`: y-axis resolution, [V] or [A] (default=1e-3).

`props`: A structure with any optional properties.

`offsetPotential`: Add this to physiology trace as compensation.

Returns a structure object with the following fields:

params_tests_dataset, ct, props (see above).

See also: `cip_traces` (p. **??**), `params_tests_fileset` (p. 157), `params_tests_db` (p. 138)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/30

A.6.2 Method `cip_traceset/cip_trace_profile`

Summary: Loads a raw cip_trace_profile given an index in this traceset.

Usage:

```
a_cip_trace_profile = cip_trace_profile(traceset, index)
```

Parameters:

`traceset`: A cip_traceset.

`index`: Index of item in traceset.

Returns:

a_cip_trace_profile: A cip_trace_profile object.

See also: `cip_trace_profile` (p. 72), `params_tests_dataset` (p. 132)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.6.3 Method `cip_traceset/display`

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.6.4 Method `cip_traceset/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.6.5 Method `cip_traceset/getItemParams`

Usage:

```
params_row = getParams(dataset, index)
```

Parameters:

`dataset`: A params_tests_dataset.

`index`: Index of item in dataset.

`a_profile`: A profile object for the item (optional).

Returns:

params_row: Parameter values in the same order of paramNames

See also: `itemResultsRow` (p. **??**), `params_tests_dataset` (p. 132), `paramNames` (p. **??**), `testNames` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/12/06

A.6.6 Method `cip_traceset/loadItemProfile`

Summary: Loads a profile object from a raw data item in the dataset.

Usage:

```
a_profile = loadItemProfile(dataset, index)
```

Description: Subclasses should overload this function to load the specific profile object they desire. The profile class should define a getResults method which is used in the itemResultsRow method.

Parameters:

`dataset`: A params_tests_dataset.

`index`: Index of item in dataset.

Returns:

a_profile: A profile object that implements the getResults method.

See also: `itemResultsRow` (p. **??**), `params_tests_dataset` (p. 132), `paramNames` (p. **??**), `testNames` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.6.7 Method `cip_traceset/paramNames`

Summary: Returns the only parameter, 'pAcip,' for this traceset.

Usage:

```
param_names = paramNames(traceset)
```

Description: Looks at the filename of the first file to find the parameter names.

Parameters:

`traceset`: A cip_traceset.

Returns:

params_names: Cell array with ordered parameter names.

See also: `params_tests_dataset` (p. 132), `paramNames` (p. **??**), `testNames` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/12/06

A.7 Class `cip_traceset_dataset`

A.7.1 Constructor `cip_traceset_dataset/cip_traceset_dataset`

Summary: Dataset of multiple cip magnitudes from cip_traces objects .

Usage:

```
obj = cip_traceset_dataset(cts, cip_mags, dy, id, props)
```

Description: This is a subclass of params_tests_dataset. Designed to extract a trace for each cip magnitude from the cip_traceset objects contained. Uses cip_traceset objects to extract multiple traces from each cip_traces object.

Parameters:

`cts`: Array or cell array of cip_traces objects.

`cip_mags`: An array of cip magnitudes to select from each cip_traces object.

`dy`: y-axis resolution, [V] or [A] (default = 1e-3).

`id`: An identification string.

`props`: A structure with any optional properties passed to cip_traceset.

Returns a structure object with the following fields:

params_tests_dataset, cip_mags

See also: `physiol_cip_traceset` (p. 170), `params_tests_dataset` (p. 132), `params_tests_db` (p. 138)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/01/28

A.7.2 Method `cip_traceset_dataset/display`

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.7.3 Method `cip_traceset_dataset/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.7.4 Method `cip_traceset_dataset/loadItemProfile`

Summary: Loads a cip_trace_profile object from a raw data file in the fileset.

Usage:

```
a_profile = loadItemProfile(fileset, neuron_id, trace_index)
```

Parameters:

`fileset`**:** A physiol_cip_traceset object.

`neuron_id` **:** tells which item in fileset (corresponds to cells_filename) to use grab the cell information

`trace_index`**:** Index of file in traceset.

Returns:

a_profile: A profile object that implements the getResults method.

See also: `itemResultsRow` (p. **??**), `params_tests_fileset` (p. 157), `paramNames` (p. **??**), `testNames` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14 and Tom Sangrey

---

A.7.5 Method `cip_traceset_dataset/readDBItems`

Summary: Reads all items to generate a params_tests_db object.

Usage:

```
[params, param_names, tests, test_names] = readDBItems(obj)
```

Description: This is a specific method to convert from cip_traceset_dataset to a params_tests_db, or a subclass. Output of this function can be directly fed to the constructor of a params_tests_db or a subclass.

Parameters:

`obj`: A physiol_cip_traceset_fileset

Returns:

params, param_names, tests, test_names: See params_tests_db.

See also: `params_tests_db` (p. 138), `params_tests_fileset` (p. 157), `itemResultsRow` `testNames` (p. **??**), `paramNames` (p. **??**), `physiol_cip_traceset_fileset` (p. 175)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/01/28

A.8 Class `cluster_db`

A.8.1 Constructor `cluster_db/cluster_db`

Summary: A database of cluster centroids generated by a clustering algorithm from a rows of orig_db.

Usage:

```
a_cluster_db = cluster_db(data, col_names, orig_db, cluster_idx, id, props)
```

Description: This is a subclass of tests_db. Use one of the clustering methods in tests_db, such as kmeansCluster, to get an instance of this class.

Parameters:

`data`: Database contents.

`col_names`: The column names.

`orig_db`: DB whose rows are clustered.

`cluster_idx`: Array of cluster numbers that correspond to each row in orig_db.

`id`: An identifying string.

`props`: A structure with any optional properties.

- `sumDistances`: Total distance of elements within each cluster.
- `distanceMeasure`: Measure used to find clusters (Default='correlation')

Returns a structure object with the following fields:

tests_db, orig_db: original DB from which clusters were obtained, cluster_idx: Array associating rows of orig_db to each cluster here. props.

See also: `tests_db` (p. 256), `tests_db/kmeansCluster` (p. 278)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/04/08

A.8.2 Method `cluster_db/display`

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.8.3 Method `cluster_db/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.8.4 Method `cluster_db/plotHist`

Summary: Creates a histogram plot showing the clustering memberships.

Usage:

```
a_plot = plotHist(a_cluster_db, title_str)
```

Parameters:

`a_cluster_db`**:** A cluster_db object.

`title_str`**:** (Optional) String to append to plot title.

Returns:

a_plot: A plot_abstract object that can be plotted.

See also: `plot_abstract` (p. 180), `plotFigure` (p. **??**), `histogram_db` (p. 101), `histogram_db/plot_abstract` (p. 105)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/04/08

A.8.5 Method `cluster_db/plotQuality`

Summary: Creates a plot_abstract of the silhouette plot showing the clustering quality.

Usage:

```
a_plot = plotQuality(a_cluster_db, title_str)
```

Parameters:

`a_cluster_db`**:** A cluster_db object.

`title_str`**:** (Optional) String to append to plot title.

Returns:

a_plot: A plot_abstract object that can be plotted.

See also: plot_abstract (p. 180), plotFigure (p. **??**), silhouette (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/04/08

A.8.6 Method cluster_db/plot_abstract

Summary: Creates a vertical plot_stack of silhouette and membership histograms for the clusters.

Usage:

```
a_plot = plot_abstract(a_cluster_db, title_str)
```

Parameters:

a_cluster_db: A cluster_db object.

title_str: (Optional) String to append to plot title.

Returns:

a_plot: A plot_abstract object that can be plotted.

See also: cluster_db/plotQuality (p. 82), cluster_db/plotHist (p. 82)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/04/08

A.9 Class corrcoefs_db

A.9.1 Constructor corrcoefs_db/corrcoefs_db

Summary: A database of correlation coefficients generated from a column of another database.

Usage:

```
a_coef_db = corrcoefs_db(col_name, coefs, coef_names, pages, id, props)
```

Description: This is a subclass of tests_3d_db. Allows generating a plot, etc.

Parameters:

col_name: The column with which the others are correlated.

coefs: Matrix where each column has another coefficient.

coef_names: Cell array of column names corresponding to coefficients.

pages: Column vector of page indices pointing to the tests_3d_db.

id: An identifying string.

props: A structure with any optional properties.

Returns a structure object with the following fields:

tests_db.

See also: `tests_db` (p. 256), `plot_simple` (p. 193), `tests_db/histogram` (p. 274)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/06

A.10 Class `current_clamp`

A.10.1 Constructor `current_clamp/current_clamp`

Summary: Current clamp object with current and voltage traces.

Description: Subclasses the voltage_clamp object that uses the generic trace object to store voltage clamp I, V data. Inherits the common methods defined in voltage_clamp and trace.

Parameters:

`a_vc`**:** An existing voltage_clamp object that actually contains current-clamp data.

Returns a structure object with the following fields:

voltage_clamp: voltage_clamp object.

See also: `trace` (p. 317), `period` (p. 162)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/02/05

A.10.2 Method `current_clamp/cip_trace`

Summary: Return a cip_trace object of the desired current step.

Usage:

```
a_ct = cip_trace(a_cc, cip_num, props)
```

Parameters:

`a_cc`**:** A cip_trace object.

`cip_num`**:** Index of CIP level.

`props`**:** A structure with any optional properties.

`stepNum`**:** Current step to get results for (default=2).

Returns:

a_ct: A cip_trace object with voltage. cip_level_pA: applied current magnitude [pA].

See also: cip_trace (p. 56), trace (p. 317), spike_shape (p. 215)

Author: Cengiz Gunay <cgunay@emory.edu>, 2011/02/22

A.10.3 Method current_clamp/get

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.10.4 Method current_clamp/getResults

Summary: Extract measurement results from all current steps.

Usage:

```
[results profs] = getResults(a_cc)
```

Parameters:

a_cc: A cip_trace object.

props: A structure with any optional properties.

stepNum: Current step to get results for (default=2).

Returns:

results: A structure associating test names with result values. profs: Cell array of results_profile objects for each current step.

See also: cip_trace (p. 56), trace (p. 317), spike_shape (p. 215)

Author: Cengiz Gunay <cgunay@emory.edu>, 2011/02/22

A.10.5 Method current_clamp/params_tests_db

Summary: Create a database of measurement results changing with applied current.

Usage:

```
a_db = params_tests_db(a_cc, props)
```

Description: Selects cip_level_pA as the only database parameter.

Parameters:

a_cc: A cip_trace object.

props: A structure with any optional properties.

`stepNum`: Current period to get results for. Choose 1 for the initial period, 2 for the pulse period (default) and so on).

`paramsStruct`: Contains parameter names and values that are constant for these traces.

`paramsVary`: Contains parameter names and their varying values for each of these traces in a structure array (e.g., struct('Na', 10, 50 ,100))

Returns:

a_db: A params_tests_db with results collected from getResults profs: (Optional) Cell array of cip_trace_allspikes_profile objects for all current steps.

See also: `getResults` (p. **??**), `cip_trace` (p. 56), `trace` (p. 317), `spike_shape` (p. 215)

Author: Cengiz Gunay <cgunay@emory.edu>, 2011/02/23

A.10.6 Method `current_clamp/set`

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.11 Class `dataset_db_bundle`

A.11.1 Constructor `dataset_db_bundle/dataset_db_bundle`

Summary: The dataset and the DB created from it bundled together.

Usage:

```
a_bundle = dataset_db_bundle(a_dataset, a_db, a_joined_db, props)
```

Description: This class is made to enable operations that require seamless connection between the high-level (joined) DB and the raw data. The raw DB is only required to make a connection to the dataset. Therefore it only needs to contain columns necessary to make this connection (e.g., ItemIndex) and other columns can be discarded to save space. The raw DB corresponds row-to-row to the dataset. The joined DB is a higher-level database where multiple rows from the raw DB is combined into single rows that represent entities (trials, neurons, etc). This is achieved with a function like mergePages or mergeMultipleCIPsInOne. There may be several steps for this process, which can be specified as a function handle in the joinDBfunc property.

Parameters:

`a_dataset`: A params_tests_dataset object or a subclass.

`a_db`: The raw tests_db object (or a subclass) created from the dataset.

`a_joined_db`: The processed DB created from the raw DB.

`props`: A structure with any optional properties.

`joinDBfunc`: A function(a_db) to be called that can generate a_joined_db from it.

Returns a structure object with the following fields:

dataset, db, joined_db, props.

See also: `tests_db` (p. 256), `params_tests_dataset` (p. 132)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/12/13

A.11.2 Method `dataset_db_bundle/constrainedMeasuresPreset`

Summary: Returns a dataset_db_bundle with constrained measures according to chosen preset.

Usage:

```
[a_bundle test_names] = constrainedMeasuresPreset(a_bundle, preset, props)
```

Parameters:

`a_bundle`: A dataset_db_bundle object.

`preset`: Choose preset measure list (default=1).

`props`: A structure with any optional properties.

Returns:

a_bundle: Modified bundle.

See also: `physiol_bundle/constrainedMeasuresPreset` (p. 166)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/06/13

A.11.3 Method `dataset_db_bundle/ctFromRows`

Summary: Loads a cip_trace object from a raw data file in the a_bundle.

Usage:

```
a_cip_trace = ctFromRows(a_bundle, a_db, cip_levels, props)
```

Description: This method is not implemented for the generic dataset_db_bundle class. See subclass implementations.

Parameters:

`a_bundle`: A dataset_db_bundle object.

`a_db`: A DB created by the dataset in the a_bundle to read the neuron index numbers from.

`cip_levels`: A column vector of CIP-levels to be loaded.

`props`: A structure with any optional properties.
(passed to a_bundle.dataset/cip_trace)

Returns:

a_cip_trace: One or more cip_trace objects that hold the raw data.

See also: `model_ct_bundle/ctFromRows` (p. 109), `physiol_bundle/ctFromRows` (p. 166)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/10/11

A.11.4 Method `dataset_db_bundle/display`

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.11.5 Method `dataset_db_bundle/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.11.6 Method `dataset_db_bundle/getNeuronRowIndex`

Summary: Returns the neuron index from bundle.

Usage:

```
a_row_index = getNeuronRowIndex(a_bundle, an_index, props)
```

Description: This is a polymorphic method. Therefor it is not defined for this class, but see subclasses of dataset_db_bundle for its more meaningful implementations.

Parameters:

`a_bundle`: A dataset_db_bundle subclass object.

`an_index`: An index number of neuron, or a DB row containing this.

`props`: A structure with any optional properties.

Returns:

a_row_index: A row index of neuron in a_bundle.joined_db.

Example:

```
» displayRows(mbundle.joined_db(getNeuronRowIndex(mbundle, 98364), :))
```

See also: dataset_db_bundle (p. 86)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/06/09

A.11.7 Method dataset_db_bundle/matchingRow

Summary: Creates a criterion database for matching the tests of a row.

Usage:

```
crit_db = matchingRow(a_bundle, row, props)
```

Description: Copies selected test values from row as the first row into the criterion db. Adds a second row for the STD of each column in the db.

Parameters:

`a_bundle`: A tests_db object.

`row`: A row index to match.

`props`: A structure with any optional properties.

`distDB`: Take the standard deviation from this db instead.

Returns:

crit_db: A tests_db with two rows for values and STDs.

Example:

```
physiol_bundle has an overloaded matchingRow method that
takes the TracesetIndex as argument:
» crit_db = matchingRow(pbundle, 61)
```

See also: rankMatching (p. **??**), tests_db (p. 256), tests2cols (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/12/21

A.11.8 Method `dataset_db_bundle/plotfICurve`

Summary: Generates a f-I curve doc_plot for neuron at given an_index in a_bundle.

Usage:

```
a_plot = plotfICurve(a_bundle, trial_num, props)
```

Parameters:

`a_bundle`: A dataset_db_bundle object.

`an_index`: An index with which to address the a_bundle.

`props`: A structure with any optional properties.

`shortCaption`: This appears in the figure caption.

`plotMStats`: If set, add the a_bundle stats plot.

`captionToStats`: Use this as its legend label.

`quiet`: if given, no title is produced
(passed to plot_superpose)

Returns:

a_plot: A plot_superpose that contains a f-I curve plot.

Example:

```
» a_p = plotfICurve(r, 1);
» plotFigure(a_p, 'The f-I curve of best matching model');
```

See also: `plot_abstract` (p. 180), `plot_superpose` (p. 196)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/01/16

A.11.9 Method `dataset_db_bundle/profileFromRows`

Summary: Loads a cip_trace object from a raw data file in the a_bundle.

Usage:

```
a_prof = profileFromRows(a_bundle, indices, props)
```

Parameters:

`a_bundle`: A dataset_db_bundle object.

`indices`: Array of trial/ItemIndex numbers, or a db that contains rows of them.

`props`: A structure with any optional properties.

`indexName`: Column to take the ItemIndex from.
(passed to params_tests_dataset/loadItemProfile)

Returns:

a_prof: One or more results_profile objects.

See also: `params_tests_dataset/loadItemProfile` (p. 135)

Author: Cengiz Gunay <cgunay@emory.edu>, 2014/06/19

A.11.10 Method `dataset_db_bundle/rankingReportTeX`

Summary: Generates a report by comparing a_bundle with the given match criteria, crit_db from crit_bundle.

Usage:

```
tex_string = rankingReportTeX(a_bundle, crit_bundle, crit_db, props)
```

Description: Generates a LaTeX document with: - (optional) Raw traces compared with some best matches at different distances - Values of some top matching a_db rows and match errors in a floating table. - colored-plot of measure errors for some top matches. - Parameter distributions of 50 best matches as a bar graph.

Parameters:

`a_bundle`: A dataset_db_bundle object that contains the DB to compare rows from.

`crit_bundle`: A dataset_db_bundle object that contains the criterion dataset.

`crit_db`: A tests_db object holding the match criterion tests and STDs which can be created with matchingRow.

`props`: A structure with any optional properties.

- `caption`: Identification of the criterion db (not needed/used?).
- `num_matches`: Number of best matches to display (default=10).
- `rotate`: Rotation angle for best matches table (default=90).

Returns:

tex_string: LaTeX document string.

See also: `displayRowsTeX` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/12/13

A.11.11 Method `dataset_db_bundle/reportNeuron`

Summary: Generates a report of neuron at given an_index of a_bundle.

Usage:

```
a_doc = reportNeuron(a_bundle, an_index, props)
```

Description: Generates a report document with preset layouts of annotated plots of the selected neuron. See reportLayout below for presets.

Parameters:

`a_bundle`: a dataset_db_bundle object which contains the neuron

`an_index`: The index to pass to ctFromRows method of a_bundle.

`props`: A structure with any optional properties.

`reportLayout`: Allows choosing one of predefined report types (strings):

`1`: Only +/- 100 pA traces in one plot (default).
1a/b: Either one of the +/- 100 pA traces in one plot.

`2`: Only +/- 100 pA traces and spike shapes in one horiz. plot.

`2a`: +/- 100 pA traces, f-I curve and spike shapes in one horiz. plot.

`3`: +100 pA raw trace and rate profile stacked vertically.

`3b`: -100 pA raw trace and rate profile stacked vertically.

`4`: Horiz stack of +/- 100 pA raw trace with rate profiles underneath.

`5`: 5-piece trace, spike shape, f-I curve, f-t curve quad-plot.

`numTraces`: Limit number of traces to show in plot (>=1).

`traces`: List of acceptable traces to load.

`traceAxisLimits`: If given, use these limits for trace plots.

`rateAxisLimits`: If given, use these limits for rate plots.

`fIAxisLimits`: If given, use these limits for fIcurve plots.

`fIstats`: Add a fI-stats plot in addition to the curve.

`sshapeAxisLimits`: If given, use these limits for spike shape plots.

`sshapeResults`: If 1, plot measures on the spike shape (default=1).

Returns:

a_doc: A doc_generate object, or a subclass, that can be plotted, or printed as a PS or PDF file.

Example:

```
» printTeXFile(reportNeuron(mbundle, 2222), 'a.tex')
or:
» plotFigure(get(reportNeuron(mbundle, 2222), 'plot'))
or if the result is one or many doc_plot objects:
» plot(reportNeuron(mbundle, 2222:2224, struct('reportLayout', '5')))
```

See also: doc_multi (p. 97), doc_generate (p. 94), doc_generate/printTeXFile (p. 96)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/01/24

A.11.12 Method dataset_db_bundle/set

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.11.13 Method dataset_db_bundle/simNewParams

Summary: Simulates new parameter set and adds it to dataset and DB.

Usage:

```
[a_bundle, a_new_db, a_new_joined_db] = simNewParams(a_bundle, a_param_row_db, props)
```

Parameters:

a_bundle: A dataset_db_bundle object.

a_db: Rows with new parameters.

props: A structure with any optional properties.

simFunc: Handle to function to call for simulating the parameter row db (e.g., @(row_db)).

trial: Trial number for new parameter set. It also indicates parallel mode, which does not alter existing files, but instead creates new files with this trial number suffix.

writePar: If 1, create a new par file (default=0).

Returns:

a_bundle: The bundle with updated parameters and measures. a_new_db: Newly created rows. a_new_joined_db: Newly created rows joined.

See also: params_tests_dataset/loadItemProfile (p. 135)

Author: Cengiz Gunay <cgunay@emory.edu>, 2014/06/23

A.11.14 Method dataset_db_bundle/subsasgn

Summary: Defines generic index-based assignment for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/02/06

A.11.15 Method `dataset_db_bundle/subsref`

Summary: Defines indexing for tests_db objects for () and . operations.

Usage:

```
obj = obj(rows, tests) obj = obj.attribute
```

Description: Returns attributes or selects the given test columns and rows and returns in a new tests_db object.

Parameters:

`obj`: A tests_db object.

`rows`: A logical or index vector of rows. If ':', all rows.

`tests`: Cell array of test names or column indices. If ':', all tests.

`attribute`: A tests_db class attribute.

Returns:

obj: The new tests_db object.

See also: `subsref` (p. **??**), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/17

A.12 Class `doc_generate`

A.12.1 Constructor `doc_generate/doc_generate`

Summary: Generic class to help generate printed or annotated documents with results.

Usage:

```
a_doc = doc_generate(text_string, id, props)
```

Description: This constitutes the base class for other doc_ classes. For convenience, this class holds a text_string to be printed when the document is generated with the printTeXFile option.

Parameters:

`text_string`: Contents of this document.

`id`: An identifying string.

`props`: A structure with any optional properties.

Returns a structure object with the following fields:

text, id, props.

See also: `doc_plot` (p. 98), `doc_multi` (p. 97)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/01/17

A.12.2 Method doc_generate/display

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.12.3 Method doc_generate/get

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.12.4 Method doc_generate/getTeXString

Summary: Returns the TeX representation for the document.

Usage:

```
tex_string = getTeXString(a_doc, props)
```

Description: This is an abstract placeholder for this method. It specifies what this method should do in the subclasses that implement it. This method should create all the auxiliary files needed by the document. The generated tex_string should be ready to be visualized.

Parameters:

`a_doc`: A tests_db object.

`props`: A structure with any optional properties.

Returns:

tex_string: A string that contains TeX commands, which upon writing to a file, can be interpreted by the TeX engine to produce a document.

Example:

```
doc_plot has an overloaded getTeXString method:
» tex_string = getTeXString(a_doc_plot)
» string2File(tex_string, 'my_doc.tex')
then my_doc.tex can be used by including from a valid LaTeX document.
```

See also: doc_generate (p. 94), doc_plot (p. 98)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/01/17

A.12.5 Method doc_generate/printTeXFile

Summary: Creates a TeX file with the contents of this document.

Usage:

```
printTeXFile(a_doc, filename, props)
```

Description: Calls getTeXString to generate the contents. The filename is adjusted with a call to properFilename to generate an acceptable TeX filename. TeX-specific should only be added at this point or at getTeXString, because before we want the object to be a generic document container.

Parameters:

`a_doc`**:** A tests_db object.

`filename`**:** To write the TeX string.

`props`**:** A structure with any optional properties.

`docDir`**:** Directory in which to create TeX file.
(passed to getTeXString)

Returns:

tex_string: A string that contains TeX commands, which upon writing to a file, can be interpreted by the TeX engine to produce a document.

Example:

```
» a_doc = doc_plot(a_plot, 'Results from cell.', 'Results.', struct, '');
» printTeXFile(a_doc, 'my_doc.tex')
then my_doc.tex can be used by including from a valid LaTeX document.
```

See also: doc_generate (p. 94), doc_plot (p. 98), string2File (p. **??**), properFilename (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/01/17

A.12.6 Method doc_generate/set

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.12.7 Method doc_generate/subsref

Summary: Defines generic indexing for objects.

A.13 Class `doc_multi`

A.13.1 Constructor `doc_multi/doc_multi`

Summary: A document that is composed of multiple other doc_generate objects.

Usage:

```
a_doc = doc_multi(docs, id, props)
```

Parameters:

`docs`: A vector of doc_generate objects.

`id`: An identifying string.

`props`: A structure with any optional properties.

Returns a structure object with the following fields:

docs, doc_generate.

Example:

```
» mydoc = doc_multi([doc_plot(a_plot1), doc_plot(a_plot2)], 'Two plots')
» printTeXFile(mydoc, 'two_plots.tex')
```

See also: `doc_generate` (p. 94), `getTeXString` (p. **??**), `doc_generate/printTeXFile` (p. 96)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/01/17

A.13.2 Method `doc_multi/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.13.3 Method `doc_multi/getTeXString`

Summary: Returns the TeX representation for the document.

Usage:

```
tex_string = getTeXString(a_doc, props)
```

Description: Concatenates TeX representations of doc_generate, or subclass, objects it contains.

Parameters:

`a_doc`: A tests_db object.

`props`: A structure with any optional properties.

Returns:

tex_string: A string that contains TeX commands, which upon writing to a file, can be interpreted by the TeX engine to produce a document.

Example:

```
doc_plot has an overloaded getTeXString method:
» tex_string = getTeXString(a_doc_plot)
» string2File(tex_string, 'my_doc.tex')
then my_doc.tex can be used by including from a valid LaTeX document.
```

See also: `doc_generate` (p. 94), `doc_plot` (p. 98)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/01/17

A.13.4 Method `doc_multi/set`

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.14 Class `doc_plot`

A.14.1 Constructor `doc_plot/doc_plot`

Summary: Generates a formatted plot for printing, annotated with captions.

Usage:

```
a_doc = doc_plot(a_plot, caption, plot_filename, float_props, id, props)
```

Description: The generated file may take an extension according to chosen format.

Parameters:

`a_plot`: A plot_abstract ready to be visualized.

`caption`: Long caption to appear under the figure.

`plot_filename`: Filename to be generated from the plot.

`float_props`: Formatting instructions passed to TeXfloat.

`id`: An identifying string.

`props`: A structure with any optional properties.

`orient`: Passed to the orient command before printing to figure file.
(others passed to doc_plot/getTeXString and TeXfloat)

Returns a structure object with the following fields:

plot, caption, plot_filename, float_props, doc_generate.

Example:

```
» a_doc = doc_plot(plotData(my_cip_trace), 'My CIP trace.  Very interesting.', ...
'trace1', struct, 'first doc');
» printTeXFile(a_doc, 'my_doc.tex');
```

See also: doc_generate (p. 94), TeXfloat (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/01/17

A.14.2 Method doc_plot/get

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.14.3 Method doc_plot/getTeXString

Summary: Returns the TeX representation for the plot document.

Usage:

```
tex_string = getTeXString(a_doc, props)
```

Description: Plots, prints EPS files and generates the necessary LaTeX code.

Parameters:

`a_doc`**:** A doc_plot object.

`props`**:** A structure with any optional properties.

`docDir`**:** Base directory for files.

`plotRelDir`**:** Subdirectory for plot files. \input commands will include this directory. (passed to TeXfloat)

Returns:

tex_string: A string that contains TeX commands, which upon writing to a file, can be interpreted by the TeX engine to produce a document.

Example:

```
doc_plot has an overloaded getTeXString method:
» tex_string = getTeXString(a_doc_plot)
» string2File(tex_string, 'my_doc.tex')
then my_doc.tex can be used by including from a valid LaTeX document.
```

See also: doc_generate (p. 94), doc_plot (p. 98)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/01/17

A.14.4 Method doc_plot/plot

Summary: Default plot method to preview the contained plot in a new figure.

Usage:

```
figure_handle = plot(a_doc, props)
```

Description: Only generate the contained plot for previewing. Opens a new figure.

Parameters:

a_doc: A doc_plot object.

props: A structure with any optional properties.

Returns:

figure_handle: Handle of newly opened figure.

Example:

```
» figure_handle = plot(a_doc_plot)
```

See also: plot_abstract/plotFigure (p. 185), doc_generate (p. 94), doc_plot (p. 98)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/01/17

A.14.5 Method doc_plot/plot_abstract

Summary: Returns the plot_abstract object within the doc_plot.

Usage:

```
a_plot = plot_abstract(a_doc, title_str, props)
```

Description: If a_doc is a vector, returns a vector of plot_abstract objects.

Parameters:

a_doc: A doc_plot object.

title_str: (Optional) String to replace plot title.

props: A structure with any optional properties.
(rest passed to plot_abstract.)

Returns:

a_plot: A plot_abstract object or vector that can be visualized.

See also: `doc_plot/plot` (p. 100), `plot_abstract` (p. 180)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/17

A.14.6 Method `doc_plot/set`

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.15 Class `histogram_db`

A.15.1 Constructor `histogram_db/histogram_db`

Summary: A database of histogram values generated for a column of another database.

Usage:

```
a_hist_db = histogram_db(col_name, bins, hist_results, id, props)
```

Description: This is a subclass of tests_db. Allows generating a histogram plot, etc. The histogram count is entered as a column named histVal.

Parameters:

`col_name`: The column name of the histogrammed value.

`bins`: The values for which the histogram values are calculated.

`hist_results`: A column vector of histogram values.

`id`: An identifying string.

`props`: A structure with any optional properties.

Returns a structure object with the following fields:

tests_db, props.

Example:

```
» [hist_results, bins] = hist(my_data);
» a_hist_db = histogram_db('firing_rate', bins, hist_results, 'rate histogram db');
» plot(a_hist_db);
```

See also: `tests_db` (p. 256), `plot_simple` (p. 193), `tests_db/histogram` (p. 274)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/20

A.15.2 Method `histogram_db/calcKLhist`

Summary: Calculates the Kullback-Leibler divergence between two histograms with same bins.

Usage:

```
kl_bits = calcKLhist(two_hist_dbs, props)
```

Description: Histograms must have same bins! See example how to use tests_db/histogram to get same bins from multiple DBs.

Parameters:

`two_hist_dbs`: Array of two histogram_db objects to calculate KL divergence.

`props`: Structure with optional parameters.

`sym`: calculate symmetric KL divergence. Can be 'sum' for sum of divergences, 'avg' for average, and 'res' for resistor average.

Returns:

kl_bits: The calculated non-negative divergence in bits.

Example:

```
» kl_bits = calcKLhist(histogram([one_db, another_db], 'var1', 100))
```

See also: `histogram_db` (p. 101), `calcKLmodel` (p. **??**), `tests_db/histogram` (p. 274)

Author: Cengiz Gunay <cgunay@emory.edu>, 2009/04/03

A.15.3 Method `histogram_db/calcKLmodel`

Summary: Calculates the Kullback-Leibler divergence of the histogram to a model distribution.

Usage:

```
[mode_val, mode_mag] = calcKLmodel(a_hist_db)
```

Parameters:

`a_hist_db`: A histogram_db object.

`dist_model`: Structure that contains the distribution parameters. Must

`be one of these`:

`Normal distribution`: struct('dist', 'norm', 'mu', mean, 'sigma', var)

`Poisson distribution`: struct('dist', 'pois', 'lambda', l)

`Exponential distribution`: struct('dist', 'exp', 'mu', m)

`Uniform distribution`: struct('dist', 'uni')

`props`: Structure with optional parameters.

`plot`: If 1, return a plot_abstract object with both distributions.

Returns:

kl_bits: The calculated divergence in bits.

See also: `histogram_db` (p. 101)

Author: Cengiz Gunay <cgunay@emory.edu>, 2009/03/24

A.15.4 Method `histogram_db/calcMode`

Summary: Finds the mode of the distribution, that is, the bin with the highest value.

Usage:

```
[mode_val, mode_mag] = calcMode(a_hist_db)
```

Parameters:

`a_hist_db`: A histogram_db object.

Returns:

mode_val: The center of the bin that has most members. mode_mag: The value of the histogram bin.

See also: `histogram_db` (p. 101)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/04/27

A.15.5 Method `histogram_db/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.15.6 Method `histogram_db/plotEqSpaced`

Summary: Generates a histogram plot where the values are equally spaced on the x-axis. For use with non-linear parameter values.

Usage:

```
a_plot = plotEqSpaced(a_hist_db, title_str, props)
```

Description: Generates a plot_simple object from this histogram.

Parameters:

`a_hist_db`: A histogram_db object.

`title_str`: Optional title string.

`props`: Optional properties passed to plot_abstract.

`quiet`: If 1, don't include database name on title.

`skipXnum`: Skip every this many values to fit labels on X-axis (default=1).

`totalXnum`: Skip to fit this number of total X-axis tick labels.

Returns:

a_plot: A object of plot_abstract or one of its subclasses.

See also: `plot_abstract` (p. 180), `plot_simple` (p. 193)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/22

A.15.7 Method `histogram_db/plotPages`

Summary: Generates a plot containing subplots from each page of histograms.

Usage:

```
a_plot = plotPages(a_hist_db, title_str, props)
```

Description: For each page of the histogram, a histogram is placed in a subplot.

Parameters:

`a_hist_db`: A histogram_db object.

`title_str`: (Optional) String to append to plot title.

`props`: A structure with any optional properties, passed to plot_stack.

`an_orient`: Stack orientation. One of 'x', or 'y' (Default='y'; vertical).

`quiet`: If 1, only display given title_str.

Returns:

a_plot: A object of plot_abstract or one of its subclasses.

See also: `plotPages` (p. **??**), `plot_simple` (p. 193)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/04

A.15.8 Method `histogram_db/plotRowMatrix`

Summary: Generates a subplot matrix of measure columns versus rows of databases.

Usage:

```
a_plot = plotRowMatrix(hist_dbs, title_str, props)
```

Description: Each row in the hist_dbs is assumed to come from a different DB. Columns represent histograms of different measurements. The plot is made such that histograms in each row have the same maximal count, and histograms in each column have the same axis limits.

Parameters:

`hist_dbs`: A matrix of histogram_db objects.

`title_str`: Title to go at the top.

`props`: A structure with any optional properties.

`rowLabels`: Cell array of y-axis labels for each row.
(rest passed to histogram_db/plot_abstract)

Returns:

a_plot: A object of plot_abstract or one of its subclasses.

See also: `plot_abstract` (p. 180)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/11/22

A.15.9 Method `histogram_db/plot_abstract`

Summary: Creates a bar plot from the histogram.

Usage:

```
a_plot = plot_abstract(a_hist_db, title_str, props)
```

Description: Generates a plot_simple object from this histogram.

Parameters:

`a_hist_db`: A histogram_db object.

`title_str`: Optional title string.

`props`: Optional properties passed to plot_abstract.

`command`: Plot command (Optional, default='bar')

`endZeros`: Prefix and suffix bins with zero values to make a smooth plot.

`lineSpec`: Line specification passed to bar command.

`logScale`: If 1, use logarithmic y-scale.
`shading`: 'faceted' (default) or 'flat'.
`barWidth`: Controls spacing between bars (see width argument for the bar command; default=0.8).
`quiet`: If 1, don't include database name on title.

Returns:

a_plot: A object of plot_abstract or one of its subclasses.

See also: `plot_abstract` (p. 180), `plot_simple` (p. 193)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/22

A.15.10 Method `histogram_db/subsref`

Summary: Defines generic indexing for objects.

A.16 Class `mesh_amira`

A.16.1 Constructor `mesh_amira/mesh_amira`

Summary: Contains 3D mesh data imported from the Amira software.

Usage:

```
a_mesh = mesh_amira(filename, id, props)
```

Description: By default it reads an ASCII formatted v2 type Amira file and expects only vertex and edge information for representing 3D reconstructions of neurons.

Parameters:

`filename`: Amira mesh file (*.am).
`id`: Identification string.
`props`: A structure with any optional properties.
`isVerbose`: If 1, produce verbose info while loading Amira file (for debugging).

Returns a structure object with the following fields:

nVertices, nEdges, nOrigins: Number of vertices, edges and origins, resp., vertices: 3D coordinates of each vertex, neighborCount: Number of neighbors for each vertex, raddi: Radius of each vertex, neighborList: Index of vertices that are neighboring to each vertex, Origins: Indices of vertices marked as 'origin', origFilename, id, props

See also: `private/loadAmiraMesh` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2012/02/03

A.16.2 Method `mesh_amira/exportMorphML`

Summary: Export mesh to MorphML XML format.

Usage:

```
a_dom = exportMorphML(a_mesh, props)
```

Parameters:

`a_mesh`**:** A mesh_amira object.

`props`**:** A structure with any optional properties.

Returns:

a_dom: A Matlab DomNode object.

Example:

```
» a_mesh = mesh_amira('my_amira.am', 'Neuron 1')
» xmlwrite('neuron.xml', exportMorphML(a_mesh))
```

See also:

Author: Cengiz Gunay <cgunay@emory.edu>, 2012/02/03

A.16.3 Method `mesh_amira/plot_abstract`

Summary: Prepare a plot the of the 3D mesh.

Usage:

```
a_p = plot_abstract(a_mesh, title_str, props)
```

Description: Can be stacked or superposed with other plot objects.

Parameters:

`a_mesh`**:** A voltage clamp object.

`title_str`**:** (Optional) Text to appear in the plot title.

`props`**:** A structure with any optional properties.

`quiet`**:** If 1, only use given title_str.

Returns:

a_p: A plot_abstract object.

Example:

```
» a_mesh = mesh_amira('my_amira.am', 'Neuron 1')
» plotFigure(plot_abstract(a_mesh, ' - side view'))
```

See also: `plotFigure` (p. **??**), `plot_superpose` (p. 196), `plot_stack` (p. 194)

Author: Cengiz Gunay <cgunay@emory.edu>, 2012/02/03

A.17 Class `model_ct_bundle`

A.17.1 Constructor `model_ct_bundle/model_ct_bundle`

Summary: The model cip_trace dataset and the DB created from it bundled together.

Usage:

```
a_bundle = model_ct_bundle(a_dataset, a_db, a_joined_db, props)
```

Description: This is a subclass of dataset_db_bundle, specialized for model datasets.

Parameters:

`a_dataset`: A params_cip_trace_fileset object.

`a_db`: The raw params_tests_db object created from the dataset. It only needs to have the pAcip, trial, and ItemIndex columns.

`a_joined_db`: The one-model-per-line DB created from the raw DB.

`props`: A structure with any optional properties.

Returns a structure object with the following fields:

dataset_db_bundle.

Example:

```
» a_joined_db = mergeMultipleCIPsInOne(a_db, ...)
» a_bundle = model_ct_bundle(a_params_cip_trace_fileset, a_db, a_joined_db)
```

See also: `dataset_db_bundle` (p. 86), `tests_db` (p. 256), `params_tests_dataset` (p. 132), `params_tests_db/mergeMultipleCIPsInOne` (p. 146)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/12/13

A.17.2 Method `model_ct_bundle/addToDB`

Summary: Concatenate to existing DB in the bundle.

Usage:

```
a_mbundle = addToDB(a_mbundle, a_raw_db, props)
```

Description: If joinedDb is not given in props, calls treatSimDB to get the joined_db from this raw DB. Then concats to both db and joined_db in bundle.

Parameters:

`a_mbundle`: A model_ct_bundle object.

`a_crit_bundle`: A physiol_bundle having a crit_db as its joined_db.

`props`: A structure with any optional properties.

`joinedDb`: The joined version of a_raw_db.

`dataset`: If given, this one is used to replace the fileset in the bundle.

Returns:

a_mbundle: a model_ct_bundle object containing the added DB.

Example:

```
» mbundle = addToDB(mbundle, params_tests_db(mfileset, [19684:59956]))
```

See also: `params_tests_fileset/addFiles` (p. 158), `multi_fileset_gpsim_cns2005/addFileDir` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/02/06

A.17.3 Method `model_ct_bundle/collectPhysiolMatches`

Summary: Compare model DB to given physiol criteria and return some top matches.

Usage:

```
row_index = collectPhysiolMatches(a_mbundle, a_crit_bundle, props)
```

Parameters:

`a_mbundle`: A model_ct_bundle object.

`a_crit_bundle`: A physiol_bundle object that holds the criterion neuron.

`props`: A structure with any optional properties.

`showTopmost`: Number of top matching models to return (default=50)

Returns:

row_index: Row indices of best matching models.

See also:

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/01/18

A.17.4 Method `model_ct_bundle/ctFromRows`

Summary: Loads a cip_trace object from a raw data file in the a_mbundle.

Usage:

```
a_cip_trace = ctFromRows(a_mbundle, a_db|trials, cip_levels, props)
```

Description: This is an overloaded method.

Parameters:

`a_mbundle`: A model_ct_bundle object.

`a_db`: A DB created by the dataset in the a_mbundle to read the trial numbers from.

`trials`: A column vector with trial numbers.

`cip_levels`: A column vector of CIP-levels to be loaded.

`props`: A structure with any optional properties.
(passed to a_mbundle.dataset/cip_trace)

Returns:

a_cip_trace: One or more cip_trace objects that hold the raw data.

See also: `params_cip_trace_fileset/ctFromRows` (p. 129)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/07/13

A.17.5 Method `model_ct_bundle/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.17.6 Method `model_ct_bundle/getNeuronLabel`

Summary: Constructs the neuron label from bundle.

Usage:

```
a_label = getNeuronLabel(a_bundle, trial_num, props)
```

Parameters:

`a_bundle`: A physiol_cip_traceset_fileset object.

`trial_num`: The trial number of model neuron.

`props`: A structure with any optional properties.

Returns:

a_label: A string label identifying selected neuron in bundle.

See also: `dataset_db_bundle` (p. 86)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/05/26

A.17.7 Method `model_ct_bundle/getNeuronRowIndex`

Summary: Returns the neuron index from bundle.

Usage:

```
a_row_index = getNeuronRowIndex(a_bundle, trial_num, props)
```

Parameters:

`a_bundle`: A model_ct_bundle object.

`trial_num`: The trial number of model neuron, or a DB row containing this.

`props`: A structure with any optional properties.

Returns:

a_row_index: A row index of neuron in a_bundle.joined_db.

See also: `dataset_db_bundle` (p. 86)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/06/09

A.17.8 Method `model_ct_bundle/getTrialNum`

Summary: Extracts identifying neuron trial number from DB.

Usage:

```
trial_num = getTrialNum(a_bundle, a_db|trial_num, props)
```

Parameters:

`a_bundle`: A physiol_cip_traceset_fileset object.

`a_db`: DB rows representing deisred model neuron(s).

`trial_num`: Trial numbers. If specified, this function does nothing but return them.

`props`: A structure with any optional properties.

Returns:

trial_num: The trial number(s) identifying selected neuron(s) in bundle.

See also: `dataset_db_bundle` (p. 86)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/05/26

A.17.9 Method `model_ct_bundle/plotComparefICurve`

Summary: Generates a f-I curve doc_plot comparing m_trial and to_index.

Usage:

```
a_plot = plotComparefICurve(m_bundle, m_trial, to_bundle, to_index, props)
```

Description: Note that this is not a general method. to_bundle should have been able to accept any type of bundle. Most probably this method is redundant and deprecated.

Parameters:

`m_bundle`: A model_ct_bundle object.

`m_trial`: Trial number of model.

`to_bundle`: A physiol_bundle object.

`to_index`: TracesetIndex of neuron.

`props`: A structure with any optional properties.

`shortCaption`: This appears in the figure caption.

`plotMStats`: If set, add the m_bundle stats plot.

`plotToStats`: If set, add the to_bundle stats plot.

`captionToStats`: Use this as its legend label.

`quiet`: if given, no title is produced
(passed to plot_superpose)

Returns:

a_plot: A plot_superpose that contains a f-I curve plot.

Example:

```
» a_p = plotComparefICurve(r, 1);
» plotFigure(a_p, 'The f-I curve of best matching model');
```

See also: `plot_abstract` (p. 180), `plot_superpose` (p. 196)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/01/16

A.17.10 Method `model_ct_bundle/plotCompareRanks`

Summary: Generates a plots of given ranks from the ranked_bundle.

Usage:

```
plots = plotCompareRanks(m_bundle, p_bundle, a_ranked_db, ranks, props)
```

Parameters:

`m_bundle`: A model_ct_bundle object.

`p_bundle`: A dataset_db_bundle object that originated the criterion.

`a_ranked_db`: A ranked_db generated from ranking m_bundle.

`ranks`: Vector of rank indices for which to generate the plots.

`props`: A structure with any optional properties.

Returns:

plots: A structure that contains the joined_db, and the plot vectors trace_d100_plots and trace_h100_plots.

Example:

```
» plots = plotCompareRanks(r, 1:10);
» plotFigure(plots.trace_d100_plots(1), 'The best matching +100 pA CIP trace');
```

See also:

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/01/16

A.17.11 Method `model_ct_bundle/rankMatching`

Summary: Create a ranked_db from given criterion db.

Usage:

```
a_ranked_db = rankMatching(a_mbundle, a_crit_db, props)
```

Parameters:

`a_mbundle`: A model_ct_bundle object.

`a_crit_db`: A crit_db created by a matchingRow method.

`props`: A structure with any optional properties.
(passed to tests_db/rankMatching)

Returns:

a_ranked_db: a ranked_db object containing the rankings.

See also: `tests_db/rankMatching` (p. 300), `ranked_db` (p. 200)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/01/18

A.17.12 Method `model_ct_bundle/reportCompareModelToPhysiolNeuron`

Summary: Generates a report by comparing given model neuron to given physiol neuron.

Usage:

```
a_doc_multi = reportCompareModelToPhysiolNeuron(m_bundle, trial_num, p_bundle,
traceset_index, props)
```

Description: Generates a report document with: - Figure displaying raw traces of the physiol neuron compared with the model neuron - Figure comparing f-I curves of the two neurons. - Figure comparing spont and pulse spike shapes of the two neurons.

Parameters:

`m_bundle, p_bundle:` dataset_db_bundle objects of the model and physiology neurons.

`trial_num:` Trial number of desired model neuron in m_bundle.

`traceset_index:` TracesetIndex of desired neuron in p_bundle.

`props:` A structure with any optional properties.

`horizRow:` If defined, create a row-figure with all plots.

`numPhysTraces:` Number of physiology traces to show in plot (>=1).

Returns:

a_doc_multi: A doc_multi object that can be printed as a PS or PDF file.

Example:

```
» printTeXFile(reportCompareModelToPhysiolNeuron(mbundle, 2222, pbundle, 34), 'a.tex')
```

See also: `doc_multi` (p. 97), `doc_generate` (p. 94), `doc_generate/printTeXFile` (p. 96)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/01/24

A.17.13 Method `model_ct_bundle/reportRankingToPhysiolNeuronsTeXFile`

Summary: Compare model DB to given physiol criterion and create a report.

Usage:

```
tex_filename = reportRankingToPhysiolNeuronsTeXFile(m_bundle, p_bundle, a_crit_db, props)
```

Description: A LaTeX report is generated following the example in physiol_bundle/matchingRow. The filename contains the neuron name, followed by the traceset index as an identifier of pharmacological applications, as in gpd0421c_s34.

Parameters:

`m_bundle`: A model_ct_bundle object.

`p_bundle`: A physiol_bundle object.

`a_crit_db`: The criterion neuron chosen with a matchingRow method.

`props`: A structure with any optional properties.

`filenameSuffix`: Append this identifier to the TeX filename.
(others passed to rankMatching)

Returns:

tex_filename: Name of LaTeX file generated.

See also: `tests_db/rankMatching` (p. 300), `physiol_cip_traceset/cip_trace` (p. 172), `physiol_bundle/matchingRow` (p. 168)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/01/18

A.17.14 Method `model_ct_bundle/set`

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.18 Class `model_data_vcs`

A.18.1 Constructor `model_data_vcs/model_data_vcs`

Summary: Combines model description that fits a voltage clamp data.

Usage:

```
a_md = model_data_vcs(model_f, data_vc, id, props)
```

Description: For tasks such as plotting comparison of model to data and generating initial fits for model. Can also have a GUI here for fitting.

Parameters:

`model_f`: Model as a param_func object or a NeuroFit file name.
Make sure to specify required param_I_Neurofit props below.

`data_vc`: Data as a voltage_clamp object directly or in a MAT file or from an ABF file.

`id`: Identification string.

`props`: A structure with any optional properties.

Returns a structure object with the following fields:

model_f, data_vc, model_vc: Obtained by simulating model.

See also: voltage_clamp (p. 338), param_func (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/10/11

A.18.2 Method model_data_vcs/convertTauFromSpline

Summary: Converts m and h tau functions from spline to Hodgkin-Huxley form.

Usage:

```
a_md = convertTauFromSpline(a_md, props)
```

Parameters:

a_md: A model_data_vcs object.

props: A structure with any optional properties.

vRange: Voltage values to evaluate spline function (default=-30:60)

Returns:

a_md: (updated)

Example:

```
» a_test_md = convertTauFromSpline(a_md)
```

See also: model_data_vcs (p. 115), voltage_clamp (p. 338), plot_abstract (p. 180), plotFigure (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2011/05/31

A.18.3 Method model_data_vcs/display

Summary: Return string representation of object.

Usage:

```
str = display(a_md)
```

Parameters:

a_md: A model_data_vcs object.

Returns:

str: Readable string

See also: model_data_vcs (p. 115)

Author: Cengiz Gunay <cgunay@emory.edu>, 2011/05/31

A.18.4 Method `model_data_vcs/fit`

Summary: Fit model to data.

Usage:

```
[a_md a_doc] = fit(a_md, title_str, props)
```

Description: Caveat: what you see in the plots may be different that what the algorithm is comparing for the fits if you provide fitRange or fitRangeRel because the plots will simulate the whole range anyway. Therefore the plots will always have a more correct output than the fitting algorithm – especially if you selected a too narrow simlation range that doesn't let the model react to voltage levels.

Parameters:

`a_md`: A model_data_vcs object.

`props`: A structure with any optional properties.

- `fitRange`: Start and end times [ms] of simulated range used for optimization. Note that simulated range must include prior stimulation steps to set state of diff. eqs.
- `fitRangeRel`: Like fitRange, but relative to first voltage step [ms]. Specify any other voltage step as the first element. See getTimeRelStep.
- `outRangeRel`: Only use this range of the simulated data for fitting. Defined same as fitRangeRel. Multiple rows can contain separate ranges are patched together.
- `fitLevels`: Indices of voltage/current levels to use from clamp data. If empty, not fit is done.
- `dispParams`: If non-zero, display params every once this many iterations.
- `dispPlot`: If non-zero, update a plot of the fit at end of this many iterations. A zero means no plot will be produced at the end.
- `saveModelFile`: If given, save the model every dispParams iteration.
- `saveModelAutoNum`: Use this as a base number and use saveModelFile as sprintf formatted string that includes a number string (e.g., 'until a non-existing file name is found.
- `savePlotFile`: If given, save the plot to this file every dispPlot iteration as the model file.
- `plotMd`: model_data_vcs or subclass object to be used for plots. This reuses the data in the md and only updates the model parameters in the plot.
- `quiet`: If 1, do not include cell name on title.

Returns:

a_md: Updated model_data_vcs object with fit. a_doc: A doc_plot object containing the annotated figure.

Example:

```
» a_md = ...
fit(a_md, '', ...
struct('fitRangeRel', [-.2 165], 'fitLevels', 1:5, ...
'dispParams', 5, ...
'optimset', struct('Display', 'iter')));
```

See also: param_I_v (p. **??**), param_func (p. **??**), doc_plot (p. 98), voltage_clamp/getTimeRelStep (p. 340)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/10/12

A.18.5 Method model_data_vcs/get

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.18.6 Method model_data_vcs/plot

Summary: Generic method to plot a tests_db or a subclass. Requires a plot_abstract method to be defined for this object.

Usage:

```
h = plot(a_obj, title_str, props)
```

Parameters:

a_obj: An object.

title_str: (Optional) String to append to plot title.

props: A structure with any optional properties, passed to plot_abstract.

Returns:

h: The figure handle created.

See also: plot_abstract (p. 180), plotFigure (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/06

A.18.7 Method `model_data_vcs/plotDataCompare`

Summary: Superpose model and data raw traces.

Usage:

```
a_p = plotDataCompare(a_md, title_str, props)
```

Parameters:

`a_md`: A model_data_vcs object.

`title_str`: (Optional) Text to appear in the plot title.

`props`: A structure with any optional properties.

`quiet`: If 1, only use given title_str.

`zoom`: Zoom into activation or inactivation parts if 'act' or 'inact', resp.

`skipStep`: Number of voltage steps to skip at the start for zoom (default=0).

`showSub`: also plot subtracted current

`showV`: also plot voltage protocol.

`levels`: Only plot these current and voltage levels

`colorLevels`: Cycle colors every this number.

`axisLimits`: Set current traces to these limits unless 'zoom' prop is specified.

`iLimits`: If specified, override axisLimits y-axis values with these only for the data plot (not the subtraction plot).

Returns:

a_p: A plot_abstract object.

Example:

```
» a_md = model_data_vcs(model, data_vc)
» plotFigure(plotDataCompare(a_md, 'my model'))
```

See also: `model_data_vcs` (p. 115), `voltage_clamp` (p. 338), `plot_abstract` (p. 180), `plotFigure` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/10/12

A.18.8 Method `model_data_vcs/plotDataModelSub`

Summary: Plot model traces subtracted from raw data.

Usage:

```
a_p = plotDataModelSub(a_md, title_str, props)
```

Parameters:

`a_md`: A model_data_vcs object.

`title_str`: (Optional) Text to appear in the plot title.

`props`: A structure with any optional properties.

`quiet`: If 1, only use given title_str.

`zoom`: Zoom into activation or inactivation parts if 'act' or 'inact', resp.

`skipStep`: Number of voltage steps to skip at the start for zoom (default=0).

`levels`: Only plot these current and voltage levels

`colorLevels`: Cycle colors every this number.

`axisLimits`: Set current traces to these limits unless 'zoom' prop is specified. (Rest passed to voltage_clamp/plot_abstract)

Returns:

a_p: A plot_abstract object.

Example:

```
» a_md = model_data_vcs(model, data_vc)
» plotFigure(plotDataModelSub(a_md, 'my model'))
```

See also: `model_data_vcs` (p. 115), `voltage_clamp` (p. 338), `voltage_clamp/plot_abstract` (p. 344), `plot_abstract` (p. 180), `plotFigure` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/10/21

A.18.9 Method `model_data_vcs/plotModelInfs`

Summary: Plot model m_inf and h_inf curves.

Usage:

```
a_p = plotModelInfs(a_md, title_str, props)
```

Parameters:

`a_md`: A model_data_vcs object.

`title_str`: (Optional) Text to appear in the plot title.

`props`: A structure with any optional properties.

`quiet`: If 1, only use given title_str.

Returns:

a_p: A plot_abstract object.

Example:

```
» a_md = model_data_vcs(model, data_vc)
» plotFigure(plotModelInfs(a_md, 'my model'))
```

See also: `model_data_vcs` (p. 115), `voltage_clamp` (p. 338), `plot_abstract` (p. 180), `plotFigure` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/10/12

A.18.10 Method `model_data_vcs/plotModelTaus`

Summary: Plot I/V curves comparing model and data.

Usage:

```
a_p = plotModelTaus(a_md, title_str, props)
```

Parameters:

`a_md`: A model_data_vcs object.

`title_str`: (Optional) Text to appear in the plot title.

`props`: A structure with any optional properties.

`quiet`: If 1, only use given title_str.

Returns:

a_p: A plot_abstract object.

Example:

```
» a_md = model_data_vcs(model, data_vc)
» plotFigure(plotModelTaus(a_md, 'I/V curves'))
```

See also: `model_data_vcs` (p. 115), `voltage_clamp` (p. 338), `plot_abstract` (p. 180), `plotFigure` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/10/11

A.18.11 Method `model_data_vcs/plotPeaksCompare`

Summary: Plot I/V curves comparing model and data.

Usage:

```
a_p = plotPeaksCompare(a_md, title_str, props)
```

Parameters:

`a_md`: A model_data_vcs object.

`title_str`: (Optional) Text to appear in the plot title.

`props`: A structure with any optional properties.

`quiet`: If 1, only use given title_str.

`skipStep`: Number of voltage steps to skip at the start (default=0).

Returns:

a_p: A plot_abstract object.

Example:

```
» a_md = model_data_vcs(model, data_vc)
» plotFigure(plotPeaksCompare(a_md, 'I/V curves'))
```

See also: `model_data_vcs` (p. 115), `voltage_clamp` (p. 338), `plot_abstract` (p. 180), `plotFigure` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/10/11

A.18.12 Method `model_data_vcs/plot_abstract`

Summary: Superpose model and data raw traces.

Usage:

```
a_p = plot_abstract(a_md, title_str, props)
```

Parameters:

`a_md`: A model_data_vcs object.

`title_str`: (Optional) Text to appear in the plot title.

`props`: A structure with any optional properties.

`quiet`: If 1, only use given title_str.

`zoom`: Zoom into activation or inactivation parts if 'act' or 'inact', resp. Can be a cell to have multiple of these.

`skipStep`: Number of voltage steps to skip at the start for zoom (default=0).

`show`: 'sub' for subtracted current, and 'v' for voltage trace at the bottom row.

`levels`: Only plot these current and voltage levels

`colorLevels`: Cycle colors every this number.

`axisLimits`: Set current traces to these limits unless 'zoom' prop is specified. If it has multiple rows, create multiple data plots for each set of limits.

`vLimits`: If given, limit all voltage plot X axes to these.

`iLimits`: If specified, override axisLimits y-axis values with these only for the data plot (not the subtraction plot).

`dataPlotProps`: Props passed to plotDataCompare.

Returns:

a_p: A plot_abstract object.

Example:

```
» a_md = model_data_vcs(model, data_vc)
» plotFigure(plot_abstract(a_md, 'my model'))
```

See also: `model_data_vcs` (p. 115), `voltage_clamp` (p. 338), `plot_abstract` (p. 180), `plotFigure` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/10/12

A.18.13 Method `model_data_vcs/selectFitParams`

Summary: Constrain or release model parameters for fast current.

Usage:

```
a_md = selectFitParams(a_md, select_what, fit_nofit, props)
```

Parameters:

`a_md`: A model_data_vcs object.

`select_what`: One of 'passive', 'fast', 'fastInact'

`fit_nofit`: 1 for including in fits and 0 for not.

`props`: A structure with any optional properties.

Returns:

a_md: Updated object.

Example:

```
» a_md = selectFitParams(model_data_vcs(model, data_vc))
```

See also: model_data_vcs (p. 115), model_data_vcs/fit (p. 117), voltage_clamp (p. 338), plot_abstract (p. 180), plotFigure (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/10/23

A.18.14 Method model_data_vcs/set

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.18.15 Method model_data_vcs/subsasgn

Summary: Defines generic index-based assignment for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/02/06

A.18.16 Method model_data_vcs/subsref

Summary: Defines indexing for tests_db objects for () and . operations.

Usage:

```
obj = obj(rows, tests) obj = obj.attribute
```

Description: Returns attributes or selects the given test columns and rows and returns in a new tests_db object.

Parameters:

`obj`: A tests_db object.

`rows`: A logical or index vector of rows. If ':', all rows.

`tests`: Cell array of test names or column indices. If ':', all tests.

`attribute`: A tests_db class attribute.

Returns:

obj: The new tests_db object.

See also: subsref (p. **??**), tests_db (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/17

A.18.17 Method `model_data_vcs/updateModel`

Summary: Simulate and save new model into object.

Usage:

```
a_md = updateModel(a_md, model_f, props)
```

Description: Simulates the model to update the model_vc contained.

Parameters:

`a_md`: A model_data_vcs object.

`model_f`: (optional) param_func or subclass object that holds the new model function. If not given, existing model is simulated.

`props`: A structure with any optional properties. (passed to voltage_clamp/simModel)

Returns:

a_md: Updated object.

Example:

```
» a_md = model_data_vcs(model_f, data_vc)
» a_md = updateModel(a_md, new_model_f))
```

See also: `model_data_vcs` (p. 115)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/10/14

A.19 Class `model_ranked_to_physiol_bundle`

A.19.1 Constructor `model_ranked_to_physiol_bundle/model_ranked_to_physiol_bundle`

Summary: A DB bundled with its dataset, ranked to a physiology DB bundle.

Usage:

```
r_bundle = model_ranked_to_physiol_bundle(a_dataset, a_db, a_ranked_db, a_crit_bundle, props)
```

Description: This is a subclass of model_ct_bundle, specialized for model datasets.

Parameters:

`a_dataset`: A params_cip_trace_fileset object.

`a_db`: The raw params_tests_db object created from the dataset. It only needs to have the pAcip, trial, and ItemIndex columns.

`a_ranked_db`: The one-model-per-line DB created from the raw DB.

`a_crit_bundle`: The bundle object associated with crit_db that caused the ranking in a_ranked_db.

`props`: A structure with any optional properties.

Returns a structure object with the following fields:

crit_bundle, model_ct_bundle.

See also: `model_ct_bundle` (p. 108), `ranked_db` (p. 200), `params_tests_dataset` (p. 132)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/12/13

A.19.2 Method `model_ranked_to_physiol_bundle/comparisonReport`

Summary: OBSOLETE - Generates a report by comparing r_bundle with the given match criteria, crit_db from crit_bundle.

Usage:

```
a_doc_multi = comparisonReport(r_bundle, crit_bundle, crit_db, props)
```

Description: Generates a LaTeX document with: - (optional) Raw traces compared with some best matches at different distances - Values of some top matching a_db rows and match errors in a floating table. - colored-plot of measure errors for some top matches. - Parameter distributions of 50 best matches as a bar graph.

Parameters:

`r_bundle`: A dataset_db_bundle object that contains the DB to compare rows from.

`crit_bundle`: A dataset_db_bundle object that contains the criterion dataset.

`crit_db`: A tests_db object holding the match criterion tests and STDs which can be created with matchingRow.

`props`: A structure with any optional properties.

`caption`: Identification of the criterion db (not needed/used?).

`num_matches`: Number of best matches to display (default=10).

`rotate`: Rotation angle for best matches table (default=90).

Returns:

tex_string: LaTeX document string.

See also: `displayRowsTeX` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/01/17

A.19.3 Method `model_ranked_to_physiol_bundle/plotCompareRanks`

Summary: OBSOLETE - Generates a plots of given ranks from the ranked_bundle.

Usage:

```
plots = plotCompareRanks(r_bundle, crit_bundle, crit_db, props)
```

Parameters:

`r_bundle`: A ranked_bundle object.

`ranks`: Vector of rank indices for which to generate the plots.

`props`: A structure with any optional properties.

Returns:

plots: A structure that contains the joined_db, and the plot vectors trace_d100_plots and trace_h100_plots.

Example:

```
» plots = plotCompareRanks(r, 1:10);
» plotFigure(plots.trace_d100_plots(1), 'The best matching +100 pA CIP trace');
```

See also:

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/01/16

A.19.4 Method `model_ranked_to_physiol_bundle/plotfICurve`

Usage:

```
a_doc = docfICurve(r_bundle, crit_bundle, crit_db, props)
```

Parameters:

`r_bundle`: A ranked_bundle object.

`rank_num`: Rank index for which to generate the a_doc.

`props`: A structure with any optional properties.

Returns:

a_doc: A doc_plot that contains a f-I curve plot and associated captions.

Example:

```
» a_d = docfICurve(r, 1);
» plot(a_d, 'The f-I curve of best matching model');
```

See also: `doc_generate` (p. 94), `doc_plot` (p. 98)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/01/16

A.20 Class `params_cip_trace_fileset`

A.20.1 Constructor `params_cip_trace_fileset/params_cip_trace_fileset`

Summary: Description of a raw dataset consisting of cip_trace files varying with parameter values.

Usage:

```
obj = params_cip_trace_fileset(file_pattern, dt, dy, pulse_time_start, pulse_time_width, id,
props)
```

Description: This is a subclass of params_tests_fileset.

Parameters:

`file_pattern`: File pattern mathing all files to be loaded.

`dt`: Time resolution [s]

`dy`: y-axis resolution [ISI (V, A, etc.)]

`pulse_time_start, pulse_time_width`: Start and width of the pulse [dt]

`id`: An identification string

`props`: A structure with any optional properties.

`profile_method_name`: Use this profile method that takes a cip_trace object and returns a results_profile class. It can be 'cip_trace_profile' (default, but outdated) or 'getProfileAllSpikes' (more current). (All other props are passed to cip_trace objects)

Returns a structure object with the following fields:

params_tests_fileset, pulse_time_start, pulse_time_width.

Example:

```
» fileset = params_cip_trace_fileset('/home/abc/data/*.bin', 1e-4, 1e-3, 20001, 10000, 'sim
dataset gpsc0501', struct('trace_time_start', 10001, 'type', 'sim', 'scale_y', 1e3))
```

See also: `params_tests_fileset` (p. 157), `params_tests_db` (p. 138)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.20.2 Method `params_cip_trace_fileset/cip_trace`

Summary: Loads raw cip_traces for each given file_index in this fileset.

Usage:

```
a_cip_trace = cip_trace(fileset, file_index|a_db, props)
```

Parameters:

`fileset`: A params_tests_fileset.

`file_index`: A single or array of indices of files in fileset.

`a_db`: A DB created by this fileset to read the item indices from.

`props`: A structure with any optional properties.

`neuronLabel`: Used for annotation purposes.

Returns:

a_cip_trace: A cip_trace object.

See also: `cip_trace` (p. 56), `params_tests_fileset` (p. 157)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/13

A.20.3 Method `params_cip_trace_fileset/cip_trace_profile`

Summary: Loads a raw cip_trace_profile given a file_index to this fileset.

Usage:

```
a_cip_trace_profile = cip_trace_profile(fileset, file_index)
```

Parameters:

`fileset`: A params_tests_fileset.

`file_index`: Index of file in fileset.

Returns:

a_cip_trace_profile: A cip_trace_profile object.

See also: `cip_trace_profile` (p. 72), `params_tests_fileset` (p. 157)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.20.4 Method `params_cip_trace_fileset/ctFromRows`

Summary: Loads a cip_trace object from raw data files in the fileset.

Usage:

```
a_cip_trace = ctFromRows(m_fileset, m_dball, a_db|itemIndices, cip_levels, props)
```

Parameters:

`m_fileset`: A physiol_cip_traceset_fileset object.

`m_dball`: A DB created by this fileset that contains the trial, pAcip, and ItemIndex cols.

`a_db`: A DB that has one trial for each cip_trace to be loaded.

`itemIndices`: A column vector with ItemIndex numbers.

`cip_levels`: A column vector of CIP-levels to be loaded.

`props`: A structure with any optional properties.

`neuronLabel`: appropriate unique neuron label generated by the bundle. (passed to params_cip_trace_fileset/cip_trace)

Returns:

a_cip_trace: One or more cip_trace objects that hold the raw data.

See also: `loadItemProfile` (p. **??**), `physiol_cip_traceset/cip_trace` (p. 172)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/07/13

A.20.5 Method `params_cip_trace_fileset/display`

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.20.6 Method `params_cip_trace_fileset/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.20.7 Method `params_cip_trace_fileset/loadItemProfile`

Summary: Loads a cip_trace_profile object from a raw data file in the fileset.

Usage:

```
[params_row, tests_row] = loadItemProfile(fileset, file_index)
```

Parameters:

`fileset`: A params_tests_fileset.

`file_index`: Index of file in fileset.

Returns:

a_profile: A profile object that implements the getResults method.

See also: `itemResultsRow` (p. **??**), `params_tests_fileset` (p. 157), `paramNames` (p. **??**), `testNames` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.20.8 Method `params_cip_trace_fileset/set`

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.21 Class `params_results_profile`

A.21.1 Constructor `params_results_profile/params_results_profile`

Summary: Profile with parameters and results together.

Description: This is a subclass of results_profile, improved by including parameter names and values. Should make it easier to code dataset classes. Usage 1 is for convenience, same information is contained in results_obj in Usage 2.

Parameters:

`params`**:** Structure with parameter names and values.

`results`**:** Structure with result names and values (Usage 1).

`results_obj`**:** A results_profile object with test results.

`id`**:** Identification string (Usage 1).

`props`**:** A structure with any optional properties (Usage 1).

Returns a structure object with the following fields:

params, results (results_obj above)

See also: `results_profile` (p. 206), `params_tests_dataset` (p. 132)

Author: Cengiz Gunay <cgunay@emory.edu>, 2011/07/05

A.21.2 Method `params_results_profile/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.22 Class `params_tests_dataset`

A.22.1 Constructor `params_tests_dataset/params_tests_dataset`

Summary: Contains a set of data objects or files of raw data varying with parameter values.

Usage:

```
obj = params_tests_dataset(list, dt, dy, id, props)
```

Description: This is an abstract base class for keeping dataset information separate from the parameters-results database (params_tests_db). The list contents can be filenames or objects (such as cip_traces) from which to get the raw data. The dataset should have all the necessary information to create a db when needed. This is an abstract class, thet it it cannot act on its own. Only fully implemented subclasses can actually hold datasets. See methods below.

Parameters:

`list`: Array of dataset items (filenames, objects, etc.).

`dt`: Time resolution [s]

`dy`: y-axis resolution [integral V, A, etc.]

`id`: An identification string.

`props`: A structure with any optional properties.

`type`: type of file (default = '')

`loadItemProfileFunc`: Function name or handle to be called as with (dataset, index, param_row, props) to load a dataset item during database creation and return a results_profile. Changing this property allows creating different databases from same dataset. It also allows loading a novel dataset through this generic class.

Returns a structure object with the following fields:

list, dt, dy, id, props (see above).

See also: `params_tests_db` (p. 138), `params_tests_fileset` (p. 157), `cip_traces_dataset` (p. 73)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/12/02

A.22.2 Method `params_tests_dataset/addItem`

Summary: Returns the new dataset with the added item.

Usage:

```
dataset = addItem(dataset, item)
```

Description: Note that, this is NOT the way to create a dataset. It is only intended for small additions to an existing dataset. This method is too slow for creating large datasets. The normal method for creating datasets is providing the full list of items to the class constructor.

Parameters:

`dataset`**:** A params_tests_dataset.

`item`**:** New item to add in dataset.

Returns:

dataset: With the added item.

See also: `itemResultsRow` (p. **??**), `params_tests_dataset` (p. 132), `paramNames` (p. **??**), `testNames` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/01/25

A.22.3 Method `params_tests_dataset/display`

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.22.4 Method `params_tests_dataset/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.22.5 Method `params_tests_dataset/getItem`

Summary: Returns the dataset item at given index.

Usage:

```
item = getItem(dataset, index)
```

Parameters:

`dataset`**:** A params_tests_dataset.

`index`**:** Index of item in dataset.

Returns:

item: Object, filename, etc.

See also: `itemResultsRow` (p. **??**), `params_tests_dataset` (p. 132), `paramNames` (p. **??**), `testNames` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/12/03

A.22.6 Method `params_tests_dataset/getItemParams`

Summary: Get the parameter values of a dataset item.

Usage:

```
params_row = getItemParams(dataset, index, a_profile)
```

Description: This method can retrieve the item parameters by using either the dataset and the index to find the item or simply by using the item profile, a_profile.

Parameters:

`dataset`**:** A params_tests_dataset.

`index`**:** Index of item in dataset.

`a_profile`**:** An item profile.

Returns:

params_row: Parameter values in the same order of paramNames

See also: `itemResultsRow` (p. **??**), `params_tests_dataset` (p. 132), `paramNames` (p. **??**), `testNames` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/10

A.22.7 Method `params_tests_dataset/itemResultsRow`

Summary: Analyze data from the dataset and return its parameter and test values.

Usage:

```
[params_row, tests_row] = itemResultsRow(dataset, index)
```

Description: This method is designed to be reused from subclasses as long as the loadItemProfile method is properly overloaded. Adds an Index column to the DB to keep track of raw data items after shuffling.

Parameters:

`dataset`**:** A params_tests_dataset.

`index`**:** Index of file in dataset.

`props`**:** A structure with any optional properties.
(passed to loadItemProfileFunc)

Returns:

params_row: Parameter values in the same order of paramNames tests_row: Test values in the same order with testNames

See also: loadItemProfile (p. **??**), params_tests_dataset (p. 132), paramNames (p. **??**), testNames (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/10

A.22.8 Method params_tests_dataset/loadItemProfile

Summary: Generates a results_profile object from a dataset item.

Usage:

```
a_profile = loadItemProfile(dataset, item_index)
```

Description: If getResults returns a params_results_profile, then implementing paramNames and getItemParams become unecessary.

Parameters:

`dataset`: A params_tests_dataset object.

`item_index`: Index of item in dataset.

`params_row`: Parameter values for this item (default=[])

`props`: Struct with optional properties.

Returns:

a_profile: A profile object that implements the getResults method.

See also: itemResultsRow (p. **??**), params_tests_fileset (p. 157), paramNames (p. **??**), testNames (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2011/07/05

A.22.9 Method params_tests_dataset/params_tests_db

Summary: Generates a params_tests_db object from the dataset.

Usage:

```
db_obj = params_tests_db(obj, items, props)
```

Description: This is a converter method to convert from params_tests_dataset to params_tests_db. Uses readDBItems to read the files. A customized subclass should provide the correct paramNames, testNames, and itemResultsRow functions. Adds a ItemIndex column to the DB to keep track of raw data files after shuffling.

Parameters:

`obj`: A params_tests_dataset object.

`items`: (Optional) List of item indices to use to create the db.

`props`: Any optional params to pass to params_tests_db.

Returns:

db_obj: A params_tests_db object.

See also: `readDBItems` (p. **??**), `params_tests_db` (p. 138), `params_tests_dataset` (p. 132), `itemResultsRow testNames` (p. **??**), `paramNames` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/09

A.22.10 Method `params_tests_dataset/readDBItems`

Summary: Reads all items to generate a params_tests_db object.

Usage:

```
[params, param_names, tests, test_names] = readDBItems(obj, items)
```

Description: This is a generic method to convert from params_tests_fileset to a params_tests_db, or a subclass. This method depends on the paramNames, testNames, and itemResultsRow functions. Outputs of this function can be directly fed to the constructor of a params_tests_db or a subclass.

Parameters:

`obj`: A params_tests_fileset object.

`items`: (Optional) List of item indices to use to create the db.

Returns:

params, param_names, tests, test_names: See params_tests_db.

See also: `params_tests_db` (p. 138), `params_tests_fileset` (p. 157), `itemResultsRow testNames` (p. **??**), `paramNames` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/24

A.22.11 Method `params_tests_dataset/set`

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.22.12 Method `params_tests_dataset/setProp`

Summary: Generic method for setting optional object properties.

Usage:

```
obj = setProp(obj, prop1, val1, prop2, val2, ...)
```

Description: Modifies or adds property values. As many property name-value pairs can be specified.

Parameters:

`obj`: Any object that has a props field.

`attr`: Property name

`val`: Property value.

Returns:

obj: The new object with the updated properties.

See also:

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/22

A.22.13 Method `params_tests_dataset/subsasgn`

Summary: Defines generic index-based assignment for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/02/06

A.22.14 Method `params_tests_dataset/subsref`

Summary: Defines generic indexing for objects.

A.22.15 Method `params_tests_dataset/testNames`

Summary: Returns the ordered names of tests for this dataset.

Usage:

```
[test_names a_prof] = testNames(dataset, item)
```

Description: Looks at the results of the first file to find the test names.

Parameters:

`dataset`: A params_tests_dataset.

`item`: (Optional) If given, read names by loading item at this index.

Returns:

test_names: Cell array with ordered parameter names. a_prof: Profile of the item read to find test names.

See also: `params_tests_dataset` (p. 132), `paramNames` (p. **??**), `testNames` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/10

A.23 Class `params_tests_db`

A.23.1 Constructor `params_tests_db/params_tests_db`

Summary: A generic database of test results varying with parameter values, organized in a matrix format.

Description: This is a subclass of tests_db. Defines all operations on this structure so that subclasses can use them.

Parameters:

`num_params`: Number of parameters.

`a_tests_db`: A tests_db upon which to build the params_tests_db.

`props`: A structure with any optional properties.

Returns a structure object with the following fields:

tests_db num_params: Number of variable parameters in simulations.

See also: `tests_db` (p. 256), `test_variable_db (N/I)` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/08

A.23.2 Method `params_tests_db/addColumns`

Summary: Inserts new columns to tests_db.

Description: Delegates to tests_db/addColumns, but maintains parameter columns for the 2nd usage.

Parameters:

`obj, b_obj`: A tests_db object.

`test_names`: A single string or a cell array of test names to be added.

`test_columns`: Data matrix of columns to be added.

Returns:

obj: The tests_db object that includes the new columns.

See also: `tests_db/addColumns` (p. 257), `allocateRows` (p. **??**), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2015/05/18

A.23.3 Method `params_tests_db/addParams`

Summary: Inserts new parameter columns to tests_db.

Usage:

```
obj = addParams(obj, param_names, param_columns)
```

Description: Adds new columns to the database and returns the new DB. This operation is expensive in the sense that the whole database matrix needs to be enlarged just to add a single new column. The method of allocating a matrix, filling it up, and then providing it to the tests_db constructor is the preferred method of creating tests_db objects. This method may be used for measures obtained by operating on raw measures.

Parameters:

`obj`: A tests_db object.

`param_names`: A cell array of param names to be added.

`param_columns`: Data matrix of columns to be added.

Returns:

obj: The tests_db object that includes the new columns.

See also: `allocateRows` (p. **??**), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/10/11

A.23.4 Method `params_tests_db/crossProd`

Summary: Create a DB by taking the cross product of two database row sets.

Usage:

```
cross_db = crossProd(a_db, b_db)
```

Description: Overloaded function to maintain correct number of parameters after cross product operation. See original in tests_db/crossProd.

Parameters:

`a_db, b_db`: A tests_db object.

Returns:

cross_db: The tests_db object with all combinations of rows.

See also: `tests_db/crossProd` (p. 266)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/10/11

A.23.5 Method `params_tests_db/delColumns`

Summary: Deletes columns from tests_db.

Usage:

```
index = delColumns(obj, tests)
```

Description: Overloaded function that maintains correct number of parameters. See original tests_db/delColumns.

Parameters:

`obj`: A tests_db object.

`tests`: Numbers or names of tests (see tests2cols)

Returns:

obj: The tests_db object that is missing the columns.

See also: `tests_db/delColumns` (p. 267)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/10/11

A.23.6 Method `params_tests_db/display`

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.23.7 Method `params_tests_db/displayRankingsTeX`

Summary: Generates and displays a ranking DB by comparing rows of a_db with the given match criteria.

Usage:

```
tex_string = displayRankingsTeX(a_db, crit_db, props)
```

Description: Generates a LaTeX document with: - Values of 10 best matching a_db rows in a floating table. - (optional) Raw traces compared with some best matches at different distances - Parameter distributions of 50 best matches as a bar graph.

Parameters:

`a_db`: A params_tests_db object to compare rows from.

`crit_db`: A tests_db object holding the match criterion tests and STDs which can be created with matchingRow.

`props`: A structure with any optional properties.

`caption`: Identification of the criterion db (not needed/used?).

`a_dataset`: Dataset for a_db.

`a_dball`: The non-joined DB for for a_db.

`crit_dataset`: Dataset for crit_db.

`crit_dball`: Dataset for crit_db.

`num_matches`: Number of best matches to display (default=10).

`rotate`: Rotation angle for best matches table (default=90).

Returns:

tex_string: LaTeX document string.

See also: `rankVsDB` (p. **??**), `displayRowsTeX` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/20

A.23.8 Method `params_tests_db/fillMissingParams`

Summary: Add missing columns as params with given default fill value.

Usage:

```
db = fillMissingParams(db, col_names, fill_value)
```

Parameters:

`db`: A tests_db object.

`col_names`: A cell array of param names.

`fill_value`: Value to be used for missing columns.

Returns:

db: The tests_db object that includes the newly filled columns.

See also: `tests_db/fillMissingColumns` (p. 272), `params_tests_db/addParams` (p. 139), `params_tests_db/unionCat` (p. 154)

Author: Cengiz Gunay <cgunay@emory.edu>, 2008/06/02

A.23.9 Method `params_tests_db/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.23.10 Method `params_tests_db/getDualCIPdb`

Summary: Generates a database by merging selected tests of depolarizing and hyperpolarizing cip results.

Usage:

```
a_db = getDualCIPdb(db, depol_tests, hyper_tests, depol_suffix, hyper_suffix)
```

Description: depol_tests need to have the RowIndex column in it.

Parameters:

`db`: A params_tests_db object.

Returns:

a_db: A params_tests_db object of organized values.

Example:

```
» control_phys_sdb = getDualCIPdb(control_phys_db, depol_tests, hyper_tests, '', 'Hyp100pA')
where depol_tests and hyper_tests are cell arrays of selected tests.
```

See also: `invarValues` (p. **??**), `tests_3D_db` (p. 246), `corrCoefs` (p. **??**), `tests_3D_db/plotPair` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/01/13

A.23.11 Method `params_tests_db/getParamNames`

Summary: Gets parameter column names.

Usage:

```
col_names = getParamNames(db)
```

Description: Convenience function that delegates to getColNames.

Parameters:

`db`: A params_tests_db object.

Returns:

col_names: A cell array of strings.

See also: tests_db/getColNames (p. 273), tests_db (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2008/04/03

A.23.12 Method params_tests_db/getParamRowIndices

Summary: Returns indices of rows with matching parameter values from rows of this db.

Usage:

```
row_indices = getParamRowIndices(a_db, rows, to_db)
```

Parameters:

`a_db`: A params_tests_db object.

`rows`: rows to find indices for.

`to_db`: Where to find the matching rows.

Returns:

row_indices: Array of row indices.

See also: makeModifiedParamDB (p. **??**), scanParamAllRows (p. **??**), scaleParamsOneRow (p. **??**), writeParFile (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/01/14

A.23.13 Method params_tests_db/getProfile

Summary: Create a profile object from a params_tests_db by collecting statistics.

Usage:

```
a_pt_profile = getProfile(a_db, props)
```

Description: Calculates the following results items: idx: Name-index pairs for accessing results arrays. t_hists: Cell array of histograms of each test. p_t3ds: Cell array of invariant relations of each parameter with all tests. pt_hists: Cell array of separate test value histograms for uniques value of each parameter. p_stats: Cell array of test stats for each param. p_coefs: Cell array of correlation coefficients for each parameter with all tests. pt_coefs_hists: Cell matrix of histograms of coefficients from correlations of each parameter with each test. pp_coefs: Cell 3D matrix of mean coefficients from correlations of each parameter with correlation coefficients of each parameter with each test.

Parameters:

`a_db`: A params_tests_db object.

`props`: A structure with any optional properties.

Returns a params_tests_profile object.

See also: `params_tests_profile` (p. 162), `results_profile` (p. 206), `params_tests_db` (p. 138), `params_tests_fileset` (p. 157), `tests_db` (p. 256), `tests_3D_db` (p. 246), `histogram_db` (p. 101), `stats_db` (p. 239), `corrcoefs_db` (p. 83)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/13

A.23.14 Method `params_tests_db/invarParam`

Summary: Generates a 3D database of invariant values of a parameter and all test columns.

Usage:

```
a_3D_db = invarParam(db, param, props)
```

Parameters:

`db`: A tests_db object.

`param`: A parameter name/column number. It can be empty [], meaning to find all unique combinations of parameters.

`props`: A structure with any optional properties.

`removeRedun`: If 1 (default), clean database by removing redundant sets of parameters.

`removeCol`: If removeRedun == 1, name of parameter column to remove if found. Default: 'trial'. (others passed to tests_db/invarValues)

A.23.15 Method `params_tests_db/invarParams`

Summary: Calculates invariant param dbs for all parameters and returns in an array.

Usage:

```
p_t3ds = invarParams(a_db)
```

Description: Skips the 'ItemIndex' test.

Parameters:

`a_db`: A tests_db object.

Returns:

p_t3ds: An array of tests_3D_dbs for each param in a_db.

See also: params_tests_profile (p. 162)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/17

A.23.16 Method params_tests_db/joinRows

Summary: Joins the rows of the given db with rows of with_db with matching RowIndex values.

Usage:

```
a_db = joinRows(db, with_db, props)
```

Description: Takes the desired columns in with_db with rows having a row index and joins them next to dedired columns from the current db. Assumes each row index only appears once in with_db. The created db preserves the ordering of with_db.

Parameters:

db: A param_tests_db object.

with_db: A tests_db object with a RowIndex column.

props: A structure with any optional properties.

indexColName: (Optional) Name of row index column (default='RowIndex').

Returns:

a_db: A params_tests_db object.

See also: tests_db (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/16

A.23.17 Method params_tests_db/matchingRow

Summary: Creates a criterion database for matching the tests of a row.

Usage:

```
crit_db = matchingRow(a_db, row, props)
```

Description: Overloaded method for skipping parameter values. STD for param values will be NaNs.

Parameters:

a_db: A tests_db object.

row: A row index to match.

props: A structure with any optional properties.

`distDB`: Take the standard deviation from this db instead.

Returns:

crit_db: A tests_db with two rows for values and STDs.

See also: `tests_db/matchingRow` (p. 280), `rankMatching` (p. **??**), `tests_db` (p. 256), `tests2cols` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/06/13

A.23.18 Method `params_tests_db/meanDuplicateParams`

Summary: Takes the mean of all measures for rows that have the same parameter columns.

Usage:

```
a_params_tests_db = meanDuplicateParams(db, props)
```

Description: Calls tests_db/meanDuplicateRows with params as main_cols and tests as rest_cols.

Parameters:

`db`: A params_tests_db object.

`props`: Structure with optional parameters.

Returns:

a_params_tests_db: The db object of with the means on page 1 and standard deviations on page 2.

See also: `tests_db/meanDuplicateRows` (p. 282), `tests_db/mean` (p. 281), `tests_db/std` (p. 310), `sortedUniqueValues` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2007/12/20

A.23.19 Method `params_tests_db/mergeMultipleCIPsInOne`

Summary: Merges multiple rows with different CIP data into one, generating a database of one row per neuron.

Usage:

```
a_db = mergeMultipleCIPsInOne(db, names_tests_cell, index_col_name, props)
```

Description: It calls invarParam to separate DB into pages with different CIP level data. Then uses the names_tests_cell to choose tests from each page to be merged into the final database row. The tests will be suffixed with the field name so that they can be distinguished. RowIndex columns will be automatically included, and one of them can be chosen with index_col_name that has values for all cells. The suffixed for needs to be used to choose index_col_name, such as 'RowIndex_H100pA', assuming 'H100pA' was the field name in names_tests_cell that corresponds to page -100 pA.

Parameters:

`db`: A params_tests_db object.

`names_tests_cell`: A cell array alternating suffix names and test column vectors.

The order of names correspond to each unique CIP level in db, with increasing order.

`index_col_name`: (Optional) Name of row index column

(default is 'RowIndex' suffixed with the first field name).

`props`: A structure with any optional properties.

`cipLevels`: In case db is missing some levels, provides a list of cip levels that correspond to names_tests_cell db. Missing levels are replaced with NaN values. DB is filtered to remove other CIP levels.

Returns:

a_db: A params_tests_db object of organized values.

Example:

```
» control_phys_sdb =
mergeMultipleCIPsInOne(control_phys_db,
struct('_H100pA', [1:10], '_D100pA', [1:10 16:18]),
'RowIndex_H100pA')
```

See also: `invarValues` (p. **??**), `tests_3D_db` (p. 246), `corrCoefs` (p. **??**), `tests_3D_db/plotVarBox` (p. 253)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/01/13

A.23.20 Method `params_tests_db/onlyRowsTests`

Summary: Returns a tests_db that only contains the desired tests and rows (and pages).

Usage:

```
obj = onlyRowsTests(obj, rows, tests, pages)
```

Description: Selects the given dimensions and returns in a new tests_db object. Makes sure num_params remains correct.

Parameters:

`obj`: A tests_db object.

`rows, tests`: A logical or index vector of rows, or cell array of names of rows. If ':', all rows. For names, regular expressions are supported if quoted with slashes (e.g., '/a.*/'). See tests2idx.

`pages`: (Optional) A logical or index vector of pages. ':' for all pages.

Returns:

obj: The new tests_db object.

See also: `subsref` (p. **??**), `tests_db` (p. 256), `test2idx` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/17

A.23.21 Method `params_tests_db/paramsCoefs`

Summary: Calculates a corrcoefs_db for each param and collects them in a cell array.

Usage:

```
p_coefs = paramsCoefs(a_db, p_t3ds)
```

Description: Skips the 'ItemIndex' test.

Parameters:

`a_db`: A tests_db object.

`p_t3ds`: Cell array of invariant parameter databases.

Returns:

p_coefs: A cell array of corrcoefs_dbs for each param in a_db.

See also: `params_tests_profile` (p. 162)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/17

A.23.22 Method `params_tests_db/paramsHists`

Summary: Calculates histograms for all parameters and returns in a cell array.

Usage:

```
p_hists = paramsHists(a_db, props)
```

Description: Useful for looking at subset databases and find out what parameter values are used most. Skips the 'ItemIndex' column(s). Finds unique values of each parameter to make histograms.

Parameters:

`a_db`: A tests_db object.

`props`: A structure with any optional properties.

`paramVals`: Array of histogram bins to use as parameter values.

Returns:

p_hists: An array of histograms for each parameter in a_db.

See also: `params_tests_profile` (p. 162)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/20

A.23.23 Method `params_tests_db/paramsParamsCoefs`

Summary: Calculates a corrcoefs_db for each param from correlations of variant params and invariant param coefs and collects them in a cell array.

Usage:

```
pp_coefs = paramsParamsCoefs(a_db, p_t3ds, p_coefs)
```

Description: Skips the 'ItemIndex' test.

Parameters:

`a_db`: A tests_db object.

`p_t3ds`: Cell array of invariant parameter databases.

`p_coefs`: Cell array of tests coefficients for each parameter.

Returns:

pp_coefs: A cell array of corrcoefs_dbs for each param combination in a_db.

See also: `params_tests_profile` (p. 162)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/17

A.23.24 Method params_tests_db/paramsTestsCoefsHists

Summary: Calculates histograms for all pairs of params and tests coefficients and returns in a cell array.

Usage:

```
pt_coefs_hists = paramsTestsCoefsHists(a_db, p_coefs)
```

Description: Skips the 'ItemIndex' test.

Parameters:

`a_db`: A tests_db object.

`p_coefs`: Cell array of tests coefficients for each parameter.

Returns:

pt_coefs_hists: A cell array of corrcoefs_dbs for each param in a_db.

See also: params_tests_profile (p. 162)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/17

A.23.25 Method params_tests_db/plotParamsHists

Summary: Create a horizontal plot_stack of parameter histograms.

Usage:

```
a_ps = plotParamsHists(a_db, title_str, props)
```

Description: Skips the 'ItemIndex' test.

Parameters:

`a_db`: A params_tests_db object.

`title_str`: (Optional) A string to be concatanated to the title.

`props`: A structure with any optional properties.

`quiet`: Do not display the DB id on the plot title.

`barAxisProps`: passed to plotEqSpaced for each bar axis.

`stackAxisLimits`: Axis limits for plot_stack (default=[NaN NaN 0 Inf]).

(Others passed to paramsHists, plotEqSpaced, and plot_stack)

Returns:

a_ps: A horizontal plot_stack of plots

See also: plot_stack (p. 194), paramsHists (p. **??**), plotEqSpaced (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/04/07

A.23.26 Method `params_tests_db/plotVarBoxMatrix`

Summary: Create a stack of parameter-test variation plots organized in a matrix.

Usage:

```
a_plot_stack = plotVarBoxMatrix(a_db, p_t3ds)
```

Description: Skips the 'ItemIndex' test.

Parameters:

`a_db`**:** A tests_db object.

`p_t3ds`**:** Cell array of invariant parameter databases.

Returns:

a_plot_stack: A plot_stack with the plots organized in matrix form

See also: `params_tests_profile` (p. 162), `plotVar` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/17

A.23.27 Method `params_tests_db/rankVsAllDB`

Summary: Generates ranking DBs by comparing rows of a_db with each row of to_db.

Usage:

```
tex_string = rankVsAllDB(a_db, to_db, a_dataset, to_dataset)
```

Description: Distance is each measure difference divided by the STD in to_db, squared and summed. Returned DB contains only the selected to_tests and the parameters from initial DB.

Parameters:

`a_db`**:** A params_tests_db object to compare rows from.

`to_db`**:** A tests_db object to compare it with.

`a_dataset`**:** Dataset for a_db.

`to_dataset`**:** Dataset for crit_db.

Returns:

ranked_dbs: Array of created DBs with original rows and a distance measure, in ascending order. tex_string: A LaTeX string for all tables created.

See also: `rankVsDB` (p. **??**), `matchingRow` (p. **??**), `rankMatching` (p. **??**), `joinRows` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/12/10

A.23.28 Method `params_tests_db/rankVsDB`

Summary: Generates a ranking DB by comparing rows of this db with the given test criteria.

Usage:

```
a_ranked_db = rankVsDB(a_db, crit_db)
```

Description: Distance is each measure difference divided by the STD in to_db, squared and summed. Returned DB contains only the selected tests from crit_db and the parameters from initial a_db.

Parameters:

`a_db`: A params_tests_db object to compare rows from.

`crit_db`: A tests_db object holding the match criterion tests and STDs which can be created with matchingRow.

Returns:

a_ranked_db: The created DB with original rows and a distance measure, in ascending order.

See also: `matchingRow` (p. **??**), `rankMatching` (p. **??**), `joinRows` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/20

A.23.29 Method `params_tests_db/reIndexNeurons`

Summary: Re-index neurons with accending numbers.

Usage:

```
new_db = reIndexNeurons(a_db, startNum, colName)
```

Description: This is useful for avoiding NeuronId conflicts when concatenating two databases. It can also remove the unused number 'hole' (e.g. after deleting rows) and make the indices continuous.

Parameters:

`a_db`: a tests_db object

`startNum`: the starting index number (default = 1)

`colName`: the column name or number of neuron index. (default = 'NeuronId')

Returns:

new_db: a new database with new neuron index.

See also: `physiol_bundle` (p. 164), `tests_db/physiol_bundle` (p. 285)

Author: Li Su. 03/21/2008

A.23.30 Method `params_tests_db/scanParamAllRows`

Summary: OBSOLETE (use instead varyParams) - Scans given parameter range for each row in DB.

Usage:

```
a_params_db = scanParamAllRows(a_db, param, min_val, max_val, num_levels, props)
```

Description: Produces rows by replacing the desired parameter value, in all rows of DB, with num_levels values between the given boundaries, min_val and max_val. This results in a DB with num_levels times more rows than the original DB. Then, writeParFile can be used to generate a parameter file from this DB to drive new simulations.

Parameters:

`a_db`**:** A params_tests_db object whose first row is subject to modifications.

`param`**:** The parameter to be varied (see tests2cols for param description).

`min_val, max_val`**:** The low and high boundaries for the parameter value.

`num_levels`**:** Number of levels to produce, including the boundaries.

`props`**:** A structure with any optional properties.

`renameTrial`**:** If given, the 'trial' column is renamed to this name.

`levelFunc`**:** Use this function to get the parameter range with feval(levelFunc, min_val, max_val, num_levels). Example: 'logLevels'

Returns:

a_params_db: A db only with params.

Example:

```
Sets NaF to given range with 100 levels:
» naf_rows_db = scanParamAllRows(a_db(desired_rows, :), 'NaF', 0, 1000, 100);
```

See also: `writeParFile` (p. **??**), `scaleParamsOneRow` (p. **??**), `ranked_db/blockedDistances` (p. 200), `getParamRowIndices` (p. **??**), `logLevels` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/02/16

A.23.31 Method `params_tests_db/set`

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.23.32 Method `params_tests_db/subsref`

Summary: Defines generic indexing for objects.

A.23.33 Method `params_tests_db/testsHists`

Summary: Calculates histograms for all tests and returns them in a cell array.

Usage:

```
t_hists = testsHists(a_db, num_bins)
```

Description: Skips the 'ItemIndex' test.

Parameters:

`a_db`**:** One or more tests_db objects in an array.

`num_bins`**:** Number of histogram bins (Optional, default=100), or vector of histogram bin centers.

Returns:

t_hists: An array of histograms for each test in a_db.

See also: `params_tests_profile` (p. 162)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/17

A.23.34 Method `params_tests_db/unionCat`

Summary: Vertically concatenate two or more databases with different parameters or tests.

Usage:

```
a_db = unionCat(db, with_db, ...)
```

Description: The parameters and tests in the result are a union of both. Adds 0 for parameter and NaN for tests in the rows which didn't have the additional columns before.

Parameters:

db, with_db, ...: tests_db objects to be concatenated together.

Author: Li Su, 2008-04-10

A.23.35 Method `params_tests_db/unionCatTwo`

Usage:

```
a_db = unionCat(db, with_db, props)
```

Description: The parameters and tests in the result are a union of both. Adds 0 for parameter and NaN for tests in the rows which didn't have the additional columns before.

Parameters:

`db, with_db`**:** tests_db objects to be concatenated together.

`props`**:** A structure with any optional properties.

`offsetTracesets`**:** If 1, make sure the TracesetIndex column is offset to non-overlapping values in the concatenated databases (default=0).

Author: Li Su, 2008-04-10

A.23.36 Method `params_tests_db/varyParams`

Summary: Varies chosen parameters in all rows by given levels.

Usage:

```
a_params_db = varyParamsOneRow(a_db, params, levels, props)
```

Description: Produces new rows by either multiplying or replacing the desired params with each value in levels. Thus, the newly created parameter db will be size of levels times bigger. Columns other than parameers will be pruned. Then, writeParFile can be used to generate a parameter file from this DB to drive new simulations.

Parameters:

`a_db`**:** A params_tests_db object.

`params`**:** Parameters to be varied (see tests2cols).

`levels`**:** Column vector of parameter values to multiply (1=unity) or to replace parameters with (see 'replace' prop).

`props`**:** A structure with any optional properties.

`replace`**:** Replace parameter values with levels instead of scaling.

Returns:

a_params_db: A db only with params.

Example:

```
Blocks NaF from 0
» naf_rows_db = varyParams(a_db(desired_row, :), 'NaF', 0:0.1:1);
```

See also: writeParFile (p. **??**), ranked_db/blockedDistances (p. 200), getParamRowIndices (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/02/16

A.23.37 Method params_tests_db/writeParFile

Summary: Creates or appends to text file all the parameter values in a_db.

Usage:

```
writeParFile(a_db, filename, props)
```

Description: Creates a text file that has a set of parameter values for each row. The first line is a header that contains number of parameters and total rows. If the file exists, the data is appended, but the header is NOT updated for efficiency considerations. Optionally, a parameter description file can be created that contains one parameter name per row. These files can be processed with various utilities to control simulations.

Parameters:

`a_db`: A params_tests_db object.

`filename`: Genesis parameter file to be created.

`props`: A structure with any optional properties.

- `trialStart`: If given, adds/replaces the trial parameter and counts forward.
- `makeParamDesc`: If 1, put the parameter names in a parameter description file with
  with a .txt extension.
- `noAppend`: If given, do not append to file even if it exists

Returns:

nothing.

Example:

```
» naf_rows_db = scanParamAllRows(a_db(desired_rows, :), 'NaF', 0, 1000, 100);
» writeParFile(naf_rows_db, 'naf.par')
```

See also: scanParamAllRows (p. **??**), scaleParamsOneRow (p. **??**), https://github.com/cengique/param-sear (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/03/13

A.24 Class `params_tests_fileset`

A.24.1 Constructor `params_tests_fileset/params_tests_fileset`

Summary: Description of a set of data files of raw data varying with parameter values.

Usage:

```
obj = params_tests_fileset(file_pattern, dt, dy, id, props)
```

Description: This is a subclass of params_tests_dataset. This class is used to generate params_tests_db objects and keep a connection to the raw data files. This class only keeps names of files and loads raw data files whenever it's requested. A database object can easily be generated using the convertion methods. Most methods defined here can be used as-is, however some should be overloaded in subclasses. The specific methods are loadItemProfile.

Parameters:

`file_pattern`: File pattern, or cell array of patterns, matching all files to be loaded.

`dt`: Time resolution [s]

`dy`: y-axis resolution [ISI (V, A, etc.)]

`id`: An identification string

`props`: A structure with any optional properties.

- `num_params`: Number of parameters that appear in filenames (auto-detected by default; see props for parseFilenameNamesVals).
- `fileParamsRegexp`: Regular expression to find parameter names and values instead of the default function parseFilenameNamesVals. It will look for named capture variables 'name' and 'val'. See the 'names' option to regexp.
- `param_trial_name`: Use this name on the filename as the 'trial' parameter.
- `trial_hash`: Structure to get integer indices from non-integer trial numbers as key.
- `trialHashFunc`: Produces structure key from trial number and precision (see num2str).
- `param_row_filename`: If given, the 'trial' parameter will be used to address rows from this file and acquire parameters.
- `param_rows`: Instead of a file, just give parameters in this matrix.
- `param_desc_filename`: Contains the parameter range descriptions one per each row. The parameter names are acquired from this file.
- `param_names`: Cell array of parameter names corresponding to the param_row_filename columns can be specified as an alternative to specifying param_desc_filename. These names are not for the parameters present in the data filename.

`profile_method_name`: It can be one of the profile-creating methods in this
class. E.g., 'trace_profile', 'srp_trace_profile', etc. OBSOLOTE: see loadItemProfileFunc prop in params_tests_dataset. (Others passed to params_tests_dataset and parseFilenameNamesVals)

Returns a structure object with the following fields:

params_tests_dataset, path: The pathname to files.

See also: `params_tests_db` (p. 138), `tests_db` (p. 256), `params_tests_dataset` (p. 132), `regexp` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/09

A.24.2 Method `params_tests_fileset/addFiles`

Summary: Adds to existing list of files in set.

Usage:

```
[a_fileset, index_list] = addFiles(a_fileset, file_pattern, props)
```

Parameters:

`a_fileset`: A params_tests_fileset object.

`file_pattern`: File pattern, or cell array of patterns, matching additional files.

`props`: A structure with any optional properties.

`param_row_filename`: Update parameters from here. The 'trial' parameter is used
to address rows from this file and acquire parameters.

Returns:

a_fileset: The augmented fileset object. index_list: The vector of index numbers of the new files added. Can be used to selectively load the new files into a DB using params_test_db.

See also: `params_tests_fileset` (p. 157), `params_tests_dataset/params_test_db.` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/02/01

A.24.3 Method `params_tests_fileset/display`

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.24.4 Method `params_tests_fileset/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.24.5 Method `params_tests_fileset/getItemParams`

Summary: Get the parameter values of a fileset item.

Usage:

```
params_row = getItemParams(fileset, index)
```

Parameters:

`fileset`**:** A params_tests_fileset.

`index`**:** Index of item in fileset.

Returns:

params_row: Parameter values in the same order of paramNames

See also: `itemResultsRow` (p. **??**), `params_tests_dataset` (p. 132), `paramNames` (p. **??**), `testNames` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/12/03

A.24.6 Method `params_tests_fileset/loadItemProfile`

Summary: Loads a profile object from a raw data file in the fileset.

Usage:

```
a_profile = loadItemProfile(fileset, file_index)
```

Description: First it looks for the prop loadItemProfileFunc to load the profile. Otherwise, it assumes fileset items can be loaded as traces. and runs props.profile_method_name if available, or simply creates a trace_profile object. Subclasses should overload this function to load the specific profile object they desire. The profile class should define a getResults method which is used in the params_tests_dataset/itemResultsRow method.

Parameters:

`fileset`**:** A params_tests_fileset.

`file_index`**:** Index of file in fileset.

`params_row`**:** Parameter values for this item (default=[])

`props`: A structure with any optional properties.
(passed to loadItemProfileFunc)

Returns:

a_profile: A results_profile object that implements the getResults method.

See also: `results_profile` (p. 206), `params_tests_dataset/itemResultsRow` (p. 134)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.24.7 Method `params_tests_fileset/paramNames`

Summary: Returns the ordered names of parameters for this fileset.

Usage:

```
param_names = paramNames(fileset, item)
```

Description: Looks at the filename of the first file to find the parameter names.

Parameters:

`fileset`: A params_tests_fileset.

`item`: (Optional) If given, read param names by loading item at this index.

Returns:

params_names: Cell array with ordered parameter names.

See also: `params_tests_fileset` (p. 157), `testNames` (p. **??**), `parseFilenameNamesVals` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/10

A.24.8 Method `params_tests_fileset/set`

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.24.9 Method `params_tests_fileset/trace`

Summary: Loads a raw trace given a file_index to this fileset.

Usage:

```
a_trace = trace(fileset, file_index)
```

Parameters:

`fileset`**:** A params_tests_fileset.

`file_index`**:** Index of file in fileset.

Returns:

a_trace: A trace object.

See also: `trace` (p. 317), `params_tests_fileset` (p. 157)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/13

A.24.10 Method `params_tests_fileset/trace_profile`

Summary: Loads a raw trace_profile given a file_index to this fileset.

Usage:

```
a_trace_profile = trace_profile(fileset, file_index)
```

Parameters:

`fileset`**:** A params_tests_fileset.

`file_index`**:** Index of file in fileset.

Returns:

a_trace_profile: A trace_profile object.

See also: `trace_profile` (p. 338), `params_tests_fileset` (p. 157)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/13

A.25 Class `params_tests_profile`

A.25.1 Constructor `params_tests_profile/params_tests_profile`

Summary: Holds the results profile from a params_tests_db.

Usage:

```
a_pt_profile = params_tests_profile(results, a_db, props)
```

Parameters:

`a_db`: A params_tests_db object.

`results`: A structure containing test results.

`props`: A structure with any optional properties.

Returns a structure object with the following fields:

results_profile: Contains results of tests. db: The params_tests_db. props.

See also: `results_profile` (p. 206), `params_tests_db/params_tests_profile` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/13

A.25.2 Method `params_tests_profile/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.26 Class `period`

A.26.1 Constructor `period/period`

Summary: Start and end times of a period in terms of the dt of the trace to which belongs.

Usage:

```
obj = period(start_time, end_time, dt)
```

Parameters:

`start_time, end_time`: Inclusive period [dt]. If end_time is missing and start_time has two values, the second one is used as end_time.

`dt`: If provided, interpret start_time and end_time in seconds and convert them using this dt.

Returns a structure object with the following fields:

start_time, end_time.

See also: trace (p. 317), spikes (p. 227), spike_shape (p. 215)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/07/30

A.26.2 Method period/array

Summary: Converts the period to an index array.

Usage:

```
an_array = array(period)
```

Parameters:

period**:** A period object

Returns:

an_array: An array of dt indices contained within the period.

See also: period (p. 162), trace (p. 317)

Author: Cengiz Gunay <cengique@users.sf.net>, 2010/03/29

A.26.3 Method period/char

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.26.4 Method period/display

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.26.5 Method period/get

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.26.6 Method `period/set`

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.26.7 Method `period/SpikeTimesinPeriod`

Usage:

```
SpkTimes=Interval(times, period)
```

Parameters:

`times`: an array of spike times.

`period`: A period object

Returns:

the_period: The cropped set of spike times that fall within a period.

See also: `period` (p. 162), `cip_trace` (p. 56), `trace` (p. 317), `spikes` (p. 227)

Author: Tom Sangrey, 2006/01/26

A.26.8 Method `period/subsref`

Summary: Defines generic indexing for objects.

A.27 Class `physiol_bundle`

A.27.1 Constructor `physiol_bundle/physiol_bundle`

Summary: The physiology dataset and the DB created from it bundled together.

Usage:

```
a_bundle = physiol_bundle(a_cell, props)
```

Description: This is a subclass of dataset_db_bundle, specialized for physiology datasets.

Parameters:

`a_cell`: A cell array that contains the following elements:

`a_dataset`: A cell-enclosed physiol_cip_traceset_fileset object.

`a_db`: The raw params_tests_db object created from the dataset.
It only needs to have the pAcip, pAbias, TracesetIndex, and ItemIndex columns.

`a_joined_db`: The one-treatment-per-line DB created from the raw DB.

`props`: A structure with any optional properties.

`controlDB`: Use this as the ontrol DB rather than computing.

Returns a structure object with the following fields:

dataset_db_bundle, joined_control_db: DB of control neurons (no pharmacological applications).

See also: `dataset_db_bundle` (p. 86), `tests_db` (p. 256), `params_tests_dataset` (p. 132)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/12/13

A.27.2 Method `physiol_bundle/bestMatchAllNeurons`

Summary: Finds the best match among given database for each physiology neuron.

Usage:

```
all_ranks_db = bestMatchAllNeurons(p_bundle, joined_db, props)
```

Description: Returns a database of best matching entries from joined_db for each entry in p_bundle.joined_control_db.

Parameters:

`p_bundle`: A physiol_bundle object.

`joined_db`: A database with neuron representations to rank against neurons.

`props`: A structure with any optional properties.
(passed to rankMatching)

Returns:

all_ranks_db: DB of best matching from joined_db. Each row corresponds to p_bundle.joined_control_db rows.

Example:

```
» all_ranks_db = ...
bestMatchAllNeurons(constrainedMeasuresPreset(pbundle2, 6), mbundle_maxcond.joined_db)
» plotXRows(all_ranks_db, 'Distance', 'maxcond DB distance per neuron', 'maxcond', ...
struct('LineStyle', '-', 'quiet', 1, 'PaperPosition', [0 0 4 3]))
```

See also: `tests_db/rankMatching` (p. 300), `tests_db/matchingRow` (p. 280)

Author: Cengiz Gunay <cgunay@emory.edu>, 2007/05/24

A.27.3 Method `physiol_bundle/constrainedMeasuresPreset`

Summary: Returns a physiol_bundle with constrained measures according to chosen preset.

Usage:

```
[a_pbundle test_names] = constrainedMeasuresPreset(a_pbundle, preset, props)
```

Parameters:

`a_pbundle`: A physiol_cip_traceset_fileset object.

`preset`: Choose preset measure list (default=1).

`props`: A structure with any optional properties.

Returns:

a_pbundle: One or more cip_trace object that holds the raw data.

See also: `loadItemProfile` (p. **??**), `physiol_cip_traceset/cip_trace` (p. 172)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/01/19

A.27.4 Method `physiol_bundle/ctFromRows`

Summary: Loads a cip_trace object from a raw data file in the a_pbundle.

Usage:

```
a_cip_trace = ctFromRows(a_pbundle, a_db|traceset_idx, cip_levels, props)
```

Parameters:

`a_pbundle`: A physiol_cip_traceset_fileset object.

`a_db`: A DB created by this fileset to read the traceset indices from.

`traceset_idx`: A column vector with traceset indices.

`cip_levels`: A column vector of CIP-levels to be loaded.

`props`: A structure with any optional properties.

- `traces`: column vector of trace indices to load.
- `showParamsList`: Cell array of params or treatments to include in the id field.

Returns:

a_cip_trace: One or more cip_trace object that holds the raw data.

See also: `loadItemProfile` (p. **??**), `physiol_cip_traceset/cip_trace` (p. 172)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/07/13

A.27.5 Method `physiol_bundle/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.27.6 Method `physiol_bundle/getNeuronLabel`

Summary: Constructs the neuron label from dataset.

Usage:

```
a_label = getNeuronLabel(a_bundle, traceset_index, props)
```

Parameters:

`a_bundle`: A physiol_cip_traceset_fileset object.

`traceset_index`: The traceset index of neuron.

`props`: A structure with any optional properties.

Returns:

a_label: A string label identifying selected neuron in bundle.

See also: `dataset_db_bundle` (p. 86)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/05/05

A.27.7 Method `physiol_bundle/getNeuronRowIndex`

Summary: Returns the neuron index from bundle.

Usage:

```
a_row_index = getNeuronRowIndex(a_bundle, traceset_index, props)
```

Parameters:

`a_bundle`: A physiol_bundle object.

`traceset_index`: The TracesetIndex number of neuron, or a DB row containing this.

`props`: A structure with any optional properties.

Returns:

a_row_index: A row index of neuron in a_bundle.joined_db.

See also: `dataset_db_bundle/getNeuronRowIndex` (p. 88)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/06/09

A.27.8 Method `physiol_bundle/matchingControlNeuron`

Summary: Creates a criterion database for matching the neuron at traceset_index.

Usage:

```
a_crit_bundle = matchingControlNeuron(a_bundle, neuron_id, props)
```

Description: Copies selected test values from row as the first row into the criterion db. Adds a second row for the STD of each column in the db.

Parameters:

`a_bundle`**:** A physiol_bundle object.

`neuron_id`**:** A NeuronId of the neuron to match.

`props`**:** A structure with any optional properties.

Returns:

a_crit_bundle: A tests_db with two rows for values and STDs.

Example:

```
Matches gpd0421c from cip_traces_all_axoclamp.txt:
» a_crit_bundle = matchingControlNeuron(pbundle, 33)
(see example in matchingRow)
```

See also: `rankMatching` (p. **??**), `tests_db` (p. 256), `tests2cols` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/12/21

A.27.9 Method `physiol_bundle/matchingRow`

Summary: Creates a criterion database for matching the neuron at traceset_index.

Usage:

```
a_crit_db = matchingRow(p_bundle, traceset_index, props)
```

Description: Copies selected test values from row as the first row into the criterion db. Adds a second row for the STD of each column in the db.

Parameters:

`p_bundle`**:** A physiol_bundle object.

`traceset_index`**:** A TracesetIndex of the neuron and treatments to match.

`props`**:** A structure with any optional properties.

Returns:

a_crit_db: A tests_db with two rows for values and STDs.

Example:

```
physiol_bundle has an overloaded matchingRow method that
takes the TracesetIndex as argument:
» a_crit_bundle = matchingRow(pbundle, 61)
» a_ranked_bundle = rankMatching(mbundle, a_crit_bundle);
» printTeXFile(comparisonReport(a_ranked_bundle), 'my_report.tex')
```

See also: rankMatching (p. **??**), tests_db/matchingRow (p. 280)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/12/21

A.27.10 Method physiol_bundle/mergeBundles

Summary: Merges two bundles together by adding w_bundle to p_bundle.

Usage:

```
p_bundle = mergeBundles(p_bundle, w_bundle, props)
```

Parameters:

p_bundle, w_bundle: physiol_bundle objects.

props: A structure with any optional properties.

Returns:

p_bundle: The merged p_bundle.

Example:

```
» p_bundle = mergeBundles(pbundle, another_bundle)
```

See also: rankMatching (p. **??**), tests_db/mergeBundles (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2008/05/18

A.27.11 Method physiol_bundle/plotfICurveStats

Summary: Generates a f-I curve mean-std plot of physiology DB.

Usage:

```
a_plot = plotfICurveStats(p_bundle, title_str, props)
```

Parameters:

p_bundle: A physiol_bundle object.

title_str: (Optional) String to append to plot title.

props: A structure with any optional properties.

`quiet`: if given, no title is produced
(passed to plot_superpose)

Returns:

a_plot: An f-I curve plot.

Example:

```
» plotFigure(plotfICurveStats(pbundle));
```

See also: `dataset_db_bundle/plotfICurve` (p. 90), `plot_abstract` (p. 180), `plot_superpose` (p. 196)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/06/16

A.27.12 Method `physiol_bundle/set`

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.28 Class `physiol_cip_traceset`

A.28.1 Constructor `physiol_cip_traceset/physiol_cip_traceset`

Summary: Dataset of cip traces from same PCDX file.

Usage:

```
obj = physiol_cip_traceset(trace_str, data_src, chaninfo, dt, dy, treatments, neuron_id,
props);
```

Description: This is a subclass of params_tests_dataset. Each trace varies in bias, pulse times and cip magnitude.

Parameters:

`trace_str`: Trace list in the format for loadtraces or just a Matlab vector.

`data_src`: Absolute path of PCDX data source.

`chaninfo`: 4-element array containing vchan, ichan, vgain, igain

`vchan, ichan`: Current and voltage channels.

`vgain, igain`: External gain factors for voltage channel and current channel (vgain does NOT include the 10X amplification from the Axoclamp, so vgain = 1 would mean no additional amplification beyond the 10X.)

`dt`: Time resolution [s].

`dy`: Y-axis resolution [V] or [A].

`treatments`: Structure containing the names and concentrations of compounds.

`neuron_id`: Neuron name.

`props`: A structure with any optional properties.

`nsHDF5`: For NeuroSAGE HDF5 files, processing is faster if the output of ns_open_file is given here. Must be defined to allow special NeuroSAGE processing.

`profile_method_name`: Use this cip_trace method to return a profile (Default: 'getProfileAllSpikes').

`cip_list`: Vector of cip levels to which the current trace will be matched. (All other props are passed to cip_trace objects)

Returns a structure object with the following fields:

params_tests_dataset, data_src, ichan, vchan, vgain, igain, treatments.

See also: `cip_traces` (p. **??**), `params_tests_dataset` (p. 132), `params_tests_db` (p. 138)

Author: Cengiz Gunay <cgunay@emory.edu> and Thomas Sangrey, 2005/01/17

A.28.2 Method `physiol_cip_traceset/CIPform`

Summary: Extracts current bias and pulse information from the current channel.

Usage:

```
[ciptype, on, off, finish, bias, pulse] = ns_CIPform(traceset,trace_index)
```

Parameters:

`traceset`: A physiol_cip_traceset object.

`trace_index`: Index of item in traceset

See also: `cip_traces` (p. **??**), `params_tests_dataset` (p. 132), `params_tests_db` (p. 138)

Author: Thomas Sangrey, 2005

A.28.3 Method `physiol_cip_traceset/cip_trace`

Summary: Loads a cip_trace object from a raw data file in the traceset.

Usage:

```
a_cip_trace = cip_trace(traceset, trace_index, props)
```

Parameters:

`traceset`: A physiol_cip_traceset object.

`trace_index`: Index of file in traceset.

`props`: A structure with any optional properties.

`showParamsList`: Cell array of params to add to id field.

`showName`: Show the name of the cell in the id field (default=1).

`TracesetIndex`: Indicates in the id field.

Returns:

a_cip_trace: A cip_trace object that holds the raw data.

See also: `itemResultsRow` (p. **??**), `params_tests_fileset` (p. 157), `paramNames` (p. **??**), `testNames` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/07/13

A.28.4 Method `physiol_cip_traceset/cip_trace_profile`

Summary: Loads a cip_trace_profile object from a raw data file in the traceset.

Usage:

```
a_profile = cip_trace_profile(traceset, trace_index)
```

Parameters:

`traceset`: A physiol_cip_traceset object.

`trace_index`: Index of file in traceset.

Returns:

a_profile: A profile object that implements the getResults method.

See also: `itemResultsRow` (p. **??**), `params_tests_fileset` (p. 157), `paramNames` (p. **??**), `testNames` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu> and Thomas Sangrey, 2005/01/18

A.28.5 Method `physiol_cip_traceset/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.28.6 Method `physiol_cip_traceset/getItemParams`

Summary: Get the parameter values of a dataset item.

Usage:

```
params_row = getItemParams(dataset, index, a_profile)
```

Parameters:

`dataset`**:** A params_tests_dataset.

`index`**:** Index of item in dataset.

`a_profile`**:** cip_trace_profile object

Returns:

params_row: Parameter values in the same order of paramNames

See also: `itemResultsRow` (p. **??**), `params_tests_dataset` (p. 132), `paramNames` (p. **??**), `testNames` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/12/06

A.28.7 Method `physiol_cip_traceset/itemResultsRow`

Summary: Processes a raw data file from the dataset and return its parameter and test values.

Usage:

```
[params_row, tests_row] = itemResultsRow(dataset, index)
```

Description: This method is designed to be reused from subclasses as long as the loadItemProfile method is properly overloaded. Adds an Index column to the DB to keep track of raw data items after shuffling.

Parameters:

`dataset`**:** A params_tests_dataset.

`index`**:** Index of file in dataset.

Returns:

params_row: Parameter values in the same order of paramNames tests_row: Test values in the same order with testNames

See also: `loadItemProfile` (p. **??**), `params_tests_dataset` (p. 132), `paramNames` (p. **??**), `testNames` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/10

A.28.8 Method `physiol_cip_traceset/loadItemProfile`

Summary: Loads a cip_trace_profile object from a raw data file in the traceset.

Usage:

```
a_profile = loadItemProfile(traceset, trace_index)
```

Parameters:

`traceset`**:** A physiol_cip_traceset object.

`trace_index`**:** Index of file in traceset.

Returns:

a_profile: A profile object that implements the getResults method.

See also: `itemResultsRow` (p. **??**), `params_tests_fileset` (p. 157), `paramNames` (p. **??**), `testNames` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.28.9 Method `physiol_cip_traceset/paramNames`

Summary: Returns the parameter names for this traceset.

Usage:

```
param_names = paramNames(traceset)
```

Description: Looks at the filename of the first file to find the parameter names.

Parameters:

`traceset`**:** A params_tests_dataset.

Returns:

param_names: Cell array with ordered parameter names.

See also: `params_tests_dataset` (p. 132), `paramNames` (p. **??**), `testNames` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/12/06

A.28.10 Method `physiol_cip_traceset/set`

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.28.11 Method `physiol_cip_traceset/setProp`

Summary: Generic method for setting optional object properties.

Usage:

```
obj = setProp(obj, prop1, val1, prop2, val2, ...)
```

Description: Modifies or adds property values. As many property name-value pairs can be specified.

Parameters:

`obj`: Any object that has a props field.

`attr`: Property name

`val`: Property value.

Returns:

obj: The new object with the updated properties.

See also:

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/22

A.29 Class `physiol_cip_traceset_fileset`

A.29.1 Constructor `physiol_cip_traceset_fileset/physiol_cip_traceset_fileset`

Summary: Physiological fileset of traceset objects (concatenated).

Usage:

```
obj = physiol_cip_traceset_fileset(traceset_items, dt, dy, props)
```

Description: This is a subclass of params_tests_dataset. It contains a set of physiol_cip_traceset items that are tied to physical data sources. Each traceset can load a set of traces for an experimental recording. Most flexible usage is obtained when the input traceset_items is given as a cell array of physiol_cip_traceset objects. These objects can each link to PCDX or NeuroSAGE HDF5 files independent of each other. A regular Matlab script can be used to create such a cell array. If a function is defined to return such an array, it can be passed as traceset_items. Alternatively, the cell array can be constructed from an ASCII file as described below, such as for deprecated PCDX data files.

Parameters:

`traceset_items`: It can be a function handle, cell array or filename string. Function should return a cell array of physiol_cip_traceset items. Alternatively a preconstructed cell array can be provided directly. If it is an ASCII filename, then it should contain the following tab-delimited items: 1. Neuron ID (name to associate with the neuron). If left blank, use the filename with the '.all' extension removed. 2. The absolute path of the data file 3. The trace numbers to load, space-delimited (e.g. 1-21 24 26 27) 4. Vchan: voltage channel number 5. Ichan: current channel number 6. Vgain: external gain on voltage channel IN ADDITION to the 10X that automatically comes from the Axoclamp 2B. 7. Igain: external gain on current channel. 8. Pairs of condition names and molar concentrations in any order e.g.: TTX 1e-8 apamin 2e-7 picrotoxin 1e-4

Returns a structure object with the following fields:

neuron_idx: A structure that points from neuron names to NeuronId numbers. params_tests_dataset

See also: `physiol_cip_traceset` (p. 170), `params_tests_dataset` (p. 132), `params_tests_db` (p. 138)

Author: Cengiz Gunay <cgunay@emory.edu> and Thomas Sangrey, 2005/01/17

A.29.2 Method `physiol_cip_traceset_fileset/cip_trace`

Summary: Loads a cip_trace object from a raw data file in the fileset.

Parameters:

`fileset`: A physiol_cip_traceset_fileset object.

`traceset_index`: Index of traceset item in this fileset (corresponds to row in cells_filename) to find the cell information.

`trace_index`: Index of item in the traceset.

`a_db`: A DB created by this fileset to read the traceset and item indices from.

`props`: A structure with any optional properties, passed to physiol_cip_traceset/cip_trace.

Returns:

a_cip_trace: One or more cip_trace object that holds the raw data.

See also: `loadItemProfile` (p. **??**), `physiol_cip_traceset/cip_trace` (p. 172)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/07/13

A.29.3 Method `physiol_cip_traceset_fileset/display`

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.29.4 Method `physiol_cip_traceset_fileset/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.29.5 Method `physiol_cip_traceset_fileset/loadItemProfile`

Summary: Loads a cip_trace_profile object from a raw data file in the fileset.

Usage:

```
a_profile = loadItemProfile(fileset, traceset_index, trace_index)
```

Parameters:

`fileset`**:** A physiol_cip_traceset object.

`traceset_index` **:** Index of traceset item in this fileset (corresponds to row in cells_filename) to use grab the cell information.

`trace_index`**:** Index of item in the traceset.

Returns:

a_profile: A profile object that implements the getResults method.

See also: `itemResultsRow` (p. **??**), `params_tests_fileset` (p. 157), `paramNames` (p. **??**), `testNames` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14 and Tom Sangrey

---

A.29.6 Method `physiol_cip_traceset_fileset/mergeFilesets`

Summary: Concatenates two physiol_cip_traceset_fileset objects.

Usage:

```
[a_fileset, traceset_offset, neuron_id_offset] = mergeFilesets(a_fileset, w_fileset)
```

Description: Concatenates the list contents, and combines the neuron_idx structures. The properties such as dt, dy and props are retained from first object.

Parameters:

`a_fileset, w_fileset`: Two physiol_cip_traceset_fileset objects without overlapping neuron_id items.

Returns:

a_fileset: The new object with combined contents.

See also: `physiol_cip_traceset_fileset` (p. 175)

Author: Cengiz Gunay <cgunay@emory.edu>, 2008/05/18

A.29.7 Method `physiol_cip_traceset_fileset/neuronNameFromId`

Summary: Returns string neuron names from a list of neuron ids.

Usage:

```
name_strs = neuronNameFromId(fileset, neuron_ids)
```

Parameters:

`fileset`: A physiol_cip_traceset_fileset object.

`neuron_ids`: One or more neuron ids in a vector.

Returns:

name_strs: Cell array of neuron names corresponding to the ids given.

See also: `physiol_cip_traceset_fileset` (p. 175)

Author: Cengiz Gunay <cgunay@emory.edu>, 2007/11/16

A.29.8 Method `physiol_cip_traceset_fileset/physiol_bundle`

Summary: Loads the database and then creates the physiol_bundle object.

Usage:

```
a_pbundle = physiol_bundle(fileset, props)
```

Description: Calls params_tests_db to get the db, and then calls tests_db/physiol_bundle to do transformations.

Parameters:

`fileset`: A physiol_cip_traceset_fileset object.

`props`: A structure with any optional properties. (Passed to tests_db/physiol_bundle)

Returns:

a_physiol_bundle: One or more physiol_bundle object that holds the raw data.

See also: tests_db/physiol_bundle (p. 285)

Author: Cengiz Gunay <cgunay@emory.edu>, 2007/12/21

A.29.9 Method physiol_cip_traceset_fileset/readDBItems

Summary: Reads all items to generate a params_tests_db object.

Usage:

```
[params, param_names, tests, test_names] = readDBItems(obj, items)
```

Description: This is a specific method to convert from physiol_cip_traceset_fileset to a params_tests_db, or a subclass. Outputs of this function can be directly fed to the constructor of a params_tests_db or a subclass.

Parameters:

`obj`: A physiol_cip_traceset_fileset

`items`: (Optional) List of item indices to use to create the db.

Returns:

params, param_names, tests, test_names: See params_tests_db.

See also: params_tests_db (p. 138), params_tests_fileset (p. 157), itemResultsRow (p. **??**)

A.29.10 Method physiol_cip_traceset_fileset/set

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.29.11 Method physiol_cip_traceset_fileset/setProp

Summary: Generic method for setting optional object properties.

Usage:

```
obj = setProp(obj, prop1, val1, prop2, val2, ...)
```

Description: Modifies or adds property values. As many property name-value pairs can be specified.

Parameters:

`obj`: Any object that has a props field.

`attr`: Property name

`val`: Property value.

Returns:

obj: The new object with the updated properties.

See also:

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/22

A.29.12 Method `physiol_cip_traceset_fileset/vertcat`

Summary: Concatenates multiple physiol_cip_traceset_fileset objects.

Usage:

```
obj = vertcat(obj, obj2)
```

Description: Concatenates the list contents, and combines the neuron_idx structures. The properties such as dt, dy and props are retained from first object.

Parameters:

`obj, obj2`: Two physiol_cip_traceset_fileset objects without overlapping neuron_id items.

Returns:

obj: The new object with combined contents.

See also: `physiol_cip_traceset_fileset` (p. 175)

Author: Cengiz Gunay <cgunay@emory.edu>, 2008/01/13

A.30 Class `plot_abstract`

A.30.1 Constructor `plot_abstract/plot_abstract`

Summary: A plot that can be directly visualized or included in subplots.

Usage:

```
obj = plot_abstract(data, axis_labels, title, legend, command, props)
```

Description: Base class that holds the necessary data to draw a plot. This data can then be used to generate different plots. Subclasses define specific plots with additional data. Subclasses should conform to the standard that the series of commands found in plotFigure should produce a valid figure.

Parameters:

`data`: A cell array of data arrays (x, y, z, etc.) that can be fed to plot commands.

`axis_labels`: Cell array of axis label strings.

`title`: Plot description string.

`legend`: Cell array of descriptions for each item plotted.

`command`: Plotting command to use (Optional, default='plot')

`props`: A structure with any optional properties.

- `axisLimits`: Sets axis limits of non-NaN values in vector.
- `tightLimits`: If 1, issues an "axis tight" command (default=0)
- `border`: Relative size of border spacing around axis, between 0 - 1. (default=0)
  If a scalar, equal border on all sides, give a four-element vector [left bottom right top] to define borders for each side.
- `colormap`: Figure colormap passed to the colormap function. If function handle, its output is passed instead.
- `grid`: Display dashed grid in background.
- `noXLabel`: No X-axis label.
- `noYLabel`: No Y-axis label.
- `noTitle`: No title.
- `rotateXLabel`: Rotates the X-axis label for smaller width.
- `rotateYLabel`: Rotates the Y-axis label for smaller width.
- `numXTicks`: Number of ticks on X-axis.
- `formatXTickLabels`: The sprintf format string for tick labels.
- `XTick, YTick`: Point locations for axis ticks.
- `XTickLabel, YTickLabel`: Axis tick labels.
- `ColorOrder`: Set the ColorOrder of the axis.
- `LineStyleOrder`: Set the LineStyleOrder of the axis.
- `legendLocation`: Passed to legend(..., 'location', legendLocation).
- `legendOrientation`: Passed to legend(..., 'orientation', legendLocation).
- `noLegends`: If exists, no legends are displayed.
- `axisProps`: Passed to set properties of the axis drawn.
- `plotProps`: Passed to set properties of the plot drawn.
- `figureProps`: Passed to set properties of the figure drawn.
- `PaperPosition`: Sets the figure property for printing at this size.

`resizeControl`: If 0, drawing after resize is disabled and prints at screen size, if 1 (default), redraws figure after each resize event and prints at PaperPosition size.

`fixedSize`: Vector of width and height in inches passed to PaperPosition property. Implies resizeControl=0.

Returns a structure object with the following fields:

data, axis_labels, title, legend, command, props

See also: `plot_abstract/plot` (p. 184), `plot_abstract/plotFigure` (p. 185)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/22

A.30.2 Method `plot_abstract/axis`

Summary: Returns the estimated axis ranges of this plot according to its data.

Usage:

```
ranges = axis(a_plot)
```

Parameters:

`a_plot`: A plot_abstract object, or a subclass object.

Returns:

ranges: The ranges as a vector in the same way 'axis' would return.

See also: `plot_abstract` (p. 180), `plot_abstract/plot` (p. 184)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/13

A.30.3 Method `plot_abstract/decorate`

Summary: Places decorations (titles, labels, ticks, etc.) on the plot.

Usage:

```
handles = decorate(a_plot, plot_handles)
```

Parameters:

`a_plot`: A plot_abstract object, or a subclass object.

`plot_handles`: Handles of plots already drawn (structure returned by plot_abstract/plot).

Returns:

handles: Structure with handles of all graphical objects drawn.

See also: plot_abstract (p. 180), plot_abstract/plot (p. 184)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/22

A.30.4 Method plot_abstract/display

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.30.5 Method plot_abstract/get

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.30.6 Method plot_abstract/matrixPlots

Summary: Organize multiple plots in a matrix formation.

Usage:

```
a_plot = matrixPlots(plots, axis_labels, title_str, props)
```

Parameters:

`plots`: Array of plot_abstract or subclass objects.

`axis_labels`: Cell array of axis label strings (optional, taken from plots).

`title_str`: Plot description string (optional, taken from plots).

`props`: A structure with any optional properties passed to the Y stack_plot.

`titlesPos, yLabelsPos, yTicksPos`: if specified, passed to the X stack_plots.

`rotateYLabel`: if specified, passed to the X stack_plots.

`axisLimits`: if specified, passed to the X stack_plots.

`goldratio`: try to make the figure in this aspect ratio.

`width, height`: if specified, make the figure have this many plots in corresponding dimension.

Returns:

a_plot: A plot_abstract object.

See also: plot_abstract (p. 180), plot_abstract/plot (p. 184), plot_abstract/plotFigure (p. 185)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/12/07

A.30.7 Method `plot_abstract/openAxis`

Summary: Calculates the extents for the axis of this plot and opens it.

Usage:

```
[axis_handle, layout_axis] = openAxis(a_plot, layout_axis)
```

Parameters:

`a_plot`**:** A plot_abstract object, or a subclass object.

`layout_axis`**:** The axis position to layout this plot (Optional).
If NaN, doesn't open a new axis.

Returns:

handles: Handles of graphical objects drawn.

See also: `plot_abstract` (p. 180)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/22

A.30.8 Method `plot_abstract/plot`

Summary: Draws this plot in the current axis.

Usage:

```
handles = plot(a_plot, layout_axis)
```

Parameters:

`a_plot`**:** A plot_abstract object, or a subclass object.

`layout_axis`**:** The axis position to layout this plot (Optional).
If NaN, doesn't open a new axis.

Returns:

handles: Handles of graphical objects drawn.

See also: `plot_abstract` (p. 180)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/22

A.30.9 Method `plot_abstract/plotFigure`

Summary: Draws this plot alone in a new figure window.

Usage:

```
[handle, plot_handles] = plotFigure(a_plot, title_str, props)
```

Parameters:

`a_plot`: A plot_abstract object, or a subclass object.

`title_str`: (Optional) String to append to plot title.

`props`: A structure with any optional properties.

`figureHandle`: Use this figure instead of opening a new one.

`delayOpen`: Wait this many seconds for figure to materialize (0.5s is good workaround for Compiz-fusion bug)

Returns:

handle: Handle of new figure. plot_handles: Structure with all plotted data and decorations.

See also: `plot_abstract` (p. 180), `plot_abstract/plot` (p. 184), `plot_abstract/decorate` (p. 182)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/22

A.30.10 Method `plot_abstract/set`

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.30.11 Method `plot_abstract/setProp`

Summary: Generic method for setting optional object properties.

Usage:

```
obj = setProp(obj, prop1, val1, prop2, val2, ...)
```

Description: Modifies or adds property values. As many property name-value pairs can be specified.

Parameters:

`obj`: Any object that has a props field.

`attr`: Property name

`val`: Property value.

Returns:

obj: The new object with the updated properties.

See also:

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/22

A.30.12 Method `plot_abstract/subsasgn`

Summary: Defines generic index-based assignment for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/02/06

A.30.13 Method `plot_abstract/subsref`

Summary: Defines generic indexing for objects.

A.30.14 Method `plot_abstract/superposePlots`

Summary: Superpose multiple plots with common command onto a single axis.

Usage:

```
a_plot = superposePlots(plots, axis_labels, title_str, command, props)
```

Description: The plot decoration will be taken from the last plot in the list, with the exception of legend labels.

Parameters:

`plots`: Array of plot_abstract or subclass objects.

`axis_labels`: Cell array of axis label strings (optional, taken from plots).

`title_str`: Plot description string (optional, taken from plots).

`command`: Plotting command to use (optional, taken from plots)

`props`: A structure with any optional properties.

`noLegends`: If exists, no legends are created.

Returns:

a_plot: A plot_abstract object.

See also: `plot_abstract` (p. 180), `plot_abstract/plot` (p. 184), `plot_abstract/plotFigure` (p. 185)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/23

A.31 Class `plot_bars`

A.31.1 Constructor `plot_bars/plot_bars`

Summary: Bar plot with error lines in individual axes for each variable.

Usage:

```
a_plot = plot_bars(mid_vals, lo_vals, hi_vals, n_vals, x_labels, y_labels, ...  title_str,
axis_limits, props)
```

Description: Additional rows of data will result in grouped bars in each axis. If all data is given as a column vector, then they will appear in a single axis. plot_bars is a subclass of plot_stack. The plot_abstract/plot command can be used to plot this data. Rows of *_vals will create grouped bars, columns will create new axes.

Parameters:

`mid_vals`: Middle points of error bars.

`lo_vals`: Low points of error bars as difference from mid_vals.

`hi_vals`: High points of error bars as difference from mid_vals.

`n_vals`: Number of samples used for the statistic (Optional).

`x_labels, y_labels`: Axis labels for each bar group. Must match with data columns.

`title_str`: Plot description.

`axis_limits`: If given, all plots contained will have these axis limits.

`props`: A structure with any optional properties.

- `dispBarsLines`: Choose between using 'bars' or 'lines' to connect the errorbars.
- `dispErrorbars`: If 1, display errorbars for lo_vals and hi_vals deviation from mid_vals
  (default=1).
- `dispInnerBars`: If 1, an inner bar extends from the base to hi_vals (default=0). Mutually exclusive with dispInnerBars. It will make the larger bars blank.
- `dispNvals`: If 1, display n_vals on top of each bar (default=1).
- `groupValues`: List of within-group labels passed to XTickLabels, instead of just a sequence of numbers.
- `groupValuesLoc`: If 1, use specified group values as the location of bars or errorbars. By default locations are set arbitrarily as 1:n.
- `truncateDecDigits`: Truncate labels to this many decimal digits.
- `barAxisProps`: props passed to plot_abstract objects with bar commands
- `barWidth`: Controls spacing between bars (see width argument for the bar command; default=0.8).

Returns a structure object with the following fields:

plot_abstract

See also: plot_abstract (p. 180), plot_abstract/plot (p. 184)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/07

A.31.2 Method plot_bars/set

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.32 Class plot_errorbar

A.32.1 Constructor plot_errorbar/plot_errorbar

Summary: Generic errorbar plot.

Usage:

```
a_plot = plot_errorbar(x_vals, mid_vals, lo_vals, hi_vals, line_spec, axis_labels, title,
legend, props)
```

Description: Subclass of plot_abstract. The plot_abstract/plot command can be used to plot this data. Needed to create this as a separate class to have the axis ranges method to measure the errorbars.

Parameters:

x_vals: X coordinates of errorbars.

mid_vals: Middle points of error bars.

lo_vals: Low points of error bars.

hi_vals: High points of error bars.

line_spec: Plot line spec to be passed to errorbar

axis_labels: Cell array for X, Y axis labels.

title: Plot description.

legend: For multiple errorbar plots (matrix form), description of each plot.

props: A structure with any optional properties to be passed to plot_abstract.

Returns a structure object with the following fields:

plot_abstract.

See also: plot_abstract (p. 180), plot_abstract/plot (p. 184)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/07

A.32.2 Method `plot_errorbar/axis`

Summary: Returns the estimated axis ranges of this plot according to its data.

Usage:

```
ranges = axis(a_plot)
```

Parameters:

`a_plot`**:** A plot_abstract object, or a subclass object.

Returns:

ranges: The ranges as a vector in the same way 'axis' would return.

See also: `plot_abstract` (p. 180), `plot_abstract/plot` (p. 184)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/13

A.32.3 Method `plot_errorbar/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.33 Class `plot_errorbars`

A.33.1 Constructor `plot_errorbars/plot_errorbars`

Summary: Plots distributions of variables with errorbars in separate axes.

Usage:

```
a_plot = plot_errorbars(labels, mid_vals, lo_vals, hi_vals, labels, title, axis_limits,
props)
```

Description: Subclass of plot_stack. The plot_abstract/plot command can be used to plot this data. Each of mid_vals, lo_vals, and hi_vals plot its rows in the same axis and columns in different axes.

Parameters:

`labels`**:** Labels of parameters to appear at bottom of each errorbar.

`mid_vals`**:** Middle points of error bars.

`lo_vals`**:** Low points of error bars.

`hi_vals`**:** High points of error bars.

`title`**:** Plot description.

`axis_limits`: If given, all plots contained will have these axis limits.

`props`: A structure with any optional properties.

Returns a structure object with the following fields:

plot_abstract, labels.

See also: `plot_abstract` (p. 180), `plot_abstract/plot` (p. 184)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/07

A.34 Class `plot_image`

A.34.1 Constructor `plot_image/plot_image`

Summary: Generic image plot.

Usage:

```
a_plot = plot_image(image_data, axis_labels, colorbar_label, title_str, props)
```

Description: Subclass of plot_abstract. The plot_abstract/plot command can be used to plot this data. Needed to create this as a separate class to have the axis ranges method implemented and take advantage of plot_abstract props.

Parameters:

`image_data`: 2D matrix with image data.

`axis_labels`: Cell array for X, Y axis labels.

`colorbar_label`: String to appear next to colorbar.

`title_str`: Plot description.

`props`: A structure with any optional properties.

- `colorbar`: If defined, show colorbar next to plot.
- `numGrads`: Number of poles in the colormap gradient. If 1 (default), it will be a monocolor gradient (e.g., gray-level); if 2, it will be a dual-color gradient (e.g., blue to red) with a black crossing point. This point is determined by the minValue (below). Default numGrads=2, if negative values exist in image_data after scaling.
- `minValue,maxValue`: Use these value to scale the data by (image_data - minValue) / (maxValue - minValue). Otherwise, its min and max is used.
- `colormap`: Colormap vector, function name or handle to colormap (e.g., 'jet').
- `numColors`: Number of colors in colormap.
- `NaNcolor`: Color to be used for NaN entries (default: [1 1 1])

(Rest passed to plotImage.)

Returns a structure object with the following fields:

plot_abstract.

Example:

```
» plotFigure(plot_image(rand(5), 'r1', 'r2', 'rand', 'random matrix'))
```

See also: plot_abstract (p. 180), plot_abstract/plot (p. 184)

Author: Cengiz Gunay <cgunay@emory.edu>, 2008/04/15

A.34.2 Method plot_image/axis

Summary: Returns the estimated axis ranges of this plot according to its data.

Usage:

```
ranges = axis(a_plot)
```

Parameters:

a_plot: A plot_abstract, or subclass, object.

Returns:

ranges: The ranges as a vector in the same way 'axis' would return.

See also: plot_abstract (p. 180), plot_abstract/plot (p. 184), axis (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/13

A.34.3 Method plot_image/get

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.35 Class plot_inset

A.35.1 Constructor plot_inset/plot_inset

Summary: Superpose multiple plots with individual axis at arbitrary locations.

Usage:

```
a_plot = plot_inset(plots, axis_locations, title_str, props)
```

Description: Subclass of plot_abstract. Contains other plot_abstract objects or subclasses thereof to be layout in arbitaray format. Allows overlapping and therefore good for insets and special plots.

Parameters:

`plots`: Cell array of plot_abstract or subclass objects.

`axis_locations`: Matrix of four-element vectors for each given plot.

`title_str`: Title to go on top of the stack

`props`: A structure with any optional properties.

`positioning`: axis_locations interpreted as 'absolute' values or 'relative' to the 1st plot (default='absolute'). Relative positioning doesn't work well.

Returns a structure object with the following fields:

plot_abstract, plots, axis_locations.

See also: `plot_abstract` (p. 180), `plot_abstract/plotFigure` (p. 185)

Author: Cengiz Gunay <cgunay@emory.edu>, 2007/06/05

A.35.2 Method `plot_inset/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.35.3 Method `plot_inset/plot`

Summary: Superposes contained plots in their own axes.

Usage:

```
handles = plot(a_plot, layout_axis)
```

Parameters:

`a_plot`: A plot_superpose object.

`layout_axis`: The axis position to layout this plot (Optional).

Returns:

handles: Handles of graphical objects drawn.

See also: `plot_abstract` (p. 180)

Author: Cengiz Gunay <cgunay@emory.edu>, 2007/06/08

A.35.4 Method `plot_inset/set`

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.36 Class `plot_simple`

A.36.1 Constructor `plot_simple/plot_simple`

Summary: Abstract description of a single plot.

Usage:

```
a_plot = plot_simple(data_x, data_y, title, label_x, label_y, legend, command, props)
```

Description: Subclass of plot_abstract. The plot_abstract/plot command can be used to plot this data.

Parameters:

`data_x`: X-axis values for the plot.

`data_y`: Y-axis values for the plot.

`title`: Plot description.

`label_x`: X-axis label string.

`label_y`: Y-axis label string.

`legend`: Short description of data points.

`command`: Plotting command to use (Optional, default='plot')

`props`: A structure with any optional properties.

Returns a structure object with the following fields:

plot_abstract.

See also: `plot_abstract` (p. 180), `plot_abstract/plot` (p. 184)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/22

A.36.2 Method `plot_simple/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.36.3 Method `plot_simple/set`

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.37 Class `plot_stack`

A.37.1 Constructor `plot_stack/plot_stack`

Summary: A horizontal or vertical stack of plots.

Usage:

```
a_plot = plot_stack(plots, axis_limits, orientation, title_str, props)
```

Description: Subclass of plot_abstract. Contains other plot_abstract objects or subclasses thereof to be layout in stack format.

Parameters:

`plots`: Cell array of plot_abstract or subclass objects.

`axis_limits`: If given, all plots contained will have these axis limits. In this vector, NaNs are untouched, Infs are replaced by minimal and maximal ranges of the stacked plots.

`orientation`: Stack orientation 'x' for horizontal, 'y' for vertical, etc.

`title_str`: Title to go on top of the stack

`props`: A structure with any optional properties.

- `yLabelsPos`: 'left' means only put y-axis label to leftmost plot.
- `yTicksPos`: 'left' means only put y-axis ticks to leftmost plot.
- `xLabelsPos`: 'bottom' means only put x-axis label to lowest plot.
- `xTicksPos`: 'bottom' means only put x-axis ticks to lowest plot.
- `titlesPos`: 'top' means only put title to top plot.
- `relaxedLimits`: Add 10
- `relativeSizes`: An array specifying relative size of each plot with one value.
  (Example: relativeSizes=[1 2] makes second plot twice wider than first.)

Returns a structure object with the following fields:

plot_abstract, plots, axis_limits, orient.

See also: `plot_abstract` (p. 180), `plot_abstract/plotFigure` (p. 185)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/04

A.37.2 Method `plot_stack/decorate`

Summary: No additional decorations for stacked plots.

Usage:

```
a_histogram_db = decorate(a_plot)
```

Parameters:

`a_plot`**:** A plot_abstract object, or a subclass object.

Returns:

handles: Handles of graphical objects drawn.

See also: `plot_abstract` (p. 180), `plot_abstract/plot` (p. 184)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/04

A.37.3 Method `plot_stack/display`

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.37.4 Method `plot_stack/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.37.5 Method `plot_stack/plot`

Summary: Draws this plot in the current axis or at the position in layout_axis.

Usage:

```
handles = plot(a_plot, layout_axis)
```

Parameters:

`a_plot`**:** A plot_abstract object, or a subclass object.

`layout_axis`**:** The axis position to layout this plot (Optional).

Returns:

handles: Handles of graphical objects drawn.

See also: `plot_stack` (p. 194), `plot_abstract` (p. 180)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/04

A.37.6 Method `plot_stack/set`

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.37.7 Method `plot_stack/superposePlots`

Summary: Superpose multiple plot_stack objects that contain exact same contents.

Usage:

```
a_plot = superposePlots(plots, axis_labels, title_str, command, props)
```

Description: The plot decoration will be taken from the last plot in the list, with the exception of legend labels.

Parameters:

`plots`: Array of plot_stack objects.

`axis_labels`: Cell array of axis label strings (optional, taken from plots).

`title_str`: Plot description string (optional, taken from plots).

`command`: Plotting command to use (optional, taken from plots)

`props`: A structure with any optional properties.

`noLegends`: If exists, no legends are created.

Returns:

a_plot: A plot_stack object.

See also: `plot_abstract` (p. 180), `plot_abstract/plot` (p. 184), `plot_abstract/plotFigure` (p. 185)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/06/14

A.38 Class `plot_superpose`

A.38.1 Constructor `plot_superpose/plot_superpose`

Summary: Multiple plot_abstract objects superposed on the same axis.

Usage:

```
obj = plot_superpose(plots, axis_labels, title_str, props)
```

Description: Subclass of plot_abstract. Contains multiple plot_abstract objects to be plotted on the same axis. This is different than the plot_abstract/superpose, where only using the same plot command is allowed. Here, each plot_abstract can have its own special plotting command. Subclasses of plot_abstract is also allowed here. The decorations comes from this object and not children plots. This behavior is different than plot_stack, where each plot has its own decorations. If you want each plot to have its own axis (e.g. an inset, or plot with multiple axis labels) then you should use plot_inset. For convenience, the props of first plot is inherited by plot_superpose objects. For instance, in the example below, the fixedSize property is used by plotFigure because it's in the first subplot.

Parameters:

`plots`**:** Cell array of plot_abstract or subclass objects.

`axis_labels`**:** Cell array of axis label strings.

`title_str`**:** Plot description string.

`props`**:** A structure with any optional properties (passed to plot_abstract).

`noCombine`**:** Do not auto-combine plots with same properties (default=0). This is especially important for plot_bars.

Returns a structure object with the following fields:

plot_abstract, plots

Example:

```
» a_p = plot_abstract([0 0], [1 2], , '', , 'plot',
struct('fixedSize', [3 2]));
» a_p2 = plot_abstract(...)
» a_p3 = plot_abstract(...)
» plotFigure(plot_superpose(a_p1, a_p2, a_p3))
```

See also: `plot_abstract/superpose` (p. **??**), `plot_superpose/plot` (p. 199)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/22

A.38.2 Method `plot_superpose/axis`

Summary: Returns the maximal axis ranges according to superposed subplots.

Usage:

```
ranges = axis(a_plot)
```

Parameters:

`a_plot`**:** A plot_abstract object, or a subclass object.

Returns:

ranges: The ranges as a vector in the same way 'axis' would return.

See also: `plot_abstract` (p. 180), `plot_abstract/plot` (p. 184)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/05/22

A.38.3 Method `plot_superpose/decorate`

Summary: Places decorations using the first plot of the superposed plots.

Usage:

```
handles = decorate(a_plot, plot_handles)
```

Parameters:

`a_plot`: A plot_abstract object, or a subclass object.

`plot_handles`: Handles of plots already drawn (structure returned by plot_superpose/plot).

Returns:

handles: Handles of graphical objects drawn.

See also: `plot_abstract` (p. 180), `plot_abstract/plot` (p. 184)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/04/11

A.38.4 Method `plot_superpose/display`

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.38.5 Method `plot_superpose/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.38.6 Method `plot_superpose/plot`

Summary: Draws this plot in the current axis.

Usage:

```
handles = plot(a_plot, layout_axis)
```

Parameters:

`a_plot`: A plot_superpose object.
`layout_axis`: The axis position to layout this plot (Optional).

Returns:

handles: Handles of graphical objects drawn.

See also: `plot_abstract` (p. 180)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/04/08

A.38.7 Method `plot_superpose/set`

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.38.8 Method `plot_superpose/superposePlots`

Summary: Superpose multiple plot_superpose objects by merging them into one.

Usage:

```
a_plot = superposePlots(plots, axis_labels, title_str, command, props)
```

Parameters:

`plots`: Array of plot_superpose objects.
`axis_labels`: Cell array of axis label strings (optional, taken from plots).
`title_str`: Plot description string (optional, taken from plots).
`command`: Plotting command to use (optional, taken from plots)
`props`: A structure with any optional properties.
    `noLegends`: If exists, no legends are created.

Returns:

a_plot: A plot_superpose object.

See also: `plot_abstract/superposePlots` (p. 186), `plot_stack/superposePlots` (p. 196)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/06/14

A.39 Class `ranked_db`

A.39.1 Constructor `ranked_db/ranked_db`

Summary: A database of distance values generated by ranking rows of orig_db with the criterion in crit_db.

Usage:

```
a_ranked_db = ranked_db(data, col_names, orig_db, crit_db, id, props)
```

Description: This is a subclass of tests_db. It should contain a Distance column. A more general ranked db class may be needed later. Use the rankMatching method to get an instance of this class.

Parameters:

`data`: Database contents.

`col_names`: The column names.

`orig_db`: DB whose rows are ranked.

`crit_db`: The criterion DB used for generating the ranking scores.

`id`: An identifying string.

`props`: A structure with any optional properties.

`tolerateNaNs`: If 0, rows with any NaN values are skipped (default=1).

Returns a structure object with the following fields:

tests_db, orig_db, crit_db, props.

See also: `tests_db` (p. 256), `tests_db/rankMatching` (p. 300), `tests_db/matchingRow` (p. 280)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/12/21

A.39.2 Method `ranked_db/blockedDistances`

Summary: Creates a db of distances to blocked versions of top ranks.

Usage:

```
[a_db, ranked_dbs] = blockedDistances(a_ranked_db, rows, blocked_db, blocked_param_indices,
block_levels, crit_db)
```

Parameters:

`a_ranked_db`: A ranked_db object.

`rows`: Use the given row rankings.

`blocked_db`: db with blocked versions of original ranks.

`blocked_param_indices`: Indices of parameters to be blocked.

`block_levels`: Number of parameter levels for blocking.

`crit_db`: Calculate distance from this criterion.

Returns:

a_db: A tests_db object with the matrix of distances. ranked_dbs: A cell array of ranked_dbs for each row.

Example:

```
» dist_matx_db = blockedDistances(rankMatching(super_db, matchingRow(rsuper_phys_db, 20)),
1:5, super_blocker_db, [1 2], 10, matchingRow(rsuper_phys_db, 21))
```

See also: makeModifiedParamDB (p. **??**), getParamRowIndices (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/01/14

A.39.3 Method `ranked_db/displayRows`

Summary: Displays rows of rankings together with errors associated with each measure.

Usage:

```
s = displayRows(db, rows, props)
```

Parameters:

`db`: A tests_db object.

`rows`: Indices of rows in db.

`props`: Struct with optional properties.

`originalCols`: Column pattern to limit result of joinOriginal.

Returns:

s: A structure of column name and value pairs.

See also: tests_db (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/15

A.39.4 Method `ranked_db/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.39.5 Method `ranked_db/getDistMatrix`

Summary: Create a matrix of total errors from the ranked DB.

Usage:

```
distmatx = getDistMatrix(db, rows, col_size, props)
```

Description: The col_size parameter is used to find the number of rows that make up the x-dimension of the matrix.

Parameters:

`db`: A tests_db object.

`rows`: Indices of rows in db after joining (and sorting).

`col_size`: Number of rows to take from DB to form the columns of matrix plot.

`props`: A structure with any optional properties.

`sortBy`: If specified, db is sorted after being joined with original using this column.

Returns:

a_plot: A plot_abstract object.

See also: `tests_db` (p. 256), `plot_abstract` (p. 180)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/12/12

A.39.6 Method `ranked_db/joinOriginal`

Summary: Joins the distance values to the original db rows with matching row indices.

Usage:

```
a_db = joinOriginal(a_ranked_db, rows, props)
```

Description: Takes the parameter columns from orig_db and all tests from crit_db. Therefore z-score values in a_ranked_db are replaced with original metric magnitudes. Alternatively, the scores can be preserved in the output by specifying the keepScores prop.

Parameters:

`a_ranked_db`: A ranked_db object.

`rows`: Join only the given rows.

`props`: A structure with any optional properties.

`includeIndices`: Also joins ItemIndex columns from the original db.

`origCols`: Also join these columns from the original DB.

`rankedCols`: Also join these columns from a_ranked_db.

`keepScores`: Keep tests (metrics) from a_ranked_db instead of original db.

Returns:

a_db: A params_tests_db object (same type as a_ranked_db.orig_db) containing the desired rows in ascending order of distance.

See also: `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/12/21

A.39.7 Method `ranked_db/plotCompareDistMatx`

Summary: Compare differences and correlations of distance matrices from two ranked DBs.

Usage:

```
a_plot = plotCompareDistMatx(db, rows, col_size, col_name, num_col_labels, row_name,
num_row_labels, title_str, props)
```

Description: Produces three plots: (1) distance difference matrix, (2) 2D cross-correlogram, and (3) repeated 1D cross-correlogram for each row.

Parameters:

`db, w_db`: The ranked_db objects to be compared.

`rows`: Indices of rows in db after joining (and sorting) for both DBs.

`col_size`: Number of rows to take from DB to form the columns of matrix plot.

`col_name, row_name`: DB column to use fot the figure column and row, respectively.

`num_col_labels, num_row_labels`: Number of labels to put on each axis.

`title_str`: If non-empty, replaces generic title with db name.

`props`: A structure with any optional properties.

`sortBy`: If specified, db is sorted after being joined with original using this column.

`colorbar`: Put a colorbar on the figure.
(also passed to plot_abstract)

Returns:

a_plot: A plot_abstract object.

See also: tests_db (p. 256), plot_abstract (p. 180)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/12/12

A.39.8 Method ranked_db/plotDistMatrix

Summary: Create a color-coded matrix plot of with total errors from the ranked DB.

Usage:

```
a_plot = plotDistMatrix(db, rows, col_size, col_name, num_col_labels, row_name,
num_row_labels, title_str, props)
```

Description: The col_size parameter is used to find the number of rows that make up the x-dimension of the color matrix plot.

Parameters:

db: A ranked_db object.

rows: Indices of rows in db after joining (and sorting).

col_size: Number of rows to take from DB to form the columns of matrix plot.

col_name, row_name: DB column to use for the figure column and row, respectively.

num_col_labels, num_row_labels: Number of labels to put on each axis.

title_str: If non-empty, replaces generic title with db name.

props: A structure with any optional properties.

sortBy: If specified, db is sorted after being joined with original using this column.

colorbar: Put a colorbar on the figure.
(also passed to plot_abstract)

Returns:

a_plot: A plot_abstract object.

Example:

```
» plotFigure(plotDistMatrix(scored_blocked_sk_gps0503b_control_db, ':', 10, 'SK', 10,
'trial', 10, 'gps0503b (control), preset 6 - top 50 matches', struct('sortBy', 'trial',
'colorbar', 1, 'PaperPosition', [0 0 5 3])));
```

See also: ranked_db (p. 200), plot_abstract (p. 180), getDistMatrix (p. **??**), plotCompareDistMatx (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/12/12

A.39.9 Method `ranked_db/plotRowErrors`

Summary: Create plot of rankings with errors associated with each measure color-coded.

Usage:

```
a_plot = plotRowErrors(a_ranked_db, rows, props)
```

Parameters:

`a_ranked_db`: A ranked_db object.

`rows`: Indices of rows in a_ranked_db.

`title_str`: (Optional) String to append to plot title.

`props`: A structure with any optional properties.

`sortMeasures`: If specified, measure order is determined with increasing overall distance.

`RowName`: Label to show on X-axis (default='Ranks')

`rowSteps`: Steps to jump in labeling rows on the x-axis.

`superposeDistances`: Superpose a white-colored distance line plot.

`colorbar`: Put a colorbar on the figure.
(rest passed to plot_abstract)

Returns:

a_plot: A plot_abstract object.

See also: `ranked_db` (p. 200), `tests_db/rankMatching` (p. 300), `plot_abstract` (p. 180), `plotImage` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/12/12

A.39.10 Method `ranked_db/renameColumns`

Summary: Rename an existing column or columns.

Usage:

```
a_db = renameColumns(a_db, test_names, new_names)
```

Description: This method is an overloaded method for ranked_db that keeps consistent the column names of the ranked, criterion and original DBs. The other DBs are not renamed for the Distance and RowIndex columns.

Parameters:

`a_db`: A ranked_db object.

`test_names`: A cell array of existing test names.

`new_names`: New names to replace existing ones.

Returns:

a_db: The ranked_db object that includes the new columns.

See also: `tests_db/renameColumns` (p. 302)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/06/07

A.39.11 Method `ranked_db/set`

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.39.12 Method `ranked_db/subsref`

Summary: Defines generic indexing for objects.

A.40 Class `results_profile`

A.40.1 Constructor `results_profile/results_profile`

Summary: Creates and collects result profiles for data objects.

Usage:

```
obj = results_profile(results, id, props)
```

Description: This is the base class for all profile classes.

Parameters:

`results`: A structure containing test results.

`id`: Identification string.

`props`: A structure with any optional properties.

Returns a structure object with the following fields:

results, id, props.

See also: `trace_profile` (p. 338), `cip_trace_profile` (p. 72)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.40.2 Method `results_profile/display`

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.40.3 Method `results_profile/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.40.4 Method `results_profile/getResults`

Summary: Return the results profile structure.

Usage:

```
results = getResults(p)
```

Parameters:

`p`: A result_profile object.

Returns:

results: A structure associating test names to values.

See also: `results_profile` (p. 206)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.40.5 Method `results_profile/plot`

Summary: Generic method to plot a tests_db or a subclass. Requires a plot_abstract method to be defined for this object.

Usage:

```
h = plot(a_tests_db, title_str, props)
```

Parameters:

`a_tests_db`: A histogram_db object.

`title_str`: (Optional) String to append to plot title.

`props`: Optional properties passed to plot_abstract.

Returns:

h: The figure handle created.

See also: `plot_abstract` (p. 180), `plotFigure` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/06

A.40.6 Method `results_profile/subsref`

Summary: Defines generic indexing for objects.

A.41 Class `script_array`

A.41.1 Constructor `script_array/script_array`

Summary: Generic class that provides the scripts for a repetitive array job.

Usage:

```
obj = script_array(num_runs, id, props)
```

Description: This is the base class for all script_array classes. Runs the runJob method as num_runs many times. Run initiated by calling runFirst, but the final result will be returned by runLast.

Parameters:

`num_runs`: The number of times the runJob script should be evoked.

`id`: Identification string.

`props`: A structure with any optional properties.

`runJobFunc`: A function name or handle to be used instead of default runJob.

`parallel`: If 1, run jobs in parallel using parfor.

Returns a structure object with the following fields:

num_runs, id, props.

Example:

```
» func1 = inline('x^2')
» runFirst(script_array(10, 'squares numbers up to 10'), struct('runJobFunc', func1))
ans = [ 1] [ 4] [ 9] [ 16] [ 25] [ 36] [ 49] [ 64] [ 81] [100]
```

See also: `runFirst` (p. **??**), `runLast` (p. **??**), `runJob` (p. **??**), `parfor` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/02/01

A.41.2 Method `script_array/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.41.3 Method `script_array/runFirst`

Summary: Method to be called at beginning of script_array jobs.

Usage:

```
job_results = runFirst(a_script_array)
```

Description: This method initiates the script_array jobs. It loops and calls runJob and finally calls runLast.

Parameters:

`a_script_array`**:** A script_array object.

Returns:

job_results: A cell array of results collected from each item of the vector jobs.

Example:

```
» results = runFirst(script_array(10, 'this one does nothing for 10 times'));
```

See also: runLast (p. **??**), runJob (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/02/01

A.41.4 Method `script_array/runJob`

Summary: Method to be called for each of the script_array jobs.

Usage:

```
job_result = runJob(a_script_array, vector_index)
```

Description: This method is provided as a placeholder and does nothing. If the run_job_func property is defined, it will call that function.

Parameters:

`a_script_array`**:** A script_array object.

`vector_index`**:** The index within the vector job.

Returns:

job_result: Any output produced by the job.

Example:

```
See real example in script_array.  Call the 5th job:
» runJob(script_array(10, 'this one does nothing for 10 times'), 5);
```

See also: runLast (p. **??**), runFirst (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/02/01

A.41.5 Method `script_array/runLast`

Summary: Method to be called last after the script_array jobs.

Usage:

```
job_results = runLast(a_script_array, job_results)
```

Description: This method is provided as a placeholder and does nothing. It can filter-out the results returned from the jobs run. Normally it is invoked internally by the runFirst method, after running and collecting results from the vector jobs with the runJob method.

Parameters:

`a_script_array`**:** A script_array object.

`job_results`**:** The index within the vector job.

Returns:

job_results: Any output produced by the job.

Example:

```
Call it directly:
» runLast(script_array(10, 'this one does nothing for 10 times'), );
```

See also: `runJob` (p. **??**), `runFirst` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/02/01

A.41.6 Method `script_array/set`

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/02/06

A.41.7 Method `script_array/subsasgn`

Summary: Defines generic index-based assignment for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/02/06

A.41.8 Method `script_array/subsref`

Summary: Defines generic indexing for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.42 Class `script_array_for_cluster`

A.42.1 Constructor `script_array_for_cluster/script_array_for_cluster`

Summary: Generic class defining a repetitive vector job to be run on a Sun Grid Engine (SGE) computing cluster.

Usage:

```
a_script_cluster = script_array_for_cluster(num_runs, sge_wrapper_script, id, props)
```

Description: This is a subclass of the script_array class. The runFirst method spawns num_runs copies of the runJob method in parallel on the cluster, followed by the invocation of the runLast method.

Parameters:

`num_runs`: The number of times the runJob script should be evoked.

`sge_wrapper_script`: A script that can be submitted with qsub and can execute arbitrary

Matlab commands on the cluster nodes. It can have qsub options prepended to it such as '-p -100 -q all.q <abs_path_to>/sge_matlab.sh'.

`id`: Identification string.

`props`: A structure with any optional properties.

`notifyByMail`: An SGE notification email is sent to this address after lastJob.
(others passed to script_array)

Returns a structure object with the following fields:

num_runs, id, props.

See also: `runFirst` (p. **??**), `runLast` (p. **??**), `runJob` (p. **??**), `script_array` (p. 208)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/02/02

A.42.2 Method `script_array_for_cluster/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.42.3 Method `script_array_for_cluster/runFirst`

Summary: Method to be called at beginning of script_array_for_cluster jobs.

Usage:

```
job_results = runFirst(a_script_cluster)
```

Description: This method initiates the script_array_for_cluster jobs. It submits an SGE vector job for running each runJob and finally runLast. There is no way of collecting outputs from individual runJob calls.

Parameters:

`a_script_cluster`**:** A script_array_for_cluster object.

Returns:

job_results: A cell array of results collected from each item of the vector jobs.

Example:

```
» runFirst(script_array_for_cluster(10, 'this one does nothing for 10 times'));
```

See also: `script_array_for_cluster` (p. 211)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/02/01

A.42.4 Method `script_array_for_cluster/set`

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/02/06

A.43 Class `script_array_loaddb`

A.43.1 Constructor `script_array_loaddb/script_array_loaddb`

Summary: Loads dataset items into a database by partitioning into parallel tasks.

Usage:

```
a_s = script_array_loaddb(num_runs, a_dataset, id, props)
```

Description: This is a subclass of the script_array class. It will analyze and load items in a params_tests_dataset (and subclass) objects by partitioning it into num_runs pieces, to be loaded in parallel on multicore machines. Partitioned loading of databases can also be beneficial in serial (by setting prop 'parallel' to 0) for large datasets, such that problematic items (that crash) do not hinder the loading of the rest of the dataset.

Parameters:

`num_runs`: The number of times the runJob script should be evoked.

`sge_wrapper_script`: A script that can be submitted with qsub and can execute arbitrary

Matlab commands on the cluster nodes. It can have qsub options prepended to it such as '-p -100 -q all.q <abs_path_to>/sge_matlab.sh'.

`id`: Identification string.

`props`: A structure with any optional properties.

`items`: If specified, only load items in this horizontal vector.
(others passed to script_array)

Returns a structure object with the following fields:

dataset, script_array.

See also: `runFirst` (p. **??**), `runLast` (p. **??**), `runJob` (p. **??**), `script_array` (p. 208)

Author: Cengiz Gunay <cgunay@emory.edu>, 2014/04/02

A.43.2 Method `script_array_loaddb/runJob`

Summary: Loads one piece of the database.

Usage:

```
a_db_piece = runJob(a_s, vector_index)
```

Description: Load the part of the database calculated by the index. Uses the params_tests_dataset/params_tests_db method to load the database.

Parameters:

`a_s`: A script_array_loaddb object.

`vector_index`: The index within the vector job.

Returns:

a_db_piece: Piece of loaded database.

See also: `script_array_loaddb` (p. 212), `runLast` (p. **??**), `runFirst` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2014/04/02

A.43.3 Method `script_array_loaddb/runLast`

Summary: Method to be called last after the script_array jobs.

Usage:

```
a_db = runLast(a_script_array, job_results)
```

Description: Combines the separately loaded databases found in job_results into a final one and returns it.

Parameters:

`a_script_array`: A script_array object.

`job_results`: The index within the vector job.

Returns:

a_db: Concatenated database.

See also: `script_array_loaddb` (p. 212), `runJob` (p. **??**), `runFirst` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2014/04/02

A.44 Class `script_factory`

A.44.1 Constructor `script_factory/script_factory`

Summary: Generic class to automatically create a set of scripts.

Usage:

```
obj = script_factory(num_scripts, out_name, id, props)
```

Description: This is the base class for all script_factory classes.

Parameters:

`num_scripts`: Number of scripts to create.

`out_name`: The file name for the output scripts. A ' filename corresponds to the script number.

`id`: Identification string.

`props`: A structure with any optional properties.

Returns a structure object with the following fields:

num_scripts, out_name, id, props.

See also: `script_factory/writeScripts` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/11/28

A.44.2 Method `script_factory/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.45 Class `spike_shape`

A.45.1 Constructor `spike_shape/spike_shape`

Summary: An action potential shape trace.

Usage:

```
obj = spike_shape(data, dt, dy, id)
```

Parameters:

`data`: A vector of data points containing the spike shape.

`dt`: Time resolution [s].

`dy`: y-axis resolution [ISI (V, A, etc.)]

`id`: Identification string.

`props`: A structure with any optional properties.

`baseline`: Resting potential.

`threshold`: Spike threshold.

`init_Vm_method`: Method to obtain spike initiation voltage.
1- maximum acceleration point 2- threshold crossing of acceleration (needs threshold) 3- threshold crossing of slope (needs threshold) 4- maximum acceleration in phase space (optionally specify maximal threshold as init_threshold) 5- point of maximum curvature, when slope is between init_lo_thr and init_hi_thr 6- local maximum of second derivative in the phase space nearest slope crossing init_threshold 7- threshold crossing of interpolated slope (needs threshold) 8- maximum curvature in phase-plane 9- Combined curvature and inflection method in time-domain.

`init_threshold`: Spike initiation threshold (deriv or accel).
(see above methods and implementation in calcInitVm)

`init_lo_thr, init_hi_thr`: Low and high thresholds for slope.

Returns a structure object with the following fields:

trace, props.

See also: `trace/spike_shape` (p. 333), `trace/analyzeSpikesInPeriod` (p. 319), `trace` (p. 317), `spikes` (p. 227), `period` (p. 162)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/07/30

A.45.2 Method `spike_shape/calcInitVm`

Summary: Calculates spike threshold related measures of the spike_shape, s.

Usage:

```
[init_val, init_idx, rise_time, amplitude, peak_mag, peak_idx, max_d1o, a_plot] =
calcInitVm(s, max_idx, min_idx)
```

Parameters:

`s`: A spike_shape object.

`max_idx`: The index of the maximal point of the spike_shape [dt].

`min_idx`: The index of the minimal point of the spike_shape [dt].

`plotit`: If non-zero, plot a graph annotating the test results (optional).

Returns:

init_val: The potential value [dy]. init_idx: Its index in the spike_shape [dt]. rise_time: Time from initiation to maximum [dt]. amplitude: Magnitude from initiation to max [dy]. peak_mag: Peak value [dy]. peak_idx: Extrapolated spike peak index [dt]. max_d1o: Maximal value of first voltage derivative [dy]. a_plot: plot_abstract, if requested.

See also: `spike_shape` (p. 215)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/02

A.45.3 Method `spike_shape/calcInitVmLtdMaxCurv`

Summary: Calculates the action potential threshold using the maximum of the curvature equation only in the limited range given with two voltage slope thresholds.

Usage:

```
[init_idx, a_plot] = calcInitVmLtdMaxCurv(s, max_idx, min_idx, lo_thr, hi_thr, plotit)
```

Description: Point of maximum curvature: Kp = V"[1 + (V')^2]^(-3/2) Taken from Sekerli, Del Negro, Lee and Butera. IEEE Trans. Biomed. Eng., 51(9): 1665-71, 2004.

Parameters:

`s`: A spike_shape object.

`max_idx`: The index of the maximal point of the spike_shape [dt].

`min_idx`: The index of the minimal point of the spike_shape [dt].

`lo_thr, hi_thr`: Lower and higher thresholds for time derivative of voltage.

`plotit`: If non-zero, plot a graph annotating the test results (optional).

Returns:

init_idx: AP threshold index in the spike_shape [dt]. a_plot: plot_abstract, if requested.

See also: `calcInitVm` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/19

A.45.4 Method `spike_shape/calcInitVmMaxCurvature`

Summary: Calculates the action potential threshold using the maximum of the curvature equation.

Usage:

```
[init_idx, a_plot] = calcInitVmMaxCurvature(s, max_idx, min_idx, plotit)
```

Description: Point of maximum curvature: Kp = V"[1 + (V')^2]^(-3/2) Taken from Sekerli, Del Negro, Lee and Butera. IEEE Trans. Biomed. Eng., 51(9): 1665-71, 2004.

Parameters:

`s`: A spike_shape object.

`max_idx`: The index of the maximal point of the spike_shape [dt].

`min_idx`: The index of the minimal point of the spike_shape [dt].

`plotit`: If non-zero, plot a graph annotating the test results (optional).

Returns:

init_idx: AP threshold index in the spike_shape [dt]. a_plot: plot_abstract, if requested.

See also: `calcInitVm` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/19

A.45.5 Method `spike_shape/calcInitVmMaxCurvPhasePlane`

Summary: Calculates the voltage at the maximum curvature in the phase plane as action potential threshold.

Usage:

```
[init_idx, max_d1o, a_plot, fail_cond] = calcInitVmMaxCurvPhasePlane(s, max_idx, min_idx,
plotit)
```

Description: First take the phase-plane v'-v from the beginning to max(v'). Then regulate intervals by interpolation. Point of maximum curvature: Kp = V"[1 + (V')^2]^(-3/2) Taken from Sekerli, Del Negro, Lee and Butera. IEEE Trans. Biomed. Eng., 51(9): 1665-71, 2004.

Parameters:

`s`: A spike_shape object.

`max_idx`: The index of the maximal point of the spike_shape [dt].

`min_idx`: The index of the minimal point of the spike_shape [dt].

`plotit`: If non-zero, plot a graph annotating the test results (optional).

Returns:

init_idx: AP threshold index in the spike_shape [dt]. max_d1o: Maximal value of first voltage derivative [dy]. a_plot: plot_abstract, if requested. fail_cond: True if this algorithm fails to be trustable.

See also: `calcInitVm` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/04/12

A.45.6 Method `spike_shape/calcInitVmSekerliV2`

Summary: Calculates the action potential threshold using the maximum second derivative of the phase space of voltage-time slope versus voltage.

Usage:

```
[init_idx, a_plot] = calcInitVmSekerliV2(s, max_idx, min_idx, plotit)
```

Parameters:

`s`: A spike_shape object.

`max_idx`: The index of the maximal point of the spike_shape [dt].

`min_idx`: The index of the minimal point of the spike_shape [dt].

`plotit`: If non-zero, plot a graph annotating the test results (optional).

Returns:

init_idx: Its index in the spike_shape [dt]. a_plot: plot_abstract, if requested.

See also: calcInitVm (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/18

A.45.7 Method spike_shape/calcInitVmSlopeThreshold

Summary: Calculates the AP threshold using the slope threhold crossing.

Usage:

```
[init_idx, a_plot] = calcInitVmSlopeThreshold(s, max_idx, min_idx, thr, plotit)
```

Parameters:

`s`: A spike_shape object.

`max_idx`: The index of the maximal point of the spike_shape [dt].

`min_idx`: The index of the minimal point of the spike_shape [dt].

`thr`: Threshold for time derivative of voltage.

`plotit`: If non-zero, plot a graph annotating the test results (optional).

Returns:

init_idx: AP threshold index in the spike_shape [dt]. a_plot: plot_abstract, if requested.

See also: calcInitVm (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/24

A.45.8 Method spike_shape/calcInitVmSlopeThresholdSupsample

Summary: Estimates the AP threshold as the first slope threshold crossing by first supersampling the data using cubic spline interpolation.

Usage:

```
[init_idx, a_plot] = calcInitVmSlopeThresholdSupsample(s, max_idx, min_idx, thr, plotit)
```

Parameters:

`s`: A spike_shape object.

`max_idx`: The index of the maximal point of the spike_shape [dt].

`min_idx`: The index of the minimal point of the spike_shape [dt].

`thr`: Threshold for time derivative of voltage.

`plotit`: If non-zero, plot a graph annotating the test results (optional).

Returns:

init_idx: AP threshold index in the spike_shape [dt]. a_plot: plot_abstract, if requested.

See also: `calcInitVm` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/03/23

A.45.9 Method `spike_shape/calcInitVmV2PPLocal`

Summary: Calculates the action potential threshold by finding the local second derivative maximum in voltage-time slope versus voltage phase plane, nearest a slope threshold crossing.

Usage:

```
[init_idx, a_plot] = calcInitVmV2PPLocal(s, max_idx, min_idx, lo_thr, plotit)
```

Parameters:

`s`: A spike_shape object.

`max_idx`: The index of the maximal point of the spike_shape [dt].

`min_idx`: The index of the minimal point of the spike_shape [dt].

`lo_thr`: Lower threshold for time voltage slope.

`plotit`: If non-zero, plot a graph annotating the test results (optional).

Returns:

init_idx: Its index in the spike_shape [dt]. a_plot: plot_abstract, if requested.

See also: `calcInitVm` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/18

A.45.10 Method `spike_shape/calcInitVmV3hKpTinterp`

Summary: Calculates candidates for action potential threshold using the first three time-domain derivatives.

Usage:

```
[init_idx, a_plot] = calcInitVmV3hKpTinterp(s, max_idx, min_idx, lo_thr, hi_thr, plotit)
```

Description: First uses interpolation to increase time points. Calculates h, the second derivative of phase-plane (d^2 v'/dv^2), in terms of time-domain derivatives. Also calculates Kp = V"[1 + (V')^2]^(-3/2), the curvature. The maxima of these functions are used as candidates for AP thresholds.

Parameters:

`s`: A spike_shape object.

`max_idx`: The index of the maximal point of the spike_shape [dt].

`min_idx`: The index of the minimal point of the spike_shape [dt].

`lo_thr, hi_thr`: Lower and higher thresholds for time derivative of voltage.

`plotit`: If non-zero, plot a graph annotating the test results (optional).

Returns:

init_idx: Indices of threshold candidates in the spike_shape [dt]. a_plot: plot_abstract, if requested.

See also: `calcInitVm` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/18

A.45.11 Method `spike_shape/calcMaxVm`

Summary: Calculates the maximal value of the spike_shape, s.

Usage:

```
[max_val, max_idx] = calcMaxVm(s)
```

Parameters:

`s`: A spike_shape object.

Returns:

max_val: The max value. max_idx: Its index in the spike_shape [dt].

See also: `period` (p. 162), `spike_shape` (p. 215), `trace/calcMax` (p. 321)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/02

A.45.12 Method `spike_shape/calcMinVm`

Summary: Calculates the minimal value of the spike_shape, s.

Usage:

```
[min_val, min_idx, max_min_time] = calcMinVm(s, max_idx)
```

Parameters:

`s`: A spike_shape object.

`max_idx`: The index of the maximal point of the spike_shape [dt].

Returns:

min_val: The min value [dy]. min_idx: Its index in the spike_shape [dt]. max_min_time: Time from max to min [dt].

See also: `period` (p. 162), `spike_shape` (p. 215), `trace/calcMin` (p. 322)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/02

A.45.13 Method `spike_shape/calcWidthFall`

Summary: Calculates the spike width and fall information of the spike_shape, s.

Usage:

```
[base_width, half_width, half_Vm, fall_time, min_idx, min_val, max_ahp, ahp_decay_constant,
dahp_mag, dahp_idx] = ...  calcWidthFall(s, max_idx, max_val, init_idx, init_val)
```

Description: max_* can be the peak_* from calcInitVm.

Parameters:

`s`: A spike_shape object.

`max_idx`: The index of the maximal point [dt].

`max_val`: The value of the maximal point [dy].

`init_idx`: The index of spike initiation point [dt].

`init_val`: The value of spike initiation point [dy].

`fixed_Vm`: The desired height for width calculation [V].

Returns:

base_width: Width of spike at base [dt] half_width: Width of spike at half_Vm [dt] half_Vm: Half height of spike [dy] fall_time: Time from peak to initialization level [dt]. min_idx: The index of the minimal point of the spike_shape [dt]. max_ahp: Magnitude from initiation to minimum [dy]. ahp_decay_constant: Approximation to refractory decay after maxAHP [dt]. dahp_mag: Magnitude of the double AHP peak dahp_mag: Index of the double AHP peak

See also: spike_shape (p. 215)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/02

A.45.14 Method spike_shape/display

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.45.15 Method spike_shape/get

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.45.16 Method spike_shape/getResults

Summary: Runs all tests defined by this class and return them in a structure.

Usage:

```
[results, a_plot] = getResults(s, plotit)
```

Parameters:

`s`: A spike_shape object.

`plotit`: If non-zero, plot a graph annotating the test results (optional).

Returns:

results: A structure associating test names to values in ms and mV. a_plot: plot_abstract, if requested.

See also: spike_shape (p. 215)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/02

A.45.17 Method `spike_shape/plotCompareMethods`

Summary: Creates a multi-plot comparing different action potential threshold finding methods.

Usage:

```
a_plot = plotCompareMethods(s, title_str)
```

Parameters:

`s`: A spike_shape object.

`title_str`: Title suffix (optional).

Returns:

a_plot: A plot_abstract object that can be visualized.

See also: `spike_shape` (p. 215), `plot_abstract` (p. 180)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/19

A.45.18 Method `spike_shape/plotCompareMethodsSimple`

Summary: Creates a multi-plot comparing different action potential threshold finding methods.

Usage:

```
a_plot = plotCompareMethodsSimple(s, title_str)
```

Parameters:

`s`: A spike_shape object.

`title_str`: Title suffix (optional).

Returns:

a_plot: A plot_abstract object that can be visualized.

See also: `spike_shape` (p. 215), `plot_abstract` (p. 180)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/19

A.45.19 Method `spike_shape/plotPP`

Summary: Plots the dV/dt vs. V phase-plane representation of the spike shape.

Usage:

```
a_plot = plotPP(s)
```

Parameters:

`s`: A spike_shape object.

Returns:

a_plot: A plot_abstract object that can be visualized.

See also: `spike_shape` (p. 215), `plot_abstract` (p. 180)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/16

A.45.20 Method `spike_shape/plotResults`

Summary: Plots the spike shape annotated with result characteristics.

Usage:

```
a_plot = plotResults(s, title_str, props)
```

Parameters:

`s`: A spike_shape object.

Returns:

a_plot: A plot_abstract object that can be visualized. title_str: (Optional) String to append to plot title. props: A structure with any optional properties, passed to trace/plotData.

See also: `spike_shape` (p. 215), `plot_abstract` (p. 180)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/17

A.45.21 Method `spike_shape/plotTPP`

Summary: Plots the dV/dt vs. V phase-plane representation of the spike shape.

Usage:

```
a_plot = plotTPP(s)
```

Description: Uses the Taylor series estimation for finding the derivative dV/dt.

Parameters:

`s`: A spike_shape object.

Returns:

a_plot: A plot_abstract object that can be visualized.

See also: `spike_shape` (p. 215), `plot_abstract` (p. 180), `diffT` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/16

A.45.22 Method `spike_shape/set`

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.46 Class `spike_shape_profile`

A.46.1 Constructor `spike_shape_profile/spike_shape_profile`

Summary: Holds the results profile from a spike_shape object.

Usage:

```
a_ss_profile = spike_shape_profile(results, a_spike_shape, props)
```

Parameters:

`results`: A structure containing test results.

`a_spike_shape`: A spike_shape object.

`props`: A structure with any optional properties.

Returns a structure object with the following fields:

results_profile: Contains results of tests. spike_shape: The spike_shape object from which results were obtained. props.

See also: `results_profile` (p. 206)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/08/17

A.46.2 Method `spike_shape_profile/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.46.3 Method `spike_shape_profile/plot_abstract`

Summary: Plots the spike shape with measurements marked in red.

Usage:

```
a_plot = plot_abstract(s, props)
```

Parameters:

`s`: A spike_shape object.

`props`: A structure with any optional properties.

`absolute_peak_time`: Shift the peak to this point on the plot.

`no_plot_spike`: Do not plot the spike shape first.

Returns:

a_plot: A plot_abstract object that can be visualized.

See also: `spike_shape` (p. 215), `plot_abstract` (p. 180)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/08/17

A.47 Class `spikes`

A.47.1 Constructor `spikes/spikes`

Summary: Spike times from a trace.

Usage:

```
obj = spikes(times, num_samples, dt, id)
```

Parameters:

`times`: The spike times [dt].

`num_samples`: Number of samples in the original trace.

`dt`: Time resolution [s].

`id`: Identification string.

Returns a structure object with the following fields:

times, num_samples, dt, id.

See also: `trace/spikes` (p. 333), `trace` (p. 317), `spike_shape` (p. 215), `period` (p. 162)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/07/30

A.47.2 Method `spikes/addSpikes`

Usage:

```
s = addSpike(s, times)
```

Parameters:

`s`: A spikes object.

`times`: Times of spikes to add

Returns:

s: The updated object.

See also: `spikes` (p. 227)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/08/16

A.47.3 Method `spikes/display`

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.47.4 Method `spikes/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.47.5 Method `spikes/getISIs`

Summary: Calculates the firing rate of the spikes found in the given period with an averaged inter-spike-interval approach.

Usage:

```
isi = getISIs(s, period)
```

Parameters:

`s`: A spikes object.

`period`: The period where spikes were found (optional)

Returns:

isi: Inter-spike-interval vector [dt]

See also: `trace` (p. 317), `spikes` (p. 227), `period` (p. 162)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/03/09

A.47.6 Method `spikes/getResults`

Summary: Runs all tests defined by this class and return them in a structure.

Usage:

```
results = getResults(s)
```

Parameters:

`s`: A spikes object.

Returns:

results: A structure associating test names to values in ms and mV (or mA).

See also: `spikes` (p. 227)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/13

A.47.7 Method `spikes/intoPeriod`

Summary: Shifts the spikes times to be within the given period.

Usage:

```
obj = intoPeriod(s, a_period)
```

Description: Assuming this spikes object's length fits into the given period, it shifts all times to start from the beginning of the given period. This may be used to reconstruct the original spikes object from subperiods that were cut out previously, using the withinPeriod method.

Parameters:

`s`: A spikes object.

`a_period`: The desired period

Returns:

obj: A spikes object

See also: `spikes` (p. 227), `period` (p. 162)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/07/31

A.47.8 Method `spikes/ISICV`

Summary: Calculates the coefficient of variation (CV) of the inter-spike-intervals (ISI).

Usage:

```
cv = ISICV(s, a_period)
```

Parameters:

`s`: A spikes object.

`a_period`: The period where spikes were found (optional)

Returns:

cv: Coefficient of variation.

See also: `spikes` (p. 227), `period` (p. 162)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/13

A.47.9 Method `spikes/periodWhole`

Summary: Returns the boundaries of the whole period of spikes, s.

Usage:

```
whole_period = periodWhole(s)
```

Parameters:

`s`: A spikes object.

See also: `period` (p. 162), `spikes` (p. 227)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/07/30

A.47.10 Method `spikes/plot`

Summary: Plots spikes.

Usage:

```
h = plot(t)
```

Parameters:

`t`: A spikes object.

`title_str`: (Optional) String to append to plot title.

Returns:

h: Handle to figure object.

See also: `spikes` (p. 227), `plot_abstract` (p. 180)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.47.11 Method `spikes/plotData`

Summary: Plots a spikes object.

Usage:

```
a_plot = plotData(s, title_str)
```

Description: If s is a vector of spikes objects, returns a vector of plot objects.

Parameters:

`s`: A spikes object.

Returns:

a_plot: A plot_abstract object that can be visualized. title_str: (Optional) String to append to plot title.

See also: `trace` (p. 317), `plot_abstract` (p. 180)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/10/21

A.47.12 Method `spikes/plotFreqVsTime`

Summary: Plots a frequency-time graph from the spikes object.

Usage:

```
a_plot = plotFreqVsTime(s, title_str, props)
```

Description: If s is a vector of spikes objects, returns a vector of plot objects.

Parameters:

`s`: A spikes object.

`title_str`: (Optional) String to append to plot title.

`props`: A structure with any optional properties.

`timeScale`: 's' for seconds, or 'ms' for milliseconds.

`type`: If 'simple' plots 1/is for each spike time, 'manhattan' uses flat lines of 1/isi height between spike times (default). (others passed to plot_abstract)

Returns:

a_plot: A plot_abstract object that can be visualized. title_str: (Optional) String to append to plot title.

See also: `trace` (p. 317), `plot_abstract` (p. 180)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/05/05

A.47.13 Method `spikes/plotISIs`

Summary: Plots a spikes object.

Usage:

```
a_plot = plotISIs(s, title_str)
```

Description: If s is a vector of spikes objects, returns a vector of plot objects.

Parameters:

`s`: A spikes object.

Returns:

a_plot: A plot_abstract object that can be visualized. title_str: (Optional) String to append to plot title.

See also: `trace` (p. 317), `plot_abstract` (p. 180)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/10/21

A.47.14 Method `spikes/set`

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.47.15 Method `spikes/SFA`

Summary: Calculates the spike frequency accommodation (SFA) of the inter-spike-intervals (ISI).

Usage:

```
sfa = SFA(s, a_period)
```

Description: SFA is the ration of the last ISI to the first ISI in the period.

Parameters:

`s`: A spikes object.

`a_period`: The period where spikes were found (optional)

Returns:

sfa: Spike frequency accommodation.

See also: `spikes` (p. 227), `period` (p. 162)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/13

A.47.16 Method `spikes/spikeAmpSlope`

Summary: Calculates the time constant and steady-state value of the spike amplitude for slow inactivating decays.

Usage:

```
[a_tau, da_inf] = spikeAmpSlope(a_spikes, a_trace, a_period)
```

Parameters:

`a_spikes`: A spikes object.

`a_trace`: A trace object.

`a_period`: The desired period (optional)

Returns:

a_tau: Approximate amplitude decay constant. da_inf: Delta change in final spike peak value from initial.

See also: `period` (p. 162), `spikes` (p. 227), `trace` (p. 317)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/15

A.47.17 Method `spikes/spikeRate`

Summary: Calculates the average firing rate [Hz] of the given spike train.

Usage:

```
freq = spikeRate(s, a_period)
```

Parameters:

`s`: A spikes object.

`a_period`: The period where spikes were found (optional)

Returns:

freq: Firing rate [Hz]

See also: spikes (p. 227), period (p. 162)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/03/09

A.47.18 Method spikes/spikeRateISI

Summary: Calculates the firing rate of the spikes found in the given period with an averaged inter-spike-interval approach.

Usage:

```
freq = spikeRateISI(s, trace_index, times, period)
```

Parameters:

`s`: A spikes object.

`period`: The period where spikes were found (optional)

Returns:

freq: Firing rate [Hz]

See also: trace (p. 317), spikes (p. 227), period (p. 162)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/03/09

A.47.19 Method spikes/subsref

Summary: Defines generic indexing for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.47.20 Method spikes/vertcat

Summary: Vertical concatanation [a_spikes;with_spikes;...] operator.

Usage:

```
a_spikes = vertcat(a_spikes, with_spikes, ...)
```

Description: Concatanates spike times of with_spikes with that of a_spikes. Overrides the built-in vertcat function that is called when [a_spikes;with_spikes] is executed.

Parameters:

a_spikes, with_spikes, ...: Spikes objects.

Returns:

a_spikes: A tests_spikes that contains times of all given spikes objects.

See also: vertcat (p. **??**), spikes (p. 227)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/08/16

A.47.21 Method spikes/withinPeriod

Summary: Returns a spikes object valid only within the given period, subtracts the offset.

Usage:

```
obj = withinPeriod(s, a_period)
```

Parameters:

s: A spikes object.

a_period: The desired period

Returns:

obj: A spikes object

See also: spikes (p. 227), period (p. 162)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/07/31

A.47.22 Method spikes/withinPeriodWOffset

Summary: Returns a spikes object valid only within the given period, keeps the offset.

Usage:

```
obj = withinPeriodWOffset(s, a_period)
```

Parameters:

s: A spikes object.

a_period: The desired period

Returns:

obj: A spikes object

See also: spikes (p. 227), period (p. 162)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/05/09

A.48 Class `spikes_db`

A.48.1 Constructor `spikes_db/spikes_db`

Summary: A database of spike shape results obtained from a period in a trace.

Usage:

```
a_spikes_db = spikes_db(data, col_names, a_trace, a_period, id, props)
```

Description: This is a subclass of tests_db. Use trace/analyzeSpikesInPeriod to get an instance of this class.

Parameters:

`data`**:** Database contents.

`col_names`**:** The column names.

`a_trace`**:** The trace where the spikes were found.

`a_period`**:** The period inside a_trace where spikes were found.

`id`**:** An identifying string.

`props`**:** A structure with any optional properties.

Returns a structure object with the following fields:

tests_db, trace, period, props.

See also: `tests_db` (p. 256), `trace` (p. 317), `period` (p. 162), `trace/analyzeSpikesInPeriod` (p. 319)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/08/17

A.48.2 Method `spikes_db/plot_abstract`

Summary: Visualizes the spikes_db by marking spike shapes measurements on the trace plot.

Usage:

```
a_pm = plot_abstract(a_db, title_str, props)
```

Parameters:

`a_db`**:** A spikes_db object.

`title_str`**:** (Optional) A string to be concatanated to the title.

`props`**:** A structure with any optional properties passed to trace/plotData.

Returns:

a_pm: A trace plot.

See also: plot_abstract/plot_abstract (p. 180), tests_db/plot_abstract (p. 298), plotFigure (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/08/17

A.49 Class sql_portal

A.49.1 Constructor sql_portal/sql_portal

Summary: For import and export to external SQL engines using the Matlab Database Toolbox.

Usage:

```
obj = sql_portal(db_conn, id, props)
```

Description: Uses the Database (DB) Toolbox's connection object to read and write to external SQL engines.

Parameters:

db_conn: An object of the database class of the DB Toolbox.

id: An identifying string.

props: A structure with any optional properties.

Returns a structure object with the following fields:

db_conn: The DB Toolbox connection object. id, props.

See also: tests_db (p. 256), database (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2007/11/29

A.49.2 Method sql_portal/get

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.49.3 Method `sql_portal/sql_statement`

Summary: Run an SQL statement and discard its results.

Usage:

```
response_str = sql_statement(a_sql_portal, statement_string, props)
```

Description: This function is for sending SQL statements that do not return any data, for such functions as inserting data into a database, creating views, and running administration commands. See sql_portal/tests_db to import select query results into Pandora.

Parameters:

`a_sql_portal`: A sql_portal object.

`statement_string`: An SQL statement that does not return data.

`props`: A structure with any optional properties.

Returns:

response_str: Response string from the SQL engine.

See also: `sql_portal/tests_db` (p. 239), `database` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2008/04/18

A.49.4 Method `sql_portal/sql_table`

Summary: Create an SQL table from the contents of a tests_db object.

Usage:

```
sql_table(a_sql_portal, a_tests_db, table_label, props)
```

Description: Converter function to get a tests_db object properly annotated with the metadata obtained from the results of the executed SQL query. Currently this function is limited to importing numeric data only.

Parameters:

`a_sql_portal`: A sql_portal object.

`a_tests_db`: A tests_db object.

`table_label`: Name of the newly created table.

`props`: A structure with any optional properties.

Returns:

See also: `tests_db/tests_db` (p. 256), `database` (p. **??**), `sql_portal/tests_db` (p. 239)

Author: Cengiz Gunay <cgunay@emory.edu>, 2008/12/17

A.49.5 Method `sql_portal/subsref`

Summary: Defines generic indexing for objects.

A.49.6 Method `sql_portal/tests_db`

Summary: Create a tests_db object from the results of a SQL query.

Usage:

```
a_db = tests_db(a_sql_portal, query_string, query_id, props)
```

Description: Converter function to get a tests_db object properly annotated with the metadata obtained from the results of the executed SQL query. Currently this function is limited to importing numeric data only.

Parameters:

`a_sql_portal`**:** A sql_portal object.

`query_string`**:** An SQL query returning numeric results.

`query_id`**:** Identifier associated witht the query, to be passed into the tests_db object.

`props`**:** A structure with any optional properties passed to tests_db.

Returns:

a_db: A tests_db object.

See also: `tests_db` (p. 256), `database` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2007/11/29

A.50 Class `stats_db`

A.50.1 Constructor `stats_db/stats_db`

Summary: A database of rows corresponding to statistical distribution properties of tests. Multiple pages can be used to indicate another dimension.

Usage:

```
a_stats_db = stats_db(test_results, col_names, row_names, page_names, id, props)
```

Description: This is a subclass of tests_3D_db. Allows generating a plot, etc.

Parameters:

`test_results`**:** The 3-d array of rows, columns, and pages. Will*/ also accept a tests_db object, but row_names must also be supplied.

`col_names`: Test names in this db.

`row_names`: Statistical test names for each row (e.g., mean, STD, SE, etc.)

`page_names`: Meaning of each separate page of data
(e.g., a different invariant parameter).

`id`: An identifying string.

`props`: A structure with any optional properties.

`axis_limits`: Limits in the form of [xmin xmax ymin ymax]
for errorbar axes.

`yTicksPos`: 'left' means only put y-axis ticks to leftmost plot.

`xTicksPos`: 'bottom' means only put x-axis ticks to lowest plot.

Returns a structure object with the following fields:

tests_3D_db.

See also: `tests_3D_db` (p. 246), `plot_abstract` (p. 180)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/07

A.50.2 Method `stats_db/compareStats`

Summary: Merges multiple stats_dbs into pages of a single stats_db for comparison.

Usage:

```
a_mult_stats_db = compareStats(a_stats_db, a_2nd_stats_db, ...)
```

Description: Generates a plot_simple object from this histogram.

Parameters:

`a_stats_db`: A stats_db object.

Returns:

a_mult_stats_db: A multi-page stats_db.

See also: `plot_abstract` (p. 180), `plot_simple` (p. 193)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.50.3 Method `stats_db/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.50.4 Method `stats_db/onlyRowsTests`

Summary: Returns a tests_db that only contains the desired tests and rows (and pages).

Usage:

```
obj = onlyRowsTests(obj, rows, tests, pages)
```

Description: Selects the given dimensions and returns in a new tests_db object.

Parameters:

`obj`: A tests_db object.

`rows`: A logical or index vector of rows. If ':', all rows.

`tests`: Cell array of test names or column indices. If ':', all tests.

`pages`: (Optional) A logical or index vector of pages. ':' for all pages.

Returns:

obj: The new tests_db object.

See also: `subsref` (p. **??**), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/17

A.50.5 Method `stats_db/plotColorVar`

Summary: Create a color-plot of parameter-test variations in a matrix.

Usage:

```
a_plot = plotColorVar(p_stats, props)
```

Description: Skips the 'ItemIndex' test.

Parameters:

`p_stats`: Array of invariant parameter databases obtained from calling tests_3D_db/paramsTestsHistsStats.

`title_str`: (Optional) String to append to plot title.

`props`: A structure with any optional properties, passed to plot_stack.

`plotMethod`: 'plotVar' uses stats_db/plotVar (default)
'plot_bars' uses stats_db/plot_bars

Returns:

a_plot: A plot_abstract with the color plot

See also: `paramsTestsHistsStats` (p. **??**), `params_tests_profile` (p. 162), `plotVar.` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/17

A.50.6 Method `stats_db/plotVar`

Summary: Generates a plot of the variation between two tests.

Usage:

```
a_plot = plotVar(a_stats_db, test1, test2, props)
```

Description: Creates a plot description where the mean values are used for solid lines and the std values of test2 is indicated with errorbars. It is assumed that each page of the stats_db contains a value to be matched.

Parameters:

`a_stats_db`: A stats_db object.

`test1`: Test column for the x-axis, only mean values are used.

`test2`: Test column for the y-axis, std values are indicated with errorbars.

`title_str`: (Optional) String to append to plot title.

`props`: Optional properties.

`plotType`: 1, only errorbars (default); 2, errorbars extending from bars.

`quiet`: If 1, only display given title_str.

(rest passed to plot_abstract)

Returns:

a_plot: A plot_abstract object or one of its subclasses.

See also: `plotVar` (p. **??**), `plot_simple` (p. 193)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/13

A.50.7 Method `stats_db/plotVarMatrix`

Summary: Create a stack of parameter-test variation plots organized in a matrix.

Usage:

```
a_plot_stack = plotVarMatrix(p_stats, props)
```

Description: Skips the 'ItemIndex' test.

Parameters:

`p_stats`: Array of invariant parameter databases obtained from calling tests_3D_db/paramsTestsHistsStats.

`props`: A structure with any optional properties, passed to plot_stack.

`plotMethod`: 'plotVar' uses stats_db/plotVar (default)
'plot_bars' uses stats_db/plot_bars

`rotateYLabel`: Rotate row labels this much (default=60).

Returns:

a_plot_stack: A plot_stack with the plots organized in matrix form

See also: `paramsTestsHistsStats` (p. **??**), `params_tests_profile` (p. 162), `plotVar.` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/17

A.50.8 Method `stats_db/plotYTests`

Summary: Create an errorbar plot of database stats measures against given X-axis values.

Usage:

```
a_p = plotYTests(a_stats_db, x_vals, tests, axis_labels, title_str, short_title, command,
props)
```

Parameters:

`a_stats_db`: A params_tests_db object.

`x_vals`: A vector of X-axis values.

`tests`: A vector or cell array of columns to correspond to each value from x_vals.

`title_str`: (Optional) A string to be concatanated to the title.

`short_title`: (Optional) Few words that may appear in legends of multiplot.

`command`: (Optional) Command to do the plotting with (default: 'plot')

`props`: A structure with any optional properties.

`LineStyle`: Plot line style to use. (default: 'd-')

`quiet`: If 1, don't include database name on title.

Returns:

a_p: A plot_abstract.

Example:

```
» a_p = plotYTests(a_stats_db, [0 40 100 200], ...
'IniSpontSpikeRateISI_0pA', 'PulseIni100msSpikeRateISI_D40pA', ...
'PulseIni100msSpikeRateISI_D100pA', 'PulseIni100msSpikeRateISI_D200pA', ...
'current pulse [pA]', 'firing rate [Hz]', ', f-I curves', 'neuron 1');
» plotFigure(a_p);
```

See also: `plotFigure` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/01/23

A.50.9 Method `stats_db/plot_abstract`

Summary: Generates an error bar graph for each db columns.

Usage:

```
a_plot = plot_abstract(a_stats_db, title_str, props)
```

Description: Generates a plot_simple object from this histogram. Looks for 'mean', 'min', 'max', and 'STD' labels in the row_idx for drawing the errorbars. Each column of a_stats_db is shown in a separate axis. Values from multiple pages of a_stats_db are shown as distinct points in the axis.

Parameters:

`a_stats_db`**:** A histogram_db object.

`title_str`**:** A title string on the plot

`props`**:** A structure with any optional properties.

Returns:

a_plot: A object of plot_abstract or one of its subclasses.

See also: `plot_abstract` (p. 180), `plot_simple` (p. 193)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.50.10 Method `stats_db/plot_bars`

Summary: Creates a bar graph with errorbars for each db column.

Usage:

```
a_plot = plot_bars(a_stats_db, title_str, props)
```

Description: Looks for 'min', 'max', and 'STD' labels in the row_idx for drawing the errorbars. Each page of the DB will produce grouped bars.

Parameters:

`a_stats_db`**:** A stats_db object.

`title_str`**:** The plot title.

`props`**:** A structure with any optional properties.

`pageVariable`**:** The column used for denoting page values.

`axis_limits`**:** Passed as argument to plot_bars/plot_bars.
(passed to plot_bars/plot_bars)

Returns:

a_plot: A object of plot_bars or one of its subclasses.

See also: plot_abstract (p. 180), plot_bars/plot_bars (p. 187)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.50.11 Method stats_db/plot_bars_ax

Summary: Bar plot with extending errorbars for all columns in the same axis.

Usage:

```
a_plot = plot_bars_ax(a_tests_db, row, props)
```

Description: Differs from stats_db/plot_bars because it does not open a new axis for each column. This is only suitable if all columns have similar extents.

Parameters:

a_stats_db: A stats_db object.

title_str: Optional title string.

props: A structure with any optional properties.

putLabels: Put special column name labels.

Returns:

a_plot: A plot_abstract object that can be plotted.

See also: plot_abstract (p. 180), plotFigure (p. **??**), stats_db/plot_bars (p. 244)

Author: Cengiz Gunay <cgunay@emory.edu>, 2008/04/17

A.50.12 Method stats_db/set

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.50.13 Method stats_db/subsref

Summary: Defines generic indexing for objects.

A.51 Class `tests_3D_db`

A.51.1 Constructor `tests_3D_db/tests_3D_db`

Summary: A database multiple pages with rows of test columns. Each page may represent aspects of the data that are different, but not defined in this object.

Usage:

```
a_3D_db = tests_3D_db(data, col_names, row_names, page_names, id, props)
```

Description: This is a subclass of tests_db. Usually it contains a RowIndex column that points to an original db from which this data originated. The row indices can be used to reach the values associated with different pages of information contained in this object.

Parameters:

`data`: The 3-d vector of rows, columns, and pages.

`col_names`: Colun names of the database.

`id`: An identifying string.

`props`: A structure with any optional properties.

`invarName`: Name of the invariant parameter for this db.

Returns a structure object with the following fields:

tests_db, page_idx.

See also: `tests_db` (p. 256), `tests_db/invarValues` (p. 275)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/30

A.51.2 Method `tests_3D_db/addPages`

Summary: Inserts new pages (third dimension) to a tests_3D_db object.

Description: Adds new third dimension pages to the database and returns the new DB. Usage 2 concatanates two DBs pagewise. This operation is expensive in the sense that the whole database matrix needs to be enlarged just to add a single new page. The method of allocating a matrix, filling it up, and then providing it to the tests_db constructor is the preferred method of creating tests_db objects. This method may be used for measures obtained by operating on raw measures.

Parameters:

`obj, b_obj`: A tests_db object.

`page_names`: A single string or a cell array of page names to be added.

`page_data`: Data matrix of pages to be added.

Returns:

obj: The tests_db object that includes the new columns.

See also: tests_db (p. 256)

Author: Cengiz Gunay <cengique@users.sf.net>, 2017/06/08

A.51.3 Method tests_3D_db/corrCoefs

Summary: Calculates correlation coefficients by comparing col1 with other cols.

Usage:

```
a_coefs_db = corrCoefs(db, col1, cols, props)
```

Description: If db has multiple pages, then each page in db produces a row of coefficients and matching PageIndex. Assuming the db was created with invarValues, this function finds the invariant correlation coefficients between its columns. The invariant correlation coefficients are the correlation of one column value with another column value when some other column values are fixed. Since there are many occurences of the invariant coefficients, a histogram can then be created and returned from the created db. The other columns that are fixed are not in this db object, but can be reached using the indices to the original db. The page number is saved in the created db, so that it can be used to find the page from which the coefficient came. Then row indices of the page points to original constant column values.

Parameters:

`db`: A tests_db object.

`col1`: Column to compare.

`cols`: Columns to be compared with col1.

`props`: A structure with any optional properties.

`skipCoefs`: If 1, coefficients of less confidence than will be skipped. (default=1)

Returns:

a_coefs_db: A corrcoefs_db of the coefficients and page indices.

See also: tests_db (p. 256), corrcoefs_db (p. 83)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/30

A.51.4 Method `tests_3D_db/diff2D`

Summary: Creates a tests_db by taking the derivative of the given test.

Usage:

```
a_tests_db = diff2D(a_db, test, props)
```

Description: Applies the diff function to the chosen test, and collapses the middle dimension of the 3D DB to create a 2D DB and transposes it. The result is that the pages of the 3D DB becomes the rows of the new database, and the differenced rows appear as new columns, each named uniquely. The column index would correspons to the row index in the 3D DB. A new column 'PageNumber' is appended to point back to the 3D DB.

Parameters:

`a_db`: A tests_3D_db object.

`test`: Test column.

`props`: Optional properties.

Returns:

a_tests_db: A tests_db that holds the requested differences of parameter values.

See also: `boxplot` (p. **??**), `plot_abstract` (p. 180)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/05/22

A.51.5 Method `tests_3D_db/display`

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.51.6 Method `tests_3D_db/flattenPages`

Summary: Convert from 3D to 2D by simply flattening pages.

Usage:

```
a_db = flattenPages(a_db)
```

Description: Allows to get the original database after doing invarValues or invarParams and then using joinRows.

Parameters:

`a_db`: A tests_3D_db object.

Returns:

a_db: A tests_db object.

See also: `tests_db/invarValues` (p. 275), `params_tests_db/invarParams` (p. 144), `tests_db/joinRows` (p. 278)

Author: Cengiz Gunay <cgunay@emory.edu>, 2014/05/07

A.51.7 Method `tests_3D_db/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.51.8 Method `tests_3D_db/histograms`

Usage:

```
a_histogram_db = histogram(db, col, num_bins)
```

Description: If one wants to get histograms of test values for each single value of the selected invariant parameter, then swapRowsPages should be done first on db.

Parameters:

`db`: A tests_3D_db object.

`col`: Column to find the histogram.

`num_bins`: Number of histogram bins (Optional, default=100)

Returns:

a_histogram_db: A histogram_db object containing the histogram.

See also: `histogram_db` (p. 101), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/04

A.51.9 Method `tests_3D_db/joinPages`

Summary: Joins the rows of the given db to the with_db rows matching with the PageIndex column.

Usage:

```
a_db = joinPages(db, with_db)
```

Description: Replicates the desired columns in the with_db with rows having a page index and joins them next to desired columns from the current 3D_db. Flattens the resulting 3D_db to become a 2D db. Assumes each page index only appears once in with_db.

Parameters:

`db`: A tests_3D_db object.

`with_db`: A tests_db object with a PageIndex column.

Returns:

a_db: A tests_db object.

See also: `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/15

A.51.10 Method `tests_3D_db/mergePages`

Summary: Merges tests from separate pages into a 2D params_tests_db.

Usage:

```
a_db = mergePages(db, page_tests, page_suffixes)
```

Description: Keeps uniqueness by adding suffixes to test names. If you're using invarParams, do swapRowsPages, then use joinRows with original db to get the parameter values.

Parameters:

`db`: A tests_3D_db object.

`page_tests`: Cell array of list of tests to take from each page.

`page_suffixes`: Cell array of suffixes to append to tests from each page.

Returns:

a_db: A tests_db object.

See also: `tests_db` (p. 256), `tests_3D_db` (p. 246), `tests_db/joinRows` (p. 278)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/01/13

A.51.11 Method `tests_3D_db/onlyRowsTests`

Summary: Returns a tests_db that only contains the desired tests and rows (and pages).

Usage:

```
obj = onlyRowsTests(obj, rows, tests, pages)
```

Description: Selects the given dimensions and returns in a new tests_db object.

Parameters:

`obj`: A tests_db object.

`rows, tests`: A logical or index vector of rows, or cell array of names of rows. If ':', all rows. For names, regular expressions are supported if quoted with slashes (e.g., '/a.*/'). See tests2idx.

`pages`: (Optional) A logical or index vector of pages. ':' for all pages.

Returns:

obj: The new tests_db object.

See also: `subsref` (p. **??**), `tests_db` (p. 256), `tests2idx` (p. **??**), `regexp` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/17

A.51.12 Method `tests_3D_db/paramsTestsHistsStats`

Summary: Calculates histograms and statistics for DB.

Usage:

```
[pt_hists, p_stats] = paramsTestsHistsStats(p_t3ds, props)
```

Description: Calculates histograms and statistics for all combinations of tests and params and returns them in a cell array. Skips the 'ItemIndex' test.

Parameters:

`p_t3ds`: Array of invariant parameter databases obtained by calling the params_tests_db/invarParams method.

`props`: Optional properties.

`statsMethod`: method to call to get a stats_db (default='statsMeanSE')

`useDiff`: If 1, takes the derivative with diff on the 3D DBs (default=0).

Returns:

pt_hists: An array of 3D histograms for each pair of param and test. p_stats: An array of stats_dbs for each param.

See also: `invarParams` (p. **??**), `params_tests_profile` (p. 162)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/17

A.51.13 Method `tests_3D_db/plotParamPairImage`

Summary: Generates an image plot of variation of a test with two parameters in the first page.

Usage:

```
a_plot = plotParamPairImage(a_db, test, title_str, props)
```

Description: It is assumed that the 3D DB is created by invariant combinations of two parameters, which are the first two columns. Each page of the db must contain a same parameter values. This is the default character of tests_3D_db created by params_tests_db/invarParam. Parameter values will be enumerated and then an image plot is created.

Parameters:

`a_db`: A tests_3D_db object.

`test`: Test column to take the measure value.

`title_str`: (Optional) String to append to plot title.

`props`: Optional properties to be passed to plot_abstract.

`truncateDecDigits`: Truncate labels to this many decimal digits.

`labelSteps`: Skip this many labels between ticks to reduce to total number.

`maxValue`: Maximal value to normalize colors and to annotate the colorbar.

Returns:

a_plot: A plot_abstract object or one of its subclasses.

Example:

```
Find relationship of two parameters against a measure:
» plotFigure(plotParamPairImage(invarParam(a_db, 'NaF', 'KCNQ'),
'PulseIni100msRest2SpikeRateISI_D100pA'));
```

See also: `params_tests_db/invarParam` (p. 144), `plotImage` (p. **??**), `plot_abstract.` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/10

A.51.14 Method `tests_3D_db/plotScatter`

Summary: Superpose scatter plots for each page of the database of the given two tests.

Usage:

```
a_p = plotScatter(a_db, test1, test2, title_str, short_title, props)
```

Description: If 'warning on verbose' is issued before this, it will display regression statistics: R^2, F, p, and the error variance.

Parameters:

`a_db`: A tests_3D_db object.

`test1, test2`: X & Y variables.

`title_str`: (Optional) A string to be concatenated to the title.

`short_title`: (Optional) Few words that may appear in legends of multiplot.

`props`: A structure with any optional properties.

`LineStyle`: Plot line style to use. (default: 'x')

`Regress`: If exists, use these props for plotting the linear regression.

`quiet`: If 1, don't include database name on title.

(all passed to tests_db/plotScatter)

Returns:

a_p: A plot_abstract.

See also: `tests_db/plotScatter` (p. 291)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/09/29

A.51.15 Method `tests_3D_db/plotVarBox`

Summary: Generates a boxplot of the variation between two tests.

Usage:

```
a_plot = plotVarBox(a_db, test1, test2, notch, sym, vert, whis, props)
```

Description: It is assumed that each page of the db contains a different parameter value.

Parameters:

`a_db`: A tests_3D_db object.

`test1`: Test column for the x-axis, only mean values are used.

`test2`: Test column for the y-axis, used for boxplot.

`notch, sym, vert, whis`: See boxplot, defaults = (1, '+', 1, 1.5).

`props`: Optional properties to be passed to plot_abstract.

Returns:

a_plot: A plot_abstract object or one of its subclasses.

See also: `boxplot` (p. **??**), `plot_abstract` (p. 180)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/10

A.51.16 Method `tests_3D_db/renamePages`

Summary: Rename one or more existing pages.

Usage:

```
a_db = renamePages(a_db, old_names, new_names)
```

Description: This is a cheap operation than modifies meta-data kept in object. For the regular expression renaming, the old_names and new_names parameters are passed to the regexprep command after removing the delimiting slashes (//). At least one grouping construct ('()') must be used in the search pattern such that it can be used in the replacement pattern (e.g., '$1'). See example above. This function uses the generic renameIdx that can work on row, column, or page indices.

Parameters:

`a_db`: A tests_db object.

`old_names`: A cell array of existing names, array of numerical indices, or a regular
expression denoted between slashes (e.g., '/(.*)/').

`new_names`: New names to replace existing ones OR regular expression
replace string (no slashes, e.g, '$1_test'). See regexprep command.

Returns:

a_db: The tests_db object that includes the new pages.

Example:

```
» new_db = renamePages(a_db, 'PulseIni100msSpikeRateISI_D40pA', 'Firing_rate');
» new_db = renamePages(a_db, 1, 'Firing_rate');
» new_db = renamePages(a_db, '/(.*)/', '$1_old');
» new_db = renamePages(a_db, 'a', 'b', 'c', 'd');
```

See also: `renameIdx` (p. **??**), `regexprep` (p. **??**), `allocateRows` (p. **??**), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2017/06/09

A.51.17 Method `tests_3D_db/set`

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.51.18 Method `tests_3D_db/swapColsPages`

Summary: Swaps the column dimension with the page dimension of the tests_3D_db.

Usage:

```
a_3D_db = swapColsPages(db)
```

Parameters:

`db`: A tests_db object.

Returns:

a_3D_db: A tests_3D_db object.

See also: `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2017/06/09

A.51.19 Method `tests_3D_db/swapRowsPages`

Summary: Swaps the row dimension with the page dimension of the tests_3D_db.

Usage:

```
a_3D_db = swapRowsPages(db)
```

Description: Assuming that this is a invariant parameter and tests relations db, this function transposes the data matrix by swapping the pages with rows. Each resulting page correspond to a single value of the chosen parameter, with each row containing a test result with different combinations of the rest of the parameters.

Parameters:

`db`: A tests_db object.

Returns:

a_3D_db: A tests_3D_db object.

See also: `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/04

A.52 Class `tests_db`

A.52.1 Constructor `tests_db/tests_db`

Summary: Construct a numeric database organized in a matrix format.

Usage:

```
obj = tests_db(test_results, col_names, row_names, id, props)
```

Description: This is the base database class. Note for loading text files: Matlab's dlmread commands is used, and it is unable to handle files that have any non-numeric data (except skipped rows). Therefore, those files are best filtered with outside tools before importing.

Parameters:

`test_results`: Either a text file (e.g., CSV) name or a matrix that contains measurement columns and separate observations as rows.

`col_names`: Cell array of column names of test_results.

`row_names`: Cell array of row names of test_results.

`id`: An identifying string.

`props`: A structure with any optional properties.

`textDelim`: Delimiter in text file to be used by dlmread (Default: ',' for CSV). Use '' for all whitespace, '\t' for tabs.

`csvArgs`: Cell array of arguments passed to dlmread function (e.g., R, C, [r1 c1 r2 c2]) for (R)ow and (C)olumn offset/range to read.

`csvReadColNames`: If 1, first row of the file or at the given offset (see csvArgs) is used to read column names. Double quotes must be used consistently.

`paramDescFile`: Load parameter names from file (one line per name).

Returns a structure object with the following fields:

data: The data matrix. row_idx, col_idx: Structure associating row/column names to indices. id, props.

See also: `params_tests_db` (p. 138), `dlmread` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/01

A.52.2 Method `tests_db/abs`

Summary: Take absolute value of all db elements.

Usage:

```
a_db = abs(left_obj)
```

Parameters:

`left_obj`: A tests_db object.

Returns:

a_db: The resulting tests_db.

See also: `abs` (p. **??**), `uop` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2008/01/16

A.52.3 Method `tests_db/addColumns`

Summary: Inserts new columns to tests_db.

Description: Adds new test columns to the database and returns the new DB. Usage 2 concatanates two DBs columnwise. This operation is expensive in the sense that the whole database matrix needs to be enlarged just to add a single new column. The method of allocating a matrix, filling it up, and then providing it to the tests_db constructor is the preferred method of creating tests_db objects. This method may be used for measures obtained by operating on raw measures.

Parameters:

`obj, b_obj`: A tests_db object.

`test_names`: A single string or a cell array of test names to be added.

`test_columns`: Data matrix of columns to be added.

Returns:

obj: The tests_db object that includes the new columns.

See also: `allocateRows` (p. **??**), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/09/30

A.52.4 Method `tests_db/addLastRow`

Summary: Inserts a row of observations at the end of tests_db.

Usage:

```
index = addLastRow(obj, row)
```

Description: Adds a new set of observations to the database and returns its row index. This operation is expensive because the whole database matrix needs to be duplicated and resized in order to add a single new row. The method of allocating a matrix, filling it up, and then providing it to the tests_db constructor is the preferred method of creating tests_db objects.

Parameters:

`obj`: A tests_db object.

`row`: A row vector that contains values for each DB column.

Returns:

obj: The tests_db object that includes the new row.

See also: `allocateRows` (p. **??**), `addRow` (p. **??**), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/08

A.52.5 Method `tests_db/addPages`

Summary: Inserts new pages (third dimension) to a tests_db object.

Description: Adds new third dimension pages to the database and returns the new DB. Page names are not maintained in tests_db; use tests_3D_db instead. Usage 2 concatanates two DBs pagewise. This operation is expensive in the sense that the whole database matrix needs to be enlarged just to add a single new page. The method of allocating a matrix, filling it up, and then providing it to the tests_db constructor is the preferred method of creating tests_db objects. This method may be used for measures obtained by operating on raw measures.

Parameters:

`obj, b_obj`: A tests_db object.

`page_names`: A single string or a cell array of page names to be added. IGNORED in tests_db.

`page_data`: Data matrix of pages to be added.

Returns:

obj: The tests_db object that includes the new columns.

See also: `tests_db` (p. 256)

Author: Cengiz Gunay <cengique@users.sf.net>, 2017/06/08

A.52.6 Method `tests_db/addRow`

Summary: Inserts a row of observations to tests_db at the given row index.

Usage:

```
index = addRow(obj, row, index)
```

Description: Adds a new set of observations to the database and returns the new DB. This operation is expensive in the sense that the whole database matrix needs to be copied to be passed to this function just to add a single new row. The method of allocating a matrix, filling it up, and then providing it to the tests_db constructor is the preferred method of creating tests_db objects.

Parameters:

`obj`: A tests_db object.

`row`: A row vector that contains values for each DB column.

`index`: The row index.

Returns:

obj: The tests_db object that includes the new row.

See also: `addLastRow` (p. **??**), `allocateRows` (p. **??**), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/08

A.52.7 Method `tests_db/allocateRows`

Summary: Preallocates a NaN-filled num_rows rows in tests_db.

Usage:

```
obj = allocateRows(obj, num_rows)
```

Description: Overwrites the database by allocating a new matrix of the desired number of rows to speed up filling up the data matrix using assignRowsTests. Using addRow after this operation is still expensive. The method of allocating a matrix, filling it up, and then providing it to the tests_db constructor is the preferred method of creating tests_db objects.

Parameters:

`obj`: A tests_db object.

`num_rows`: The predicted number of observations for this tests_db.

Returns:

obj: The new tests_db object.

See also: `assignRowsTests` (p. **??**), `addRow` (p. **??**), `setRows` (p. **??**), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/08

A.52.8 Method `tests_db/anyRows`

Summary: Returns db rows matching any of the given rows.

Usage:

```
idx = anyRows(db, rows)
```

Description: The db rows are compared to each row and row indices succeeding any of these comparisons are returned.

Parameters:

`db`: A tests_db object.

`rows`: Row array, matrix or database to be compared with db rows.

Returns:

idx: A logical column vector of matching db row indices. rows_idx: Indices of rows entries corresponding to each db row. Non-matching entries were left as NaN.

Example:

```
» db(anyRows(db(:, 'trial'), [12; 46; 37]), :)
returns a db with rows having trial equal to any of the given values.
```

See also: `compareRows` (p. **??**), `eq` (p. **??**), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/17

A.52.9 Method `tests_db/approxMappingLIBSVM`

Summary: Approximates the desired input-output mapping using a support vector machine (SVM).

Usage:

```
[an_approx_db, an_svm] = approxMappingLIBSVM(a_db, input_cols, output_cols, props)
```

Description: Uses the LIBSVM package (http://www.csie.ntu.edu.tw/ cjlin/libsvm/). If 'warning on verbose' is issued prior to running, it provides additional debug info.

Parameters:

`a_db`: A tests_db object.

`input_cols, output_cols`: Input and output columns to be mapped (see tests2cols for accept column specifications).

`props`: A structure with any optional properties.

`classProbs`: 'prob': use probabilistic sampling to normalize prior class probabilities.

`kernel`: Kernel type (default='poly').

`crossFold`: n to use in libsvm's n-fold cross-validation (default=0; disabled).

`testControl`: Ratio of dataset to train the data and rest to test for success (default=0; disabled).

`svmCost`: Add the -c option to svmOpts with this value.

`svmGamma`: Add the -g option to svmOpts with this value.

`svmOpts`: Passed to LIBSVM overwriting default options (see output of svm-train for options; default='-s0 -t2 -d2'). (Rest passed to balanceInputProbs and tests_db)

Returns:

an_approx_db: A tests_db object containing the original inputs and the approximated outputs. an_svm: The Matlab neural network approximator object.

Example:

```
» [a_class_db, an_svm = approxMappingLIBSVM(my_db, 'NaF', 'Kv3', 'spike_width');
» plotFigure(plot_superpose(plotScatter(my_db, 'NaF', 'spike_width'),
plotScatter(a_class_db, 'NaF', 'spike_width')))
```

See also: `tests_db` (p. 256), `newff` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2007/12/12

A.52.10 Method `tests_db/approxMappingNNet`

Summary: Approximates the desired input-output mapping using a Matlab neural network.

Usage:

```
[an_approx_db, a_nnet] = approxMappingNNet(a_db, input_cols, output_cols, props)
```

Description: Approximates the mapping between the given inputs to outputs using the Matlab Neural Network Toolbox. By default it creates a feed-forward network to be trained with a Levenberg-Marquardt training algorithm (see newff). Returns and the trained network object and a database with output columns obtained from the approximator. The outputs can then be compared to the original database to test the success of the approximation. If 'warning on verbose' is issued prior to running, it provides additional debug info.

Parameters:

`a_db`: A tests_db object.

`input_cols, output_cols`: Input and output columns to be mapped (see tests2cols for accept column specifications).

`props`: A structure with any optional properties.

`nnetFcn`: Neural network classifier function (default='newff')

`nnetParams`: Cell array of parameters passed to nnetFcn after inputs and outputs.

`trainMode`: 'batch' or 'incr'.

`testControl`: Ratio of dataset to train the data and rest to test for success (default=0; disabled).

`classProbs`: 'prob': use probabilistic sampling to normalize prior class probabilities.

`maxEpochs`: maximum number of epochs to train for. (Rest passed to balanceInputProbs and tests_db)

Returns:

an_approx_db: A tests_db object containing the original inputs and the approximated outputs. a_nnet: The Matlab neural network approximator object.

Example:

```
» [a_class_db, a_nnet = approxMappingNNet(my_db, 'NaF', 'Kv3', 'spike_width');
» plotFigure(plot_superpose(plotScatter(my_db, 'NaF', 'spike_width'),
plotScatter(a_class_db, 'NaF', 'spike_width')))
```

See also: `tests_db` (p. 256), `newff` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2007/12/12

A.52.11 Method `tests_db/approxMappingSVM`

Summary: Approximates the desired input-output mapping using a support vector machine (SVM).

Usage:

```
[an_approx_db, an_svm] = approxMappingSVM(a_db, input_cols, output_cols, props)
```

Description: Uses the SVM-KM package (http://asi.insa-rouen.fr/enseignants/ arakotom/toolbox/index.html). If 'warning on verbose' is issued prior to running, it provides additional debug info.

Parameters:

`a_db`: A tests_db object.

`input_cols, output_cols`: Input and output columns to be mapped (see tests2cols for accept column specifications).

`props`: A structure with any optional properties.

`classProbs`: 'prob': use probabilistic sampling to normalize prior class probabilities.

`kernel`: Kernel type (default='poly').

(Rest passed to balanceInputProbs and tests_db)

Returns:

an_approx_db: A tests_db object containing the original inputs and the approximated outputs. an_svm: The Matlab neural network approximator object.

Example:

```
» [a_class_db, an_svm = approxMappingSVM(my_db, 'NaF', 'Kv3', 'spike_width');
» plotFigure(plot_superpose(plotScatter(my_db, 'NaF', 'spike_width'),
plotScatter(a_class_db, 'NaF', 'spike_width')))
```

See also: `tests_db` (p. 256), `newff` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2007/12/12

A.52.12 Method `tests_db/assignRowsTests`

Summary: Assign the values to the tests and rows (and pages) of the tests_db.

Usage:

```
obj = assignRowsTests(obj, val, rows, tests, pages)
```

Description: Selects the given dimensions and returns in a new tests_db object.

Parameters:

`obj`: A tests_db object.

`val`: DB object or data matrix to be assigned to the addressed indices.

`rows`: A logical or index vector of rows. If ':', all rows.

`tests`: Cell array of test names or column indices. If ':', all tests.

`pages`: (Optional) A logical or index vector of pages. ':' for all pages.

Returns:

obj: The new tests_db object.

See also: `subsref` (p. **??**), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/02/08

A.52.13 Method `tests_db/checkConsistentCols`

Summary: Check if two DBs have exactly the same columns.

Usage:

```
[col_names, with_col_names] = checkConsistentCols(db, with_db, props)
```

Parameters:

`db`: A tests_db object.

`with_db`: A tests_db object whose column names are checked for consistency.

`props`: A structure with any optional properties.

`useCommon`: Tolerate mismatching column names and only return the common columns.

Returns:

col_names, with_col_names: list of column names of each DB.

See also: `vertcat` (p. **??**), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2007/01/18

A.52.14 Method `tests_db/compareRows`

Summary: Returns differing rows of db and the given row(s).

Usage:

```
[idx, compared] = compareRows(db, rows)
```

Description: It can compare all db rows to corresponding row entries or to a single row. For the case with only one entry, returns all db rows that do not match the given row in idx, and the result of the differences in compared. For the case of multiple rows, rows must have exactly same number of rows with db. In both cases, idx must be negated to test for equality.

Parameters:

`db`: A tests_db object.

`rows`: Row array, matrix or database to be compared with db rows.

Returns:

idx: A inverted logical column vector of comparison results. (false if db == rows, true otherwise) compared: A column vector of differences of each DB row to the given row (i.e., compared = db - rows).

Example:

```
» db(db(:, 'trial') > [12]), :)
calls gt which calls compareRows to check for equality.  Returns a db
only containing rows with trial numbers greater than 12.
```

See also: eq (p. **??**), anyRows (p. **??**), ge (p. **??**), gt (p. **??**), le (p. **??**), lt (p. **??**), tests_db (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/17

A.52.15 Method tests_db/corrcoef

Summary: Calculates a correlation coefficient matrix by comparing cols.

Usage:

```
a_coefs_db = corrcoef(db, cols, props)
```

Parameters:

`db`: A tests_db object.

`cols`: Columns to be compared.

`props`: A structure with any optional properties.

- `skipCoefs`: If 1, coefficients of less confidence than will be skipped. (default=1)
- `alpha`: Skip coefs with p values lower than this (default=0.05).
- `partialCols`: Columns to calculate partial correlations by controlling for the other columns.
- `bonfer`: Bonferroni correction to alpha value.
- `resample`: Shuffle columns and resample correlations this many times to get statistics for the null hypothesis.

Returns:

a_coefs_db: A tests_3D_db of the coefficient matrix, and their upper/lower limits on different pages.

See also: tests_db (p. 256), corrcoefs_db (p. 83), corrcoef (p. **??**), partialcorr (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2008/04/25

A.52.16 Method `tests_db/cov`

Summary: Generates a database of the covariance of given DB.

Usage:

```
a_cov_db = cov(db, props)
```

Parameters:

`db`: A tests_db object.

`props`: A structure with any optional properties.

`keepOrigDB`: Keep db as origDB in the props.
(others passed to tests_db)

Returns:

a_cov_db: A tests_db which contains the covariance matrix.

See also: `cov` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2007/05/25

A.52.17 Method `tests_db/crossProd`

Summary: Create a DB by taking the cross product of two database row sets.

Usage:

```
cross_db = crossProd(a_db, b_db)
```

Description: This is not a vector cross product operation. Each row of the two DBs are matched and added as a new row to a DB. The end is a DB with all combinations of rows from both DBs. The final DB contains columns of both DBs.

Parameters:

`a_db, b_db`: A tests_db object.

Returns:

cross_db: The tests_db object with all combinations of rows.

See also: `allocateRows` (p. **??**), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/10/11

A.52.18 Method `tests_db/dbsize`

Summary: Returns the size of the data matrix of db.

Usage:

```
s = dbsize(db)
```

Parameters:

`db`: A tests_db object.

Returns:

s: The size values.

See also: `size` (p. **??**), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/06

A.52.19 Method `tests_db/delColumns`

Summary: Deletes columns from tests_db.

Usage:

```
obj = delColumns(obj, tests)
```

Description: Deletes test columns from the database and returns the new DB. This operation is expensive in the sense that the whole database matrix needs to be copied just to delete a single column. The method of allocating a matrix, filling it up, and then providing it to the tests_db constructor is the preferred method of creating tests_db objects. This method may be used for measures obtained by operating on raw measures.

Parameters:

`obj`: A tests_db object.

`tests`: Numbers or names of tests (see tests2cols)

Returns:

obj: The tests_db object that is missing the columns.

See also: `allocateRows` (p. **??**), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/10/06

A.52.20 Method `tests_db/diff`

Summary: Creates a tests_db by taking the derivative of all tests.

Usage:

```
a_db = diff(a_db, props)
```

Description: Applies the diff function to whole DB. The resulting DB will have one less row.

Parameters:

`a_db`: A tests_db object.

`props`: Optional properties.

Returns:

a_db: The resulting tests_db.

See also: `diff` (p. **??**), `tests_3D_db/getDiff2DDB` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/05/24

A.52.21 Method `tests_db/display`

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.52.22 Method `tests_db/displayRows`

Summary: Displays rows of data with associated column labels.

Usage:

```
s = displayRows(db, rows, pages)
```

Description: Use transpose() on db to rotate display.

Parameters:

`db`: A tests_db object.

`rows`: Indices of rows in db.

`pages`: Pages of db.

Returns:

s: A cell array of trasposed database contents, prefixed with column names on each row. Meant to be displayed on the screen.

See also: `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/15

A.52.23 Method `tests_db/displayRowsCSV`

Summary: Returns a comma-separated values (CSV) version of the table.

Usage:

```
csv_string = displayRowsCSV(a_db, props)
```

Description: Uses displayRows. See its documentation for details.

Parameters:

`a_db`: A tests_db object.

`props`: A structure with any optional properties.

Returns:

csv_string: String that can be saved as a file or copy-pasted into other software.

Example:

```
» string2File(displayRowsCSV(a_db(1:10, 4:7)), 'excel-export.csv')
```

See also: displayRows (p. **??**), displayRowsTeX (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2011/07/07

A.52.24 Method `tests_db/displayRowsTeX`

Summary: Generates a LaTeX table that lists rows of this DB.

Usage:

```
tex_string = displayRowsTeX(a_db, caption, props)
```

Description: Can be then written to a file for processing by Latex or inclusion in other documents.

Parameters:

`a_db`: A tests_db object.

`caption`: Table caption.

`props`: A structure with any optional properties, passed to TeXfloat.

`landscape`: If 1, rotate table 90 degrees and scale to 90

Returns:

tex_string: LaTeX string for table float.

Example:

```
» string2File(displayRowsTeX(a_db(1:10, 4:7), 'some values',
struct('rotate', 0)), 'table.tex')
```

See also: displayRows (p. **??**), TeXfloat (p. **??**), cell2TeX (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/12/16

A.52.25 Method tests_db/end

Summary: Overloaded primitive matlab function, returns maximal dimension size.

Usage:

```
s = end(db, index, total)
```

Parameters:

db: A tests_db object.

Returns:

s: The size.

See also: size (p. **??**), tests_db (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/06

A.52.26 Method tests_db/enumerateColumns

Summary: Replaces each value with an integer pointing to the index of enumerated unique values in a column.

Usage:

```
a_db = enumerateColumns(a_db, tests, props)
```

Description: Finds unique values of each column, and replaces the original values with the enumerated indices of these unique values. Useful for normalizing all parameter values in a hypercube.

Parameters:

a_db: A tests_db object.

tests: Array of tests to be enumerated.

props: Optional properties.

truncateDecDigits: Use only up to this many decimal digits after the point
when checking for uniqueness.

Returns:

a_db: The modified DB.

Example:

```
» enumerated_db = enumerateColumns(a_db(:, 1:9));
```

See also: uniqueValues (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/06/14

A.52.27 Method tests_db/eq

Summary: Equality (==) operator. Returns logical indices of db rows that match with given row.

Usage:

```
rows = eq(db, row)
```

Parameters:

`db`: A tests_db object.

`row`: Row array to be compared with db rows.

Returns:

rows: A logical or index vector to be used in indexing db objects.

See also: eq (p. **??**), tests_db (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/17

A.52.28 Method tests_db/factoran

Summary: Generates a database of factor loadings obtained from the factor analysis of db with factoran. Each row corresponds to a rotated factor and columns represent observed variables.

Usage:

```
a_factors_db = factoran(db, num_factors, props)
```

Description: Uses the promax method to rotate common factors.

Parameters:

`db`: A tests_db object.

`num_factors`: Number of common factors to look for.

`props`: A structure with any optional properties.

Returns:

a_factors_db: A corrcoefs_db of the coefficients and page indices.

See also: `tests_db` (p. 256), `corrcoefs_db` (p. 83)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/08

A.52.29 Method `tests_db/fillMissingColumns`

Summary: Add missing columns with given default fill value.

Usage:

```
db = fillMissingColumns(db, col_names, fill_value)
```

Parameters:

`db`: A tests_db object.

`col_names`: A cell array of column names.

`fill_value`: Value to be used for missing columns.

Returns:

db: The tests_db object that includes the newly filled columns.

See also: `tests_db` (p. 256), `addColumns` (p. **??**), `params_tests_db/addParams` (p. 139), `params_tests_db/unionCat` (p. 154)

Author: Cengiz Gunay <cgunay@emory.edu>, 2008/06/02

A.52.30 Method `tests_db/ge`

Summary: Greater or equal to (>=) operator. Returns logical indices of db rows that are greater than or equal to given row.

Usage:

```
rows = ge(db, row)
```

Parameters:

`db`: A tests_db object.

`row`: Row array to be compared with db rows.

Returns:

rows: A logical or index vector to be used in indexing db objects.

See also: `ge` (p. **??**), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/17

A.52.31 Method `tests_db/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.52.32 Method `tests_db/getColNames`

Summary: Gets column names.

Usage:

```
col_names = getColNames(db, tests)
```

Description: Performs a light operation without touching the data matrix.

Parameters:

`db`: A tests_db object.

`tests`: Columns for which to get names (Optional, default = ':')

Returns:

col_names: A cell array of strings.

See also: `getColNames` (p. **??**), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/05/24

A.52.33 Method `tests_db/groupBy`

Summary: Groups same values of column(s) into separate pages of a 3D db.

Usage:

```
a_tests_3D_db = groupBy(db, cols)
```

Description: Functionality similar to SQL's GROUP BY keyword. This function uses invarValues, but resulting pages will not be sorted.

Parameters:

`db`: A tests_db object.

`cols`: Columns whose same values will be in one page (see tests2cols for column representation).

Returns:

a_tests_3D_db: A tests_3D_db object of organized values.

Example:

```
» a_db = tests_db([ ... ], 'par1', 'par2', 'measure1', 'measure2')
» a_3d_db = groupBy(a_db, 'par1')
»
» joined_3d_db = joinRows(a_db, a_3d_db)
» displayRows(joined_3d_db(:, :, 1))
```

See also: invarValues (p. **??**), tests_3D_db (p. 246), tests_3D_db/corrCoefs (p. 247), tests_3D_db/plotPair (p. **??**), joinRows (p. **??**), tests_3D_db/swapRowsPages (p. 255), tests_3D_db/mergePages (p. 250)

Author: Cengiz Gunay <cgunay@emory.edu>, 2008/05/27

A.52.34 Method tests_db/gt

Summary: Greater than (>) operator. Returns logical indices of db rows that are greater than given row.

Usage:

```
rows = gt(db, row)
```

Parameters:

`db`: A tests_db object.

`row`: Row array to be compared with db rows.

Returns:

rows: A logical or index vector to be used in indexing db objects.

See also: gt (p. **??**), tests_db (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/17

A.52.35 Method tests_db/histogram

Summary: Returns histogram of chosen database column.

Usage:

```
a_histogram_db = histogram(db, col, num_bins, props)
```

Description: Generates a histogram_db object with rows corresponding to histogram entries. If an array of DBs is given, finds and uses common histogram bin centers.

Parameters:

`db`: A tests_db object.

`col`: Column to find the histogram.

`num_bins`: Number of histogram bins (Optional, default=100), or vector of histogram bin centers.

`props`: A structure with any optional properties.

`normalized`: If 1, normalize histogram counts.

Returns:

a_histogram_db: A histogram_db object containing the histogram.

Example:

```
» a_hist_db = histogram(my_db, 'spike_width');
» plot(a_hist_db);
```

See also: `histogram_db` (p. 101), `tests_db` (p. 256), `hist` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/17

A.52.36 Method `tests_db/invarValues`

Summary: Finds all sets in which given columns vary while the rest are invariant.

Usage:

```
a_tests_3D_db = invarValues(db, cols, in_page_unique_cols, props)
```

Description: Useful when trying to find relationships between some columns independent of other columns. In a database that contains results of a multivariate function, this function can find the effect of one or more parameters when other parameters are kept constant (i.e., invariant). Rows with the values of the desired columns are separated into the pages of a tests_3D_db for each unique set of the other column values. These invariant values of the other columns are missing from the resulting tests_3D_db, instead a RowIndex is kept pointing to the db in which they can be found. See joinRows for joining the results back with the invariant columns. If in_page_unique_cols is given, this function by default row-sorts the database to ensure that each page has the same parameter values in the same rows. This is important because when the rows and pages of database is swapped (see tests_3D_db/swapRowsPages) each page has the same value of the in_page_unique_cols variables. Other functions such as tests_3D_db/mergePages also depend on this property. In databases that contain all unique combinations of certain parameters, the resulting 3D database becomes symmetric. However, for databases with missing combinations, in_page_unique_cols specifies which columns is used to guide which rows of the page to place values found. This function will fail if you do not have such a column. Note: the trial column will be ignored before finding invariant values.

Parameters:

`db`: A tests_db object.

`cols`: Vector of column numbers to find values when others are invariant. Include result columns here.

`in_page_unique_cols`: Vector of columns that have the same unique values in each page

(Optional; used only if database is not symmetric, to ignore missing values of in_page_unique_cols)

`props`: A structure with any optional properties.

`sortPages`: If 1, page-sorts even symmetric databases (default=1).

Returns:

a_tests_3D_db: A tests_3D_db object of organized values.

Example:

```
» a_db = tests_db([ ... ], 'par1', 'par2', 'measure1', 'measure2')
» a_3d_db = invarValues(a_db, [2:4], 'par2')
»
» joined_3d_db = joinRows(a_db, a_3d_db)
» displayRows(joined_3d_db(:, :, 1))
```

See also: `tests_3D_db` (p. 246), `tests_3D_db/corrCoefs` (p. 247), `tests_3D_db/plotPair` (p. **??**), `joinRows` (p. **??**), `tests_3D_db/swapRowsPages` (p. 255), `tests_3D_db/mergePages` (p. 250)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/30

A.52.37 Method `tests_db/isinf`

Summary: Returns logical row indices of Inf-valued columns.

Usage:

```
rows = isinf(db, col)
```

Parameters:

`db`: A tests_db object.

`col`: Column to check (Optional, default = 1)

Returns:

rows: A logical column vector of rows.

See also: `isinf` (p. **??**), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/08/16

A.52.38 Method `tests_db/isnan`

Summary: Returns logical row indices of NaN-valued columns.

Usage:

```
rows = isnan(db, col)
```

Parameters:

`db`: A tests_db object.

`col`: Column to check (Optional, default = 1)

Returns:

rows: A logical column vector of rows.

See also: `isnan` (p. **??**), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/06

A.52.39 Method `tests_db/isnanrows`

Summary: Finds rows with any NaN values. Returns logical indices of db rows.

Usage:

```
rows = isnanrows(db)
```

Description: Some operations need that no NaN values exist in the matrix. This method can be used to find and then remove NaN-contaminated rows from DB. Note that sometimes no rows can be found, and some columns should be discarded before this operation.

Parameters:

`db`: A tests_db object.

Returns:

rows: A logical vector to be used in indexing db objects or passed through other logical operators.

See also: `isnan` (p. **??**), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/08

A.52.40 Method `tests_db/joinRows`

Summary: Joins a_db rows with w_db rows having matching RowIndex values.

Usage:

```
a_db = joinRows(a_db, w_db, props)
```

Description: Concatenates columns of rows matching the join condition from the two databases. Each row index must appear only once in w_db. The created db preserves the ordering of w_db. See the multipleIndices option if there are several redundant index columns. Multiple pages in w_db are accepted (see keepNaNs option). This function is the equivalent of a "right outer join" command in SQL, w_db being the database table on the right.

Parameters:

`a_db`: A tests_db object.

`w_db`: A tests_db object with a row index column.

`props`: A structure with any optional properties.

`indexColName`: (Optional) Name of row index column (default='RowIndex').

`keepNaNs`: If 1, substitute NaN values for NaN indices. (default=1, for multi-page DBs; 0, otherwise).

`multipleIndices`: If 1, search for substitute RowIndex* columns for indices with NaN values. It will fail if all indices are NaNs. (default=0)

`keepIndex`: If 1, don't delete the indexColName (default='RowIndex')

Returns:

a_db: A tests_db object.

See also: `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/16

A.52.41 Method `tests_db/kmeansCluster`

Summary: Generates a database of cluster centers obtained from a k-means cluster analysis with the command kmeans.

Usage:

```
a_cluster_db = kmeansCluster(db, num_clusters, props)
```

Parameters:

`db`: A tests_db object.

`num_clusters`: Number of clusters to form.

`props`: A structure with any optional properties.

`DistanceMeasure`: Choose one appropriate for kmeans.

Returns:

a_cluster_db: A tests_db where each row is a cluster center. a_hist_db: histogram_db showing cluster membership from original db. idx: Cluster indices of each row or original db. sum_distances: Quality of clustering indicated by total distance from centroid for each cluster.

See also: `tests_db` (p. 256), `histogram_db` (p. 101)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/04/06

A.52.42 Method `tests_db/le`

Summary: Less or equal (<=) operator. Returns logical indices of db rows that are less than or equal to given row.

Usage:

```
rows = le(db, row)
```

Parameters:

`db`: A tests_db object.

`row`: Row array to be compared with db rows.

Returns:

rows: A logical or index vector to be used in indexing db objects.

See also: `le` (p. **??**), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/17

A.52.43 Method `tests_db/lt`

Summary: Less than (<) operator. Returns logical indices of db rows that are less than given row.

Usage:

```
rows = lt(db, row)
```

Parameters:

`db`: A tests_db object.

`row`: Row array to be compared with db rows.

Returns:

rows: A logical or index vector to be used in indexing db objects.

See also: `lt` (p. **??**), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/17

A.52.44 Method `tests_db/matchingRow`

Summary: Creates a criterion database for matching the tests of a row.

Usage:

```
crit_db = matchingRow(db, row, props)
```

Description: Copies selected test values from row as the first row into the criterion db. Adds a second row for the STD of each column in the db. Calculates the covariance for using the Mahalonobis distance in the ranking.

Parameters:

`db`: A tests_db object.

`row`: A row index to match.

`props`: A structure with any optional properties.

`distDB`: Take the standard deviation and covariance of this db instead.

Returns:

crit_db: A tests_db with two rows for values and STDs.

Example:

```
» crit_db = matchingRow(phys_control_compare_db,
find(phys_control_compare_db(:, 'TracesetIndex') == 61))
```

See also: `rankMatching` (p. **??**), `tests_db` (p. 256), `tests2cols` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/12/08

A.52.45 Method `tests_db/max`

Summary: Returns the max of the data matrix of a_db. Ignores NaN and Inf values.

Usage:

```
[a_db, n, i] = max(a_db, dim)
```

Description: Does a recursive operation over dimensions in order to remove NaN and Inf values. This takes more time than a straightforward max operation.

Parameters:

`a_db`**:** A tests_db object.

`dim`**:** Work down dimension.

Returns:

a_db: The DB with one row of max values. n: (Optional) Numbers of non-NaN rows included in calculating each column. i: Indices where the value was found.

See also: `max` (p. **??**), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/06

A.52.46 Method `tests_db/mean`

Summary: Returns the mean of the data matrix of a_db. Ignores NaN values.

Usage:

```
[a_db, n] = mean(a_db, dim)
```

Description: Does a recursive operation over dimensions in order to remove NaN values. This takes more time than a straightforward mean operation.

Parameters:

`a_db`**:** A tests_db object.

`dim`**:** Work down dimension.

Returns:

a_db: The DB with one row of mean values. n: (Optional) Numbers of non-NaN rows included in calculating each column.

See also: `mean` (p. **??**), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/06

A.52.47 Method `tests_db/meanDuplicateRows`

Summary: Row-reduces a db by finding sets of rows with same main_cols values, and replacing each set with a single row containing main_cols and the mean of rest_cols.

Usage:

```
a_tests_db = meanDuplicateRows(db, main_cols, rest_cols)
```

Description: The database is sorted for the values of the columns of interest (main_cols) and all rows with duplicate values of these columns are identified. The rest of the columns (rest_cols) are averaged and reduced to a single row, and attached to the nominal values of main_cols. Two additional parameter columns will be added to the database created. The NumDuplicates column is the the number of duplicates used in the mean operation, and RowIndex is the row number points to the first of a set of duplicate values.

Parameters:

`db`: A tests_db object.

`main_cols`: Vector of columns in which to find duplicates.

`rest_cols`: Vector of columns to be averaged for duplicate main_cols.

Returns:

a_tests_db: The db object of with the means on page 1 and standard deviations on page 2.

See also: `tests_db/mean` (p. 281), `tests_db/std` (p. 310), `sortedUniqueValues` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/30

A.52.48 Method `tests_db/min`

Summary: Returns the min of the data matrix of a_db. Ignores NaN and Inf values.

Usage:

```
[a_db, n, i] = min(a_db, dim)
```

Description: Does a recursive operation over dimensions in order to remove NaN and Inf values. This takes more time than a straightforward min operation.

Parameters:

`a_db`: A tests_db object.

`dim`: Work down dimension.

Returns:

a_db: The DB with one row of min values. n: (Optional) Numbers of non-NaN rows included in calculating each column. i: Indices where the value was found.

See also: `min` (p. **??**), `max` (p. **??**), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/06

A.52.49 Method `tests_db/minus`

Summary: Subtracts a DB from another or from a scalar.

Usage:

```
a_db = minus(left_obj, right_obj)
```

Description: If DBs have mismatching columns only the common columns will be kept. In any case, the resulting DB columns will be sorted in the order of the left-hand-side DB.

Parameters:

`left_obj, right_obj:` Operands of the subtraction. One must be of type tests_db
and the other can be a scalar.

Returns:

a_db: The resulting tests_db.

See also: `minus` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/05/24

A.52.50 Method `tests_db/mtimes`

Summary: Multiplies the DB with a scalar.

Usage:

```
a_db = mtimes(left_obj, right_obj)
```

Parameters:

`left_obj, right_obj:` Operands of the multiplication. One or more must be of type tests_db.

Returns:

a_db: The resulting tests_db.

See also: tests_db/times (p. 314), mtimes (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/05/24

A.52.51 Method tests_db/ne

Summary: Returns logical indices of db rows that doesn't match with given row.

Usage:

```
rows = ne(db, row)
```

Parameters:

db: A tests_db object.

row: Row array to be compared with db rows.

Returns:

rows: A logical or index vector to be used in indexing db objects.

See also: ne (p. **??**), tests_db (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/17

A.52.52 Method tests_db/noNaNRows

Summary: Returns a DB by removing rows containing any NaN or Inf.

Usage:

```
a_db = noNaNRows(a_db)
```

Parameters:

a_db: A tests_db object.

Returns:

a_db: DB with missing rows.

See also: tests_db/isnanrows (p. 277)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/09/21

A.52.53 Method `tests_db/onlyRowsTests`

Summary: Returns a tests_db that only contains the desired tests and rows (and pages).

Usage:

```
obj = onlyRowsTests(obj, rows, tests, pages)
```

Description: Selects the given dimensions and returns in a new tests_db object.

Parameters:

`obj`: A tests_db object.

`rows, tests`: A logical or index vector of rows, or cell array of names of rows. If ':', all rows. For names, regular expressions are supported if quoted with slashes (e.g., '/a.*/'). See tests2idx.

`pages`: (Optional) A logical or index vector of pages. ':' for all pages.

Returns:

obj: The new tests_db object.

See also: `subsref` (p. **??**), `tests_db` (p. 256), `tests2idx` (p. **??**), `regexp` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/17

A.52.54 Method `tests_db/physiol_bundle`

Summary: Create a physiol_bundle from a raw physiology database.

Usage:

```
a_pbundle = physiol_bundle(phys_dball, phys_dataset, props)
```

Description: Removes small bias currents, calculates input resistance by averaging negative CIP traces, averages multiple traces with similar treatments, selects certain CIP levels collapses its rows to create a one-neuron-per-pow database. It includes post-DB calculated columns such as rate ratios between spont and recovery periods.

Parameters:

`phys_dball`: A raw database obtained by loading traces from the tracesets.

`phys_dataset`: Dataset object passed to physiol_bundle.

`props`: Optional parameters.

`weedCols`: Cell array of parameter columns to be weed-out before averaging rows
that are same w.r.t other parameters. (default='pulseOn', 'pulseOff', 'traceEnd', 'pAbias', 'ItemIndex').

`drugCols`: Cell array of drug names that need to be zero for the control db (default='TTX', 'Apamin', 'EBIO', 'XE991', 'Cadmium', 'drug_4AP').

`CIPList`: row array specifying the CIP levels to choose (eliminate the others), default is an empty array, which means to choose all.

`biasLimit`: Limit in pA, biases larger +/- than which will be eliminated. (default=30)

Returns:

phys_joined_db: Final one row per cip and neuron db. phys_joined_control_db: Rows where all drug treatments are zero. phys_db: Original db only with parameter and including the weedCols.

See also: `physiol_bundle` (p. 164), `params_tests_db` (p. 138)

Author: Cengiz Gunay <cgunay@emory.edu>, 2007/12/21

A.52.55 Method `tests_db/plot`

Summary: Generic method to plot a tests_db or a subclass. Requires a plot_abstract method to be defined for this object.

Usage:

```
h = plot(a_tests_db, title_str, props)
```

Parameters:

`a_tests_db`: A histogram_db object.

`title_str`: (Optional) String to append to plot title.

`props`: A structure with any optional properties, passed to plot_abstract.

Returns:

h: The figure handle created.

See also: `plot_abstract` (p. 180), `plotFigure` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/06

A.52.56 Method `tests_db/plotBox`

Summary: Creates a boxplot from each column in tests_db in separate axes.

Usage:

```
a_plot = plotBox(a_tests_db, title_str, props)
```

Parameters:

`a_tests_db`: A tests_db object.

`title_str`: Optional title.

`props`: A structure with any optional properties.

`putLabels`: Put special column name labels.

`notch`: If 1, put notches on boxplots (default=1).

`whis`: Whisker size passed to boxplotp (default=1.5);

Returns:

a_plot: A plot_abstract object that can be plotted.

See also: `plot_abstract` (p. 180), `plotFigure` (p. **??**), `boxplotp` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2008/01/16

A.52.57 Method `tests_db/plotCircular`

Summary: Circular plot.

Usage:

```
a_p = plotCircular(a_db, theta_col, title_str, short_title, props)
```

Description: Radius is taken to be constant on the unit circle.

Parameters:

`a_db`: A tests_db object.

`theta_col`: Column with angle values to plot on circle.

`title_str`: (Optional) A string to be concatanated to the title.

`short_title`: (Optional) Few words that may appear in legends of multiplot.

`props`: A structure with any optional properties.

`avgVector`: If 1, plot an average vector from polar coordinates.

`vectorMarker`: Specify a marker for the average vector (default='.').

`radius`: The radius at which angles are plotted (default=1).

`angles1`: If 1, angles are in the range of 0-1, and they will be converted to radians.

`jitter`: Add this much random jitter to radius while plotting.

`quiet`: If 1, don't include database name on title.

Returns:

a_p: A plot_abstract.

See also: `polar` (p. **??**), `pol2cart` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2014/07/14

A.52.58 Method `tests_db/plotCovar`

Summary: Generates an image plot of the covariance-type values in a_db.

Usage:

```
[a_plot, shuffle_idx] = plotCovar(a_db, title_str, props)
```

Parameters:

`a_db`: A tests_db object that resulted from a function like cov.

`title_str`: (Optional) String to append to plot title.

`props`: Optional properties.

`inverse`: If 1, take inverse of the data matrix.

`corrcoef`: If 1, normalize matrix elements to get corrcoef values.

`logScale`: If 1, take logarithm of values before plotting.

`localityIters`: Apply a locality optimization algorithm with this many iterations. (rest passed to plot_image.)

Returns:

a_plot: A plot_abstract object or one of its subclasses. shuffle_idx: (Optional) Ordering of columns found by locality optimization.

Example:

```
» plotFigure(plotCovar(cov(get(constrainedMeasuresPreset(pbundle2, 6),
'joined_control_db'))));
```

See also: `tests_db/cov` (p. 266), `plotImage` (p. **??**), `tests_db/matchingRow` (p. 280), `corrcoefs.` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2007/05/30

A.52.59 Method `tests_db/plotImage`

Summary: Create an image plot of a measure changing with the given two parameters.

Usage:

```
a_p = plotImage(a_db, par1, par2, col, title_str, short_title, props)
```

Parameters:

`a_db`: A tests_db object.

`par1, par2`: X & Y variables.

`col`: Plot this column.

`title_str`: (Optional) A string to be concatanated to the title.

`short_title`: (Optional) Few words that may appear in legends of multiplot.

`props`: A structure with any optional properties.

`logScale`: If 1, take logarithm of values before plotting.

`quiet`: If 1, don't include database name on title.

Returns:

a_p: A plot_abstract.

See also: `plotScatter` (p. **??**), `plotScatter3D` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2009/02/17

A.52.60 Method `tests_db/plotParamsCoverage`

Summary: Lower triangle matrix of parameter-parameter combination scatter plots.

Usage:

```
a_pm = plotParamsCoverage(a_db, params, title_str, props)
```

Parameters:

`a_db`: A tests_db or params_tests_db object.

`params`: Columns to be used in the parameter coverage.

`title_str`: (Optional) A string to be concatanated to the title.

`props`: A structure with any optional properties, passed to plot_abstract.

`colorTest`: Use this column to specify colors of points (see plotScatter for other props to control behavior).

`quiet`: Don't put the DB id on the title.
(rest passed to plotScatter)

Returns:

a_pm: Resulting plot_stack object.

See also: plotScatter (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2016/05/27

A.52.61 Method tests_db/plotrow

Summary: Creates a plot_abstract describing the desired db row.

Usage:

```
a_plot = plotrow(a_tests_db, row, title_str, props)
```

Parameters:

a_tests_db: A tests_db object.

row: Row number to visualize.

title_str: Optional title string.

props: A structure with any optional properties.

putLabels: Put special column name labels.

Returns:

a_plot: A plot_abstract object that can be plotted.

See also: plot_abstract (p. 180), plotFigure (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/08

A.52.62 Method tests_db/plotrows

Summary: Creates a plot_stack describing the db rows.

Usage:

```
a_plot = plotrows(a_tests_db, axis_limits, orientation, props)
```

Parameters:

a_tests_db: A tests_db object.

axis_limits: If given, all plots contained will have these axis limits.

orientation: Stack orientation 'x' for horizontal, 'y' for vertical, etc.

title_str: Optional title string.

props: A structure with any optional properties passed to plot_stack.

Returns:

a_plot: A plot_stack object that can be plotted.

See also: `plot_abstract` (p. 180), `plotFigure` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/09

A.52.63 Method `tests_db/plotScatter`

Summary: Create a scatter plot of the given two tests.

Usage:

```
[a_p, b] = plotScatter(a_db, test1, test2, title_str, short_title, props)
```

Description: Newer versions of Matlab (e.g. R2014b) won't allow line styles in a scatter plot. To draw lines between points, one can switch to the default 'plot' function using the 'command' prop (see above) - but at the expense of losing the ability to use different colors for each point.

Parameters:

`a_db`: A tests_db object.

`test1, test2`: X & Y variables.

`title_str`: (Optional) A string to be concatanated to the title.

`short_title`: (Optional) Few words that may appear in legends of multiplot.

`props`: A structure with any optional properties.

- `LineStyle`: Plot line style to use. (default: 'x')
- `Regress`: If exists, use these as props for plotting linear
  - `regression and displays statistics`: R^2, F, p, and the error variance.
- `colorTest`: Use this column as index into colormap.
- `colormap`: Colormap vector, function name or handle to colormap (e.g., 'jet').
- `minValue, maxValue`: Set boundaries for the displayed colors (see plotColormap). Values outside of the boundaries will be truncated.
- `command`: Plot command to use (default: 'scatter'). If 'plot' is selected, colormap cannot be used.
- `numColors`: Number of colors desired in colormap (default: 50).
- `quiet`: If 1, don't include database name on title.
- `markerArea`: Passed as the 'area' argument to scatter (default=36). (Others passed to plotColormap and plot_abstract).

Returns:

a_p: A plot_abstract. b: A double holding the regression coefficient (optional)

See also: plotScatter3D (p. **??**), plotImage (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/09/29

A.52.64 Method tests_db/plotScatter3D

Summary: Create a 3D scatter plot of the given three tests.

Usage:

```
a_p = plotScatter3D(a_db, test1, test2, test3, title_str, short_title, props)
```

Parameters:

`a_db`: A params_tests_db object.

`test1, test2, test3`: X, Y, & Z variables.

`title_str`: (Optional) A string to be concatanated to the title.

`short_title`: (Optional) Few words that may appear in legends of multiplot.

`props`: A structure with any optional properties.

- `LineStyle`: Plot line style to use. (default: 'x')
- `Regress`: Calculate and plot a linear regression.
- `quiet`: If 1, don't include database name on title.

Returns:

a_p: A plot_abstract.

See also:

Author: Cengiz Gunay <cgunay@emory.edu>, 2007/11/30

A.52.65 Method tests_db/plotTestsHistsMatrix

Summary: Create a matrix plot of test histograms.

Usage:

```
a_pm = plotTestsHistsMatrix(a_db, title_str, props)
```

Description: Skips the 'ItemIndex' test. If no axisLimits is given in stackProps, y-ranges are the maximal found from db.

Parameters:

`a_db`: One or more params_tests_db object. Multiple databases in an array will produce vertical stacks.

`title_str`: (Optional) A string to be concatanated to the title.

`props`: A structure with any optional properties, passed to plot_abstract.

`orient`: Orientation of the plot_stack. 'x', 'y', or 'matrix' (default).

`histBins`: Number of histogram bins.

`quiet`: Don't put the DB id on the title.

`plotProps`: Props passed to individual plots.

`stackProps`: Passed to vertical plot stacks.

Returns:

a_pm: A plot_stack with the plots organized in matrix form.

See also: `params_tests_profile` (p. 162), `plotVar` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/17

A.52.66 Method `tests_db/plotUITable`

Summary: Display rows in figure table element.

Usage:

```
a_p = plotUITable(a_db, title_str, props)
```

Parameters:

`a_db`: A params_tests_db object.

`title_str`: (Optional) A string to be concatanated to the title.

`props`: A structure with any optional properties

Returns:

a_p: A plot_abstract.

Example:

```
» plotFigure(plotUITable(my_db(1:5, :), 'my DB'))
```

See also: `displayRows` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2014/10/22

A.52.67 Method `tests_db/plotUniquesStats2D`

Summary: 2D image plot of the change in column mean for unique values of two other columns.

Usage:

```
an_image_plot = plotUniquesStats2D(a_db, unique_test1, unique_test2, stat_test, title_str, props)
```

Parameters:

`a_db`: A tests_db.

`unique_test1, unique_test2`: Columns whose unique values make up the X
& Y of the 2D image plot.

`stat_test`: Column for which statsMeanSTD will be calculated for each unique value.

`props`: A structure with any optional properties.

- `popMean`: If specified, plot a dotted line specifying the population mean. If NaN, calculate from given a_db.
- `popDev`: Use this value +/- to choose colorbar extents (default=.3 or 2*STD if popMean=NaN).
- `colorbar`: Show vertical colorbar axis (see plotImage).
- `uniqueVals1,uniqueVals2`: Use these unique values for unique_test1,unique_test2.
- `statsFunc`: tests_db/stats* method to use (default: statsMeanStd).
- `statsRow`: The row to pick from statsFunc results (default: mean). (rest passed to plotImage and plot_abstract).

Returns:

an_image_plot: A plot_abstract object to be plotted.

Example:

```
» plotFigure(plotUniquesStats2D(triplet_param_success_db, ...
'F_tau_m', 'S_tau_m', 'successDefault', ...
'accross triplets',
struct('fixedSize', [4 3], 'popMean', NaN, ...
'colorbar', 1, 'quiet', 1, 'border', [0.03 0 0.03 0])))
```

See also: `tests_db` (p. 256), `sortedUniqueValues` (p. **??**), `statsMeanStd` (p. **??**), `plot_abstract` (p. 180), `plotImage` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2008/04/14

A.52.68 Method `tests_db/plotUniquesStatsBars`

Summary: Creates a mean-STD bar plot of a column for unique values of another column.

Usage:

```
a_bar_plot = plotUniquesStatsBars(a_db, unique_test, stat_test, title_str, props)
```

Parameters:

`a_db`: A tests_db.

`unique_test`: Column for which to generate bars for each of its unique values.

`stat_test`: Column for which statsMeanSTD will be calculated for each bar.

`props`: A structure with any optional properties.

- `popMean`: If specified, plot a dotted line specifying the population mean. If NaN, calculated from a_db.
- `yLims`: two-element vector for specifying y axis limits showing interesting part of the bar plot.
- `uniqueVals`: Use these unique values for unique_test. (rest passed to plot_bars [and plot_superpose if popMean]).

Returns:

a_bar_plot: A plot_abstract object to be plotted.

Example:

```
» plotFigure(plotUniquesStatsBars(triplet_param_success_db, 'F_tau_m', ...
'successDefault', 'across triplets', ...
struct('fixedSize', [4 2], 'yLims', [.7 .9], ...
'popMean', 0.82, 'quiet', 1)))
```

See also: `tests_db` (p. 256), `sortedUniqueValues` (p. **??**), `statsMeanStd` (p. **??**), `plot_bars` (p. 187)

Author: Cengiz Gunay <cgunay@emory.edu>, 2008/04/14

A.52.69 Method `tests_db/plotUniquesStatsStacked3D`

Summary: Stack of 2D image plots of a column mean at unique values of three other columns.

Usage:

```
a_stacked_plot = plotUniquesStatsStacked3D(a_db, unique_test1, unique_test2, unique_test3,
stat_test, title_str, props)
```

Parameters:

`a_db`: A tests_db.

`unique_test1, unique_test2`: Columns whose unique values make up the X
& Y of the 2D image plot.

`unique_test3`: Column whose unique values make up stacked dimension.

`stat_test`: Column for which statsMeanSTD will be calculated for each unique value.

`props`: A structure with any optional properties.
(rest passed to plotUniquesStats2D and plot_stack).

Returns:

a_stacked_plot: A plot_abstract object to be plotted.

See also: `tests_db` (p. 256), `sortedUniqueValues` (p. **??**), `statsMeanStd` (p. **??**), `plot_abstract` (p. 180), `plotImage` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2008/04/15

A.52.70 Method `tests_db/plotXRows`

Summary: Create a scatter plot with a test versus the row numbers on the X-axis.

Usage:

```
a_p = plotXRows(a_db, test_y, title_str, short_title, props)
```

Parameters:

`a_db`: A params_tests_db object.

`test_y`: Y variable.

`title_str`: (Optional) A string to be concatanated to the title.

`short_title`: (Optional) Few words that may appear in legends of multiplot.

`props`: A structure with any optional properties passed to plotScatter.

`RowName`: Label to show on X-axis, becomes a db column (default='RowNumber')
`Vertical`: If provided, put the rows on the Y-axis instead.

Returns:

a_p: A plot_abstract.

See also:

Author: Cengiz Gunay <cgunay@emory.edu>, 2007/01/16

A.52.71 Method `tests_db/plotYTests`

Summary: Create a plot given database measures against given X-axis values, for each row.

Usage:

```
a_p = plotYTests(a_db, x_vals, tests, axis_labels, title_str, short_title, command, props)
```

Parameters:

`a_db`: A params_tests_db object.

`x_vals`: A vector of X-axis values.

`tests`: A vector or cell array of columns to correspond to each value from x_vals.

`axis_labels`: Cell array of X & Y axis labels.

`title_str`: (Optional) A string to be concatanated to the title.

`short_title`: (Optional) Few words that may appear in legends of multiplot.

`command`: (Optional) Command to do the plotting with (default: 'plot')

`props`: A structure with any optional properties.

`LineStyle`: Plot line style to use. (default: 'd-')

`ShowErrorbars`: If 1, errorbars are added to each point.

`StatsDB`: If given, use this stats_db for the errorbar (default=statsMeanStd(a_db)).

`jitterX`: Randomly jitter x-axis locations by this magnitude.

`quiet`: If 1, don't include database name on title.

Returns:

a_p: A plot_abstract.

Example:

```
» a_p = plotYTests(a_db_row, [0 40 100 200], ...
'IniSpontSpikeRateISI_0pA', 'PulseIni100msSpikeRateISI_D40pA', ...
'PulseIni100msSpikeRateISI_D100pA', 'PulseIni100msSpikeRateISI_D200pA', ...
'current pulse [pA]', 'firing rate [Hz]', ', f-I curves', 'neuron 1');
» plotFigure(a_p);
```

See also: plotFigure (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/01/23

A.52.72 Method `tests_db/plot_abstract`

Summary: Default visualization for a database.

Usage:

```
a_pm = plot_abstract(a_db, title_str)
```

Description: Calls plotTestsHistsMatrix. Subclasses should override this method to provide their own visualization.

Parameters:

`a_db`: A params_tests_db object.

`title_str`: (Optional) A string to be concatanated to the title.

`props`: A structure with any optional properties.

Returns:

a_pm: A plot_stack with the plots organized in matrix form

Example:

```
» plot(my_db, ':  first impression')
will call this function and send the generated plot to the plotFigure function.
```

See also: `plot_abstract/plot_abstract` (p. 180), `plotTestsHistsMatrix` (p. **??**), `plotFigure` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/08/17

A.52.73 Method `tests_db/plot_bars`

Summary: Creates a bar graph comparing all DB rows in groups, with a separate axis for each column.

Usage:

```
a_plot = plot_bars(a_tests_db, title_str, props)
```

Parameters:

`a_tests_db`: A tests_db object.

`title_str`: (Optional) The plot title.

`props`: A structure with any optional properties.

Returns:

a_plot: A object of plot_bars or one of its subclasses.

See also: `plot_abstract` (p. 180), `plot_simple` (p. 193)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/03/13

A.52.74 Method `tests_db/plus`

Summary: Adds a DB to another or to a scalar.

Usage:

```
a_db = plus(left_obj, right_obj)
```

Description: If DBs have mismatching columns only the common columns will be kept. In any case, the resulting DB columns will be sorted in the order of the left-hand-side DB.

Parameters:

`left_obj, right_obj`: Operands of the addition. One must be of type tests_db and the other can be a scalar or tests_db.

Returns:

a_db: The resulting tests_db.

See also: `plus` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2007/12/13

A.52.75 Method `tests_db/princomp`

Summary: Generates a database of the principal components of given DB.

Usage:

```
a_pca_db = princomp(db, props)
```

Parameters:

`db`: A tests_db object.

`props`: A structure with any optional properties.

`normalized`: If specified zscore is used before princomp.

Returns:

a_pca_db: A tests_db where each row is a principal component.

See also: `princomp` (p. **??**), `zscore` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/09/21

A.52.76 Method `tests_db/processDimNonNaNInf`

Summary: Recursively process the specified dimension with the desired function after removing NaNs and Infs.

Usage:

```
[a_db, n, i] = processDimNonNaNInf(a_db, dim, a_func, a_func_name)
```

Description: Does a recursive operation over other dimensions in order to remove NaN and Inf values. This takes more time than applying the function directly.

Parameters:

`a_db`**:** A tests_db object.

`dim`**:** Work down dimension (see mean).

`a_func`**:** A function name or handle to be passed to feval that takes the data as the first argument and dimension to work as second.

`a_func_name`**:** (Optional) A name to add to the id of a_db.

Returns:

a_db: The DB with one row of max values, with selected dimension replaced by the output of the given function. n: (Optional) Numbers of used values in each call of a_func. i: (Optional) Indices returned by a_func.

Example:

```
a_db = tests_3D_db(rand(5, 5, 5));
» b_db = processDimNonNaNInf(a_db, 1, 'mean')
will find the mean of rows in each page of the random 3D matrix.
» b_db = processDimNonNaNInf(a_db, 1, @(x,y)(max(x, [], y)), 'max')
more complex function form with 'max'.
```

See also: `max` (p. **??**), `mean` (p. **??**), `feval` (p. **??**), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2008/05/27

A.52.77 Method `tests_db/rankMatching`

Summary: Create a ranking db of row distances of db to given criterion db.

Usage:

```
a_ranked_db = rankMatching(db, crit_db, props)
```

Description: The crit_db parameter can be created with the matchingRow method. TestWeights modify the importance of each measure. Rows containing NaNs can be removed using noNaNRows before calling rankMatching. Warning: existing RowIndex column in db will be replaced with the ranking RowIndex used for joinOriginal.

Parameters:

`db`: A tests_db to rank.

`crit_db`: A tests_db object holding the match criterion tests and stds.

`props`: A structure with any optional properties.

`limitSTD`: Truncate error values at this many STDs.

`tolerateNaNs`: Multiplied by 3xSTD to replace NaN values. 0 means to skip NaNs in the distance calculation, scaling sum of errors by number of non-NaN entries. Negative values accepted; -1 means -3xSTDs error (default=+1).

`testWeights`: Structure array associating tests and multiplicative weights.

`restoreWeights`: Reverse the testWeights application after calculating distances.

`topRows`: If given, only return this many of the top rows.

`useMahal`: Use the Mahalonobis distance from the covariance matrix in crit_db.

Returns:

a_ranked_db: A ranked_db object.

Example:

```
Select a target row from mesured distribution (e.g., experimental data):
» a_crit_db = matchingRow(exp_db, 12);
Rank another database by comparing to selected row:
» a_ranked_db = rankMatching(orig_db, a_crit_db);
Look at top ranked rows:
» displayRows(a_ranked_db(1:5, :))
```

See also: matchingRow (p. **??**), joinOriginal (p. **??**), tests_db (p. 256), noNaNRows (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/12/08

A.52.78 Method `tests_db/rdivide`

Summary: Adds a DB to another or to a scalar.

Usage:

```
a_db = rdivide(left_obj, right_obj)
```

Description: If DBs have mismatching columns only the common columns will be kept. In any case, the resulting DB columns will be sorted in the order of the left-hand-side DB.

Parameters:

`left_obj, right_obj`: Operands of the addition. One must be of type tests_db and the other can be a scalar or tests_db.

Returns:

a_db: The resulting tests_db.

See also: `rdivide` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2007/12/13

A.52.79 Method `tests_db/renameColumns`

Summary: Rename one or more existing columns.

Usage:

```
a_db = renameColumns(a_db, old_names, new_names)
```

Description: This is a cheap operation than modifies meta-data kept in object. For the regular expression renaming, the old_names and new_names parameters are passed to the regexprep command after removing the delimiting slashes (//). At least one grouping construct ('()') must be used in the search pattern such that it can be used in the replacement pattern (e.g., '$1'). See example above. This function uses the generic renameIdx that can work on row, column, or page indices.

Parameters:

`a_db`: A tests_db object.

`old_names`: A cell array of existing names, array of numerical indices, or a regular
expression denoted between slashes (e.g., '/(.*)/').

`new_names`: New names to replace existing ones OR regular expression replace string (no slashes, e.g, '$1_test'). See regexprep command.

Returns:

a_db: The tests_db object that includes the new columns.

Example:

```
» new_db = renameColumns(a_db, 'PulseIni100msSpikeRateISI_D40pA', 'Firing_rate');
» new_db = renameColumns(a_db, 1, 'Firing_rate');
» new_db = renameColumns(a_db, '/(.*)/', '$1_old');
» new_db = renameColumns(a_db, 'a', 'b', 'c', 'd');
```

See also: `renameIdx` (p. **??**), `regexprep` (p. **??**), `allocateRows` (p. **??**), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2017/06/09

A.52.80 Method `tests_db/renameRows`

Summary: Rename one or more existing rows.

Usage:

```
a_db = renameRows(a_db, old_names, new_names)
```

Description: This is a cheap operation than modifies meta-data kept in object. For the regular expression renaming, the old_names and new_names parameters are passed to the regexprep command after removing the delimiting slashes (//). At least one grouping construct ('()') must be used in the search pattern such that it can be used in the replacement pattern (e.g., '$1'). See example above. This function uses the generic renameIdx that can work on row, column, or page indices.

Parameters:

`a_db`**:** A tests_db object.

`old_names`**:** A cell array of existing names, array of numerical indices, or a regular
expression denoted between slashes (e.g., '/(.*)/').

`new_names`**:** New names to replace existing ones OR regular expression
replace string (no slashes, e.g, '$1_test'). See regexprep command.

Returns:

a_db: The tests_db object that includes the new rows.

Example:

```
» new_db = renameRows(a_db, 'PulseIni100msSpikeRateISI_D40pA', 'Firing_rate');
» new_db = renameRows(a_db, 1, 'Firing_rate');
» new_db = renameRows(a_db, '/(.*)/', '$1_old');
» new_db = renameRows(a_db, 'a', 'b', 'c', 'd');
```

See also: `renameIdx` (p. **??**), `regexprep` (p. **??**), `allocateRows` (p. **??**), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2017/06/09

A.52.81 Method `tests_db/rop`

Summary: Prepares aligned columns in two DBs or one DB and a scalar for an array arithmetic operation.

Usage:

```
a_db = rop(left_obj, right_obj, op_func, op_id)
```

Description: If DBs have mismatching columns only the common columns will be kept. In any case, the resulting DB columns will be sorted in the order of the left-hand-side DB. Array addition (plus), subtraction (minus), multiplication (mtimes) and division (rdivide) use this function to align columns.

Parameters:

`left_obj, right_obj`: Operands of the operation. One must be of type tests_db and the other can be a scalar or tests_db.

`op_func`: Operation function (e.g., @plus).

`op_id`: A string to represent the operation that will show up in the returned id.

Returns:

a_db: The resulting tests_db.

See also: `tests_db/plus` (p. 299), `tests_db/minus` (p. 283), `tests_db/mtimes` (p. 283), `tests_db/rdivide` (p. 301)

Author: Cengiz Gunay <cgunay@emory.edu>, 2007/12/13

A.52.82 Method `tests_db/rows2Struct`

Summary: Convert given rows of database to a structure array.

Usage:

```
s = rows2Struct(db, rows, pages)
```

Parameters:

`db`: A tests_db object.

`rows`: Indices of rows in db.

`pages`: Pages of db.

Returns:

s: A structure of column name and value pairs.

See also: `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/08/17

A.52.83 Method `tests_db/set`

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.52.84 Method `tests_db/setProp`

Summary: Generic method for setting optional object properties.

Usage:

```
obj = setProp(obj, prop1, val1, prop2, val2, ...)
```

Description: Modifies or adds property values. As many property name-value pairs can be specified.

Parameters:

`obj`: Any object that has a props field.

`attr`: Property name

`val`: Property value.

Returns:

obj: The new object with the updated properties.

See also:

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/22

A.52.85 Method `tests_db/setRows`

Summary: Sets the rows of observations in tests_db.

Usage:

```
index = setRows(obj, rows)
```

Description: Sets a new set of observations to the database and returns the new DB.

Parameters:

`obj`: A tests_db object.

`rows`: A matrix that contains rows for the DB.

Returns:

obj: The tests_db object with the new rows.

See also: `allocateRows` (p. **??**), `addRow` (p. **??**), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/08

A.52.86 Method `tests_db/shufflecols`

Summary: Returns a db with shuffled columns of given rows.

Usage:

```
s = shufflecols(db, rows, grouped)
```

Description: Can be used for shuffle prediction. Basically, shuffle columns of tests and rerun high order analyses.

Parameters:

`db`: A tests_db object.

`rows`: Rows to shuffle.

`grouped`: If 1 then apply same shuffling to all rows,
if 0 shuffle each row independently (default=0).

Returns:

a_db: The shuffled db.

See also: `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2015/11/19

A.52.87 Method `tests_db/shufflerows`

Summary: Returns a db with shuffled rows of given columns.

Usage:

```
s = shufflerows(db, tests, grouped)
```

Description: Can be used for shuffle prediction. Basically, shuffle rows of tests and rerun high order analyses.

Parameters:

`db`: A tests_db object.

`tests`: Columns to shuffle.

`grouped`: If 1 then shuffle tests all together,
if 0 shuffle each test separately (default=0).

Returns:

a_db: The shuffled db.

See also: `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/10

A.52.88 Method `tests_db/sortrows`

Summary: Returns a sorted_db according to given columns.

Usage:

```
[sorted_db, idx] = sortrows(db, cols, props)
```

Description: WARNING: For multi-page dbs, sorts only the first page and applies the ordering to all other pages which may produce wrong results for some applications.

Parameters:

`db`: A tests_db object.

`cols`: Columns to use for sorting.

`props`: Structure of optional parameters.

`reverse`: If exists, sort in reverse order.

Returns:

sorted_db: The sorted tests_db. idx: The row index permutation vector, such that sorted_db = db(idx, :).

See also: `sortrows` (p. **??**), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/11

A.52.89 Method `tests_db/sqrt`

Summary: Takes the square root of a_db.

Usage:

```
a_db = sqrt(a_db)
```

Description: Overloaded sqrt function.

Parameters:

`a_db`: A tests_db.

Returns:

a_db: The resulting tests_db.

See also: `sqrt` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2007/12/13

A.52.90 Method `tests_db/statsAll`

Summary: Makes a stats_db with rows of mean, STD, SE, and CV of the tests' distributions in db.

Usage:

```
a_stats_db = statsAll(db, tests, props)
```

Parameters:

`db`: A tests_db object.

`tests`: A selection of tests (see onlyRowsTests).

`props`: A structure with any optional properties for stats_db.

Returns:

a_stats_db: A stats_db object.

See also: `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/08/24

A.52.91 Method `tests_db/statsBounds`

Summary: Generates a stats_db object with three rows corresponding to the mean, min, max and number of observations of the tests' distributions.

Usage:

```
a_stats_db = statsBounds(a_db, tests, props)
```

Description: A page is generated for each page of data in db.

Parameters:

`a_db`: A tests_db object.

`tests`: A selection of tests (see onlyRowsTests).

`props`: A structure with any optional properties for stats_db.

Returns:

a_stats_db: A stats_db object.

See also: `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/07

A.52.92 Method `tests_db/statsMeanSE`

Summary: Generates a stats_db object with two rows corresponding to the mean and standard error (SE) of the tests' distributions.

Usage:

```
a_stats_db = statsMeanSE(db, tests, props)
```

Parameters:

`db`: A tests_db object.

`tests`: A selection of tests (see onlyRowsTests).

`props`: A structure with any optional properties for stats_db.

Returns:

a_stats_db: A stats_db object.

See also: `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/07

A.52.93 Method `tests_db/statsMeanStd`

Summary: Generates a stats_db object with mean, STD, and number of observations of the tests' distributions.

Usage:

```
a_stats_db = statsMeanStd(db, tests, props)
```

Parameters:

`db`: A tests_db object.

`tests`: A selection of tests (see onlyRowsTests).

`props`: A structure with any optional properties for stats_db.

Returns:

a_stats_db: A stats_db object.

See also: `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/07

A.52.94 Method `tests_db/std`

Summary: Returns the std of the data matrix of a_db. Ignores NaN values.

Usage:

```
[a_db, n] = std(a_db, sflag, dim)
```

Description: Does a recursive operation over dimensions in order to remove NaN values. This takes considerable amount of time compared with a straightforward std operation.

Parameters:

`a_db`: A tests_db object.

`dim`: Work down dimension.

Returns:

a_db: The DB with std values. n: (Optional) Numbers of non-NaN rows included in calculating each column.

See also: `std` (p. **??**), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/06

A.52.95 Method `tests_db/subsasgn`

Summary: Defines generic index-based assignment for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/02/06

A.52.96 Method `tests_db/subsref`

Summary: Defines indexing for tests_db objects for () and . operations.

Usage:

```
obj = obj(rows, tests) obj = obj.attribute
```

Description: Returns attributes or selects the given test columns and rows and returns in a new tests_db object.

Parameters:

`obj`: A tests_db object.

`rows`: A logical or index vector of rows. If ':', all rows.

`tests`: Cell array of test names or column indices. If ':', all tests.

`attribute`: A tests_db class attribute.

Returns:

obj: The new tests_db object.

See also: `subsref` (p. **??**), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/17

A.52.97 Method `tests_db/sum`

Summary: Creates a tests_db by summing all rows.

Usage:

```
a_db = sum(a_db, dim, props)
```

Description: Applies the sum function to whole DB. The resulting DB will have one row.

Parameters:

`a_db`: A tests_db object.

`props`: Optional properties.

Returns:

a_db: The resulting tests_db.

See also: `sum` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/05/24

A.52.98 Method `tests_db/swapColsPages`

Summary: Swaps the column dimension with the page dimension of the tests_db.

Usage:

```
a_db = swapColsPages(db)
```

Description: Watered-down version of the tests_3D_db/swapColsPages function that does not touch column indices.

Parameters:

`db`: A tests_db object.

Returns:

a_db: A tests_db object.

See also: `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2017/06/09

A.52.99 Method `tests_db/swapRowsPages`

Summary: Swaps the row dimension with the page dimension of the tests_db.

Usage:

```
a_db = swapRowsPages(db)
```

Description: Watered-down version of the tests_3D_db/swapRowsPages function that does not touch row indices.

Parameters:

`db`: A tests_db object.

Returns:

a_db: A tests_db object.

See also: `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/04

A.52.100 Method `tests_db/tests2cols`

Summary: Find column numbers from a test names/numbers specification.

Usage:

```
cols = tests2cols(db, tests)
```

Description: Uses tests2idx.

Parameters:

`db`: A tests_db object.

`tests`: Either a single or array of column numbers, or a single test name or a cell array of test names. If ':', all tests. For name strings, regular expressions are supported if quoted with slashes (e.g., '/a.*/'). See tests2idx for more.

Returns:

cols: Array of column indices.

See also: `tests_db` (p. 256), `tests2idx` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/07

A.52.101 Method `tests_db/tests2idx`

Summary: Find dimension indices from a test names/numbers specification.

Usage:

```
idx = tests2idx(db, dim_num, tests)
```

Parameters:

`db`: A tests_db object.

`dim_num`: Number between 1-3 to choose dimension: row, column, or page.

`tests`: Either a single or array of column numbers, or a single test name or a cell array of test names. If ':', all tests. For name strings, regular expressions are supported if quoted with slashes (e.g., '/a.*/')

Returns:

idx: Array of column indices.

Example:

```
» cols = tests2idx(a_db, 2, 'col1', '/col2+/');
will return indices of col1 and columns like col2, col22, col22, etc.
```

See also: `tests_db` (p. 256), `tests2cols` (p. **??**), `regexp` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/07

A.52.102 Method `tests_db/tests2log`

Summary: Return logical array of indices from a test names/numbers specification.

Usage:

```
a_log = tests2log(db, dim_name, tests)
```

Description: See tests2idx.

Parameters:

`db`: A tests_db object.

`dim_num`: Number between 1-3 to choose dimension: row, column, or page.

`tests`: Either a single or array of column numbers, or a single test name or a cell array of test names. If ':', all tests. For name strings, regular expressions are supported if quoted with slashes (e.g., '/a.*/')

Returns:

a_log: Array of column indices.

Example:

```
» cols = tests2log(a_db, 'col', 'col1', '/col2+/');
» stripped_db = a_db(:,  cols)
will remove columns col1 and col2, col22, col22, etc.  from stripped_db.
```

See also: tests_db (p. 256), tests2cols (p. **??**), regexp (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2008/05/27

A.52.103 Method tests_db/testsHists

Summary: Calculates histograms for all tests.

Usage:

```
t_hists = testsHists(a_db, num_bins)
```

Parameters:

a_db: A tests_db object.

num_bins: Number of histogram bins (Optional, default=100), or vector of histogram bin centers.

Returns:

t_hists: An array of histograms for each test in a_db.

See also: params_tests_profile (p. 162)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/04/27

A.52.104 Method tests_db/times

Usage:

```
a_db = mtimes(left_obj, right_obj)
```

Parameters:

left_obj, right_obj: Operands of the multiplication. One or more must be of type tests_db.

Returns:

a_db: The resulting tests_db.

See also: tests_db/times (p. 314), mtimes (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/05/24

A.52.105 Method `tests_db/transpose`

Summary: Transposes data matrix and swaps row and columns metadata as well.

Usage:

```
a_db = transpose(a_db)
```

Parameters:

`a_db`: A tests_db.

Returns:

a_db: The transposed tests_db.

See also: `transpose` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2007/02/07

A.52.106 Method `tests_db/uminus`

Summary: Unary minus or negation.

Usage:

```
a_db = uminus(left_obj)
```

Parameters:

`left_obj`: A tests_db object.

Returns:

a_db: The resulting tests_db.

See also: `uminus` (p. **??**), `uop` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2008/01/16

A.52.107 Method `tests_db/unique`

Summary: Returns DB with unique rows.

Usage:

```
[a_db idx] = unique(a_db)
```

Description: Keeps the original DB order. Uses the uniqueValues function.

Parameters:

`a_db`: tests_db from which to find uniques.

Returns:

a_db: The resulting tests_db. idx: Indices of the unique rows in the original data matrix.

See also: uniqueValues (p. **??**), unique (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2007/11/19

A.52.108 Method tests_db/uop

Summary: Unary operation.

Usage:

```
a_db = uop(left_obj, op_func, op_id)
```

Description: Applies the operation to the database contents and updates its id field. Unary minus (uminus) uses this function.

Parameters:

`left_obj`: Operands of the operation.

`op_func`: Operation function (e.g., @plus).

`op_id`: A string to represent the operation that will show up in the returned id.

Returns:

a_db: The resulting tests_db.

See also: tests_db/uminus (p. 315), uminus (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2008/01/16

A.52.109 Method tests_db/vertcat

Summary: Vertical concatanation [db;with_db;...] operator.

Usage:

```
a_db = vertcat(db, with_db, ...)
```

Description: Concatanates rows of with_db to rows of db. Overrides the built-in vertcat function that is called when [db;with_db] is executed. If the first argument is a array of DBs, then this functionality is not needed; built-in vertcat is called.

Parameters:

`db`: A tests_db object.

`with_db`: A tests_db object whose rows are concatanated to db.

Returns:

a_db: A tests_db that contains rows of db and with_db.

See also: `vertcat` (p. **??**), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/01/25

A.53 Class `trace`

A.53.1 Constructor `trace/trace`

Summary: Load a generic data trace. It can be membrane voltage, current, etc.

Usage:

```
obj = trace(data_src, dt, dy, id, props)
```

Description: This object is designed to recognize most data file formats. See the data_src parameter below. Traces for specific experimental or simulation protocols can extend this class for adding new parameters.

Parameters:

`data_src`: Trace data as a column vector OR name of a data file generated by either

Genesis (.bin, .gbin, .genflac), Neuron, PCDX (.all), Matlab (.mat), NeuroShare (.nsn, .nev, .stb, .plx, .nex, .map, .son, .smr, .mcd), ASCII (.txt) or Axoclamp (.abf) files. As last resort, file will be loaded with Matlab's load() function.

`dt`: Time resolution in [s], unless specified in HDF5 or NeuroShare file. For Neuron ASCII files, it is used as the file data units.

`dy`: y-axis resolution in [ISI (V, A, etc.)], unless specified in HDF5 or NeuroShare file.

`id`: Identification string

`props`: A structure with any optional properties.

`scale_y`: Y-axis scale to be applied to loaded data.

`offset_y`: Y-axis offset to be added to loaded and scaled data.

`unit_y`: Unit of Y-axis as in 'V' or 'A' (default='V').

`y_label`: String to put on Y-axis of plots.

`trace_time_start`: Samples in the beginning to discard [dt]

`baseline`: Resting potential.

`channel`: Channel to read from file Genesis, PCDX, NeuroShare or Neuron file, or column in a data vector.

`numTraces`: Divide the single column vector of data into this many columns by making it a matrix.

`file_type`: Specify file type instead of guessing from extension: 'genesis': Raw binary files created with Genesis disk_out method. 'genesis_flac': Compressed Genesis binary files. 'neuron': Binary files created with Neuron's Vector.vwrite method. 'neuronascii': Ascii files created from Neuron's Vector objects. Uses time step in file to scale given dt (Must be in ms). 'pcdx': .ALL data acquisition files from PCDX program. 'matlab': Matlab .MAT binary files with matrix data. 'neuroshare': One of the vendor formats recognized by NeuroShare Windows DLLs. See above and http://neuroshare.org. A scale_y value may need to be supplied to get the correct units. 'abf': AxoClamp .ABF format read with abf2load from Matlab File-Exchange. 'txt': Generic, space-separated ASCII file.

`file_endian`: 'l' for little endian and 'b' for big endian (default='n', for native endian). See machineformat option in fopen for more info.

`traces`: Trace numbers as a numeric array or as a string with numeric ranges (e.g., '1 2 5-10 28') for PCDX files.

`spike_finder`: Method of finding spikes (1 uses findFilteredSpikes.m, 2 for Li Su's findspikes, 3 for Alfonso Delgado Reyes's findspikes_old, and 4 for using Matlab's findpeaks method). Methods 2-4 require a threshold. For method 4, see additional findpeaks* props below.

`threshold`: Spike finding threshold. For the findspikes method, it is either a scalar, or [thres1 thres2] to define a range. For findFilteredSpikes it is used on the filtered data and the default is 2/3 max amplitude of band-passed data, but with a minimum of 15. For findpeaks, it sets the MinPeakHeight parameter.

`downThreshold`: (Only for findFilteredSpikes) Size of the trough after the spike peak in filtered data (Default=-2).

`minInit2MaxAmp, minMin2MaxAmp`: For spike_shape elimination, conditions of minimal allowed values for initial point to max point and minimal point to max point, respectively (Default=10 for both).

`init_Vm_method`: Method of finding spike thresholds during spike shape calculation (see spike_shape/spike_shape).

`init_threshold`: Spike initiation threshold (deriv or accel). (see above methods and implementation in calcInitVm)

`init_lo_thr, init_hi_thr`: Low and high thresholds for slope.

`custom_filter`: Recommended if sampling rate differs appreciably from 10 kHz.

If custom_filter == 1, a filter with custom lowpass and highpass cutoffs can be specified. This allows for fast and accurate spike discrimination. The

filter type used is a 2-pole butterworth, different than the default high-order Cheby2. Creates new prop called 'butterWorth' to hold the filter.

`lowPassFreq`: If set, it sets a new low pass cutoff for custom filter. Default is 3000Hz

`highPassFreq`: If set it sets a new high pass cutoff for custom filter. Default is 50 Hz

`findpeaksArgs`: Cell array of arguments to pass to findpeaks.

`findpeaksSign`: Choose -1 to flip the sign of data and look for negative peaks (default=1).

`quiet`: If 1, reduces the amount of textual description in plots and does not add information to id field.

Returns a structure object with the following fields:

data: The trace column matrix. dt, dy, id, props (see above)

See also: `spikes` (p. 227), `spike_shape` (p. 215), `cip_trace` (p. 56), `period` (p. 162), `findpeaks` (p. **??**), `findFilteredSpikes` (p. **??**), `findspikes` (p. **??**), `findspikes_old` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/07/30

A.53.2 Method `trace/analyzeSpikesInPeriod`

Summary: Returns results and a db of spikes by collecting test results of a trace, analyzing each individual spike.

Usage:

```
[results period_spikes a_spikes_db spikes_stats_db spikes_hists_dbs] =
analyzeSpikesInPeriod(a_trace, a_spikes, period, prefix_str)
```

Parameters:

`a_trace`: A trace object.

`a_spikes`: A spikes object from the a_trace object.

`period`: A period of object of a_trace object of interest.

`prefix_str`: Prefix string to be added to spike shape results.

Returns:

results: Results structure names prefixed with prefix_str. period_spikes: Corrected spikes object for this period. a_spikes_db: A mini spikes database of results from each spike in period. spikes_stats_db: Statistics from the mini spikes database. spikes_hists_dbs: Cell array of histograms from the mini spikes database.

See also: trace (p. 317), spikes (p. 227), period (p. 162), spike_shape (p. 215), getProfileAllSpikes (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/05/04

A.53.3 Method trace/avgTraces

Summary: Average multiple traces.

Usage:

```
[avg_tr sd_tr] = avgTraces(traces, props)
```

Parameters:

traces: A vector of trace objects.

props: A structure with any optional properties.

calcSE: If given, calculate standard error instead of deviation.

id: String to replace the id property of averaged trace. The term "average" or "SD" will be prepended to it. By default it will be lengthy and show the arithmetic done.

Returns:

avg_tr: A trace object that holds the average. sd_tr: A trace object that holds the standard deviation or error.

See also: trace (p. 317)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/11/09

A.53.4 Method trace/binary_op

Summary: Generic binary operator applications for trace objects.

Usage:

```
result_tr = binary_op(left_tr, right_tr, op_func, op_id, props)
```

Parameters:

left_tr, right_tr: trace objects.

op_func: Operation function (e.g., @plus).

op_id: A string to represent the operation that will show up in the returned id.

props: A structure with any optional properties.

Returns:

result_tr: Resulting trace object.

Example:

```
» result_tr = binary_op(vc1, vc2, @minus, '-')
```

See also: trace (p. 317), plus (p. **??**), minus (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/05/21

A.53.5 Method trace/calcAvg

Summary: Calculates the average value of the given period of the trace, t.

Usage:

```
avg_val = calcAvg(t, period)
```

Parameters:

`t`: A trace object.

`period`: A period object (optional).

See also: period (p. 162), trace (p. 317)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/07/30

A.53.6 Method trace/calcMax

Summary: Calculates the maximal value of the given period of the trace, t.

Usage:

```
[max_val, max_idx] = calcMax(t, period)
```

Parameters:

`t`: A trace object.

`period`: A period object (optional).

Returns:

max_val: The max value. max_idx: Its index in the trace.

See also: period (p. 162), trace (p. 317)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/07/30

A.53.7 Method `trace/calcMin`

Summary: Calculates the minimal value of the given period of the trace, t.

Usage:

```
[min_val, min_idx] = calcMin(t, a_period)
```

Parameters:

`t`: A trace object.

`a_period`: A period object (optional).

Returns:

min_val: The min value. min_idx: Its index in the trace.

See also: `period` (p. 162), `trace` (p. 317)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/07/30

A.53.8 Method `trace/display`

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.53.9 Method `trace/findFilteredSpikes`

Summary: Runs a frequency filter over the data and then finds all peaks using findspikes.

Usage:

```
[times, peaks, n] = findFilteredSpikes(t, a_period, plotit, minamp)
```

Description: Runs a 50-300 Hz band-pass filter over the data and then calls findspikes. The filter both removes low-frequency offset changes, such as cip period effects, and high-frequency noise that is detected as local peaks by findspikes. The spikes found are post-processed to make sure the rise and fall times are consistent. Note: The filter employed only works with data sampled at 10kHz.

Parameters:

`t`: Trace object

`a_period`: Period of interest.

`plotit`: Plots the spikes found if 1.

minamp (optional): minimum amplitude above baseline that must be reached. –> adjust as necessary to discriminate spikes from EPSPs.

`props`: A structure with any optional properties, such as:

`downThreshold`: Size of trough after spike (default=-2)

Returns:

times: The times of spikes [dt]. peaks: The peaks corresponding to the times of spikes. n: The number of spikes.

See also: `findspikes` (p. **??**), `period` (p. 162)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/03/08

A.53.10 Method `trace/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.53.11 Method `trace/getDy`

Summary: Returns dy.

Usage:

```
dy = getDy(t)
```

Parameters:

`t`: A trace object.

Returns:

dy: The dy value.

See also: `trace` (p. 317)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/31

A.53.12 Method `trace/getPotResults`

Usage:

```
results = getPotResults(t)
```

Parameters:

`t`: A trace object.

Returns:

results: A structure associating potential info names to values in mV as follows: min - minimum potential for the whole trace. avg - average potential for the whole trace. max - maximum potential for the whole trace.

See also: spike_shape (p. 215)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/13

A.53.13 Method trace/getProfileAllSpikes

Summary: Creates a trace_allspikes_profile object by collecting test results of a trace, analyzing each individual spike.

Usage:

```
profile_obj = getProfileAllSpikes(a_trace)
```

Description: Analyzes the spontaneous (periodIniSpont), pulse (periodPulse) and the recovery (periodRecSpont) periods separately and produces spike shape distribution results. Rate and CIP measurements are added to these.

Parameters:

a_trace: A trace object.

Returns:

profile_obj: A trace_allspikes_profile object.

See also: trace (p. 317), trace_allspikes_profile (p. 337)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/04/26

A.53.14 Method trace/getRateResults

Summary: Calculate test results related to spike rate for the whole spike period.

Usage:

```
results = getRateResults(a_trace, a_spikes)
```

Parameters:

a_trace: A trace object.

a_spikes: A spikes object.

Returns:

results: A structure associating test names with result values.

See also: trace (p. 317), spikes (p. 227), spike_shape (p. 215)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/30

A.53.15 Method trace/getResults

Summary: Runs all tests defined by this class and return them in a structure.

Usage:

```
results = getResults(a_trace, a_spikes)
```

Parameters:

a_trace: A trace object.

a_spikes: spikes object obtained from the trace object.

Returns:

results: A structure associating test names to values in ms and mV (or mA).

See also: spike_shape (p. 215), spikes (p. 227)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/13

A.53.16 Method trace/getSpike

Summary: Convert a spike in the trace to a spike_shape object.

Usage:

```
obj = getSpike(trace, spikes, spike_num, props)
```

Description: Creates a spike_shape object from desired spike. It is more efficient if you already have the spikes object.

Parameters:

trace: A trace object.

spikes: (Optional) A spikes object obtained from trace, calculated automatically if given as [].

spike_num: The index of spike to extract.

props: A structure with any optional properties.

spike_id: A prefix string added to the spike_shape object's id.

Example:

```
This will create an annotated plot of the third spike in my_trace:
» plotFigure(plotResults(getSpike(my_trace, [], 3)))
```

See also: spike_shape (p. 215)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/04/19

A.53.17 Method trace/lowpassfilt

Summary: Applies a low-pass Butterworth filter to the trace data.

Usage:

```
t = lowpassfilt(t, n, cutoff_freq)
```

Parameters:

`t`: A trace object.

`n`: Order of the filter (e.g., 2)

`cutoff_freq`: Cutoff frequency, max <= sampling rate/2 [Hz].

Returns:

t: updated trace object.

See also: trace (p. 317), butter (p. **??**), filter (p. **??**), filtfilt (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/04/08

A.53.18 Method trace/medianfilt

Summary: Applies a median filter to the trace data.

Usage:

```
t = medianfilt(t, window_size)
```

Parameters:

`t`: A trace object.

`window_size`: N parameter of medianfilt1 (default=3).

Returns:

t: updated trace object.

See also: trace (p. 317), medianfilt1 (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/04/05

A.53.19 Method `trace/minus`

Summary: Subtract trace object right_tr from left_tr.

Usage:

```
sub_tr = minus(left_tr, right_tr, props)
```

Parameters:

`left_tr, right_tr:` trace objects.

`props:` A structure with any optional properties.

Returns:

sub_tr: trace object with subtracted data of left_tr.

Example:

```
» sub_tr = minus(vc1, vc2)
OR
» sub_tr = vc1 - vc2;
plot the subtracted voltage clamp
» plot(sub_tr)
```

See also: `trace` (p. 317), `minus` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/03/11

A.53.20 Method `trace/mtimes`

Summary: Matrix multiply trace object right_tr with left_tr.

Usage:

```
res_tr = mtimes(left_tr, right_tr, props)
```

Parameters:

`left_tr, right_tr:` trace objects.

`props:` A structure with any optional properties.

Returns:

res_tr: resulting trace object

Example:

```
» res_tr = mtimes(vc1, vc2)
OR
» res_tr = vc1 * vc2;
plot the resulting trace
» plot(res_tr)
```

See also: trace (p. 317), mtimes (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/03/11

A.53.21 Method trace/periodWhole

Summary: Returns the boundaries of the whole period of trace, t.

Usage:

```
whole_period = periodWhole(t)
```

Parameters:

`t`: A trace object.

See also: period (p. 162), trace (p. 317)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/07/30

A.53.22 Method trace/plot

Summary: Plots a trace.

Usage:

```
h = plot(t)
```

Parameters:

`t`: A trace object.

`title_str`: (Optional) String to append to plot title.

`props`: A structure with any optional properties, passed to plot_abstract.

Returns:

h: Handle to figure object.

See also: trace (p. 317), plot_abstract (p. 180)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.53.23 Method `trace/plotData`

Summary: Plots a trace.

Usage:

```
a_plot = plotData(t, title_str, props)
```

Description: If t is a vector of traces, returns a vector of plot objects.

Parameters:

`t`: A trace object.

`title_str`: (Optional) String to append to plot title.

`props`: A structure with any optional properties.

`timeScale`: 's' for seconds, or 'ms' for milliseconds.

`quiet`: If 1, only display given title_str.

(rest passed to plot_abstract.)

Returns:

a_plot: A plot_abstract object that can be visualized.

See also: `trace` (p. 317), `trace/plot` (p. 328), `plot_abstract` (p. 180)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/17

A.53.24 Method `trace/plot_abstract`

Summary: Plots a trace by calling plotData.

Usage:

```
a_plot = plot_abstract(t, title_str, props)
```

Description: If t is a vector of traces, returns a vector of plot objects.

Parameters:

`t`: A trace object.

`title_str`: (Optional) String to append to plot title.

`props`: A structure with any optional properties.

`timeScale`: 's' for seconds, or 'ms' for milliseconds.

(rest passed to plot_abstract.)

Returns:

a_plot: A plot_abstract object that can be visualized.

See also: `trace` (p. 317), `trace/plot` (p. 328), `plot_abstract` (p. 180)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/17

A.53.25 Method `trace/plus`

Summary: Subtract trace object right_tr from left_tr.

Usage:

```
sub_tr = plus(left_tr, right_tr, props)
```

Parameters:

`left_tr, right_tr:` trace objects.

`props:` A structure with any optional properties.

Returns:

sub_tr: trace object with subtracted data of left_tr.

Example:

```
» sub_tr = plus(vc1, vc2)
OR
» sub_tr = vc1 + vc2;
plot the subtracted voltage clamp
» plot(sub_tr)
```

See also: `trace` (p. 317), `plus` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/03/11

A.53.26 Method `trace/power`

Summary: left_tr to the power of right_tr (either can be scalars).

Usage:

```
res_tr = power(left_tr, right_tr, props)
```

Parameters:

`left_tr, right_tr:` trace objects.

`props:` A structure with any optional properties.

Returns:

res_tr: resulting trace object

Example:

```
» res_tr = power(vc1, 3)
OR
» res_tr = vc1 .^ 2;
plot the resulting trace
» plot(res_tr)
```

See also: trace (p. 317), power (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/11/09

A.53.27 Method trace/rdivide

Summary: Scalar divide trace object left_tr by right_tr.

Usage:

```
res_tr = rdivide(left_tr, right_tr, props)
```

Parameters:

`left_tr, right_tr:` trace objects.

`props:` A structure with any optional properties.

Returns:

res_tr: resulting trace object

Example:

```
» res_tr = rdivide(vc1, vc2)
OR
» res_tr = vc1 ./ vc2;
plot the resulting trace
» plot(res_tr)
```

See also: trace (p. 317), rdivide (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/11/09

A.53.28 Method trace/runAvg

Summary: Returns a trace which is the running average of this trace.

Usage:

```
avg_t = runAvg(t)
```

Parameters:

`t:` A trace object.

Returns:

avg_t: A trace object that contains the running average of this trace.

See also: trace (p. 317)

Author: Cengiz Gunay <cgunay@emory.edu>, 2008/05/14

A.53.29 Method `trace/saveAsNeuronVecAscii`

Summary: Writes trace in ASCII file in Neuron simulator Vector format.

Usage:

```
saveAsNeuronVecAscii(trace, filename)
```

Description: Data converted to Neuron units of nA and mV. Uses writeNeuronVecAscii.

Parameters:

`a_t`**:** A trace object.

`filename`**:** (optional) Full path to Neuron file. If omitted, a_t.id prefixed with 'neuron-vec-' is used as filename in current directory.

Returns:

Example:

```
saveAsNeuronVecAscii('myvec.dat', data, 1e-4, 1e-3, 'V', 'my membrane voltage');
```

Author: Cengiz Gunay <cengique@users.sf.net> 2012/03/23

A.53.30 Method `trace/set`

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.53.31 Method `trace/setProp`

Summary: Generic method for setting optional object properties.

Usage:

```
obj = setProp(obj, prop1, val1, prop2, val2, ...)
```

Description: Modifies or adds property values. As many property name-value pairs can be specified.

Parameters:

`obj`**:** Any object that has a props field.

`attr`**:** Property name

`val`**:** Property value.

Returns:

obj: The new object with the updated properties.

See also:

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/22

A.53.32 Method `trace/spikes`

Summary: Convert trace to spikes object for spike timing calculations.

Usage:

```
obj = spikes(trace, a_period, plotit, minamp)
```

Description: Creates a spikes object.

Parameters:

`trace`: A trace object.

`a_period`: A period object denoting the part of trace of interest (optional, if empty vector, taken as wholePeriod).

`plotit`: If non-zero, a plot is generated for showing spikes found (optional).

`minamp`: minimum amplitude that must be reached if using findFilteredSpikes. –> adjust as needed to discriminate spikes from EPSPs. (optional)

See also: `spikes` (p. 227)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/07/30

A.53.33 Method `trace/spike_shape`

Summary: Convert averaged spikes in the trace to a spike_shape object.

Usage:

```
obj = spike_shape(trace, spikes, props)
```

Description: Creates a spike_shape object from averaged spikes. USE THIS ONLY IF YOU WANT TO USE AVERAGED SPIKE SHAPES.

Parameters:

`trace`: A trace object.

`spikes`: A spikes object on trace.

See also: `spike_shape` (p. 215)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.53.34 Method `trace/sqrt`

Summary: Square root of trace object.

Usage:

```
res_tr = sqrt(a_tr, props)
```

Parameters:

`a_tr`: A trace object.

`props`: A structure with any optional properties.

Returns:

res_tr: Resulting trace object.

Example:

```
» a_tr = sqrt(vc1)
plot the result
» plot(a_tr)
```

See also: `trace` (p. 317), `sqrt` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/03/11

A.53.35 Method `trace/subsasgn`

Summary: Defines generic index-based assignment for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/02/06

A.53.36 Method `trace/subsref`

Summary: Defines generic indexing for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.53.37 Method `trace/times`

Summary: Scalar multiply trace object right_tr with left_tr.

Usage:

```
res_tr = times(left_tr, right_tr, props)
```

Parameters:

`left_tr, right_tr:` trace objects.

`props:` A structure with any optional properties.

Returns:

res_tr: resulting trace object

Example:

```
» res_tr = times(vc1, vc2)
OR
» res_tr = vc1 .* vc2;
plot the resulting trace
» plot(res_tr)
```

See also: `trace` (p. 317), `times` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/03/11

A.53.38 Method `trace/uminus`

Summary: Revert sign of trace object.

Usage:

```
res_tr = uminus(a_tr, props)
```

Parameters:

`a_tr:` A trace object.

`props:` A structure with any optional properties.

Returns:

res_tr: Resulting trace object.

Example:

```
» a_tr = uminus(vc1)
OR
» a_tr = -vc1;
plot the result
» plot(a_tr)
```

See also: `trace` (p. 317), `uminus` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/03/11

A.53.39 Method `trace/unary_op`

Summary: Generic unary operator applications for trace objects.

Usage:

```
result_tr = unary_op(a_tr, op_func, op_id, props)
```

Parameters:

`a_tr`: A trace object.

`op_func`: Unary operation function (e.g., @uminus).

`op_id`: A string to represent the operation that will show up in the returned id.

`props`: A structure with any optional properties.

Returns:

result_tr: Resulting trace object.

Example:

```
» result_tr = unary_op(vc1, @uminus, '-')
```

See also: `trace` (p. 317), `binary_op` (p. **??**), `uminus` (p. **??**), `sqrt` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/11/09

A.53.40 Method `trace/withinPeriod`

Summary: Returns a trace object valid only within the given period.

Usage:

```
[obj a_period] = withinPeriod(t, a_period, props)
```

Parameters:

`t`: A trace object.

`a_period`: The desired period

`props`: A structure with any optional properties.

`useAvailable`: If 1, don't stop if period not available, use maximum available.

Returns:

obj: A trace object a_period: The period object, updated if useAvailable is requested.

See also: `trace` (p. 317), `period` (p. 162)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/25

A.54 Class `trace_allspikes_profile`

A.54.1 Constructor `trace_allspikes_profile/trace_allspikes_profile`

Summary: Collects individual spike-based test results of a trace.

Usage:

```
obj = trace_allspikes_profile(a_trace, a_spikes, a_spikes_db, results_obj, props)
```

Description: This is a subclass of results_profile. It is made to be used from subclass constructors.

Parameters:

`a_trace`: A trace object.

`a_spikes`: A spikes object.

`spont_spikes_db, pulse_spikes_db, recov_spikes_db`: tests_dbs with spontaneous, pulse and recovery period spike info.

`results_obj`: A results_profile object with test results.

`id`: Identification string.

`props`: A structure with any optional properties.

Returns a structure object with the following fields:

trace, spikes, spont_spikes_db, pulse_spikes_db, recov_spikes_db, props

See also: `trace` (p. 317), `spikes` (p. 227), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/05/04

A.54.2 Method `trace_allspikes_profile/display`

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.54.3 Method `trace_allspikes_profile/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.54.4 Method `trace_allspikes_profile/set`

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.55 Class `trace_profile`

A.55.1 Constructor `trace_profile/trace_profile`

Summary: Creates and collects test results of a trace.

Description: The first usage is fully customizable to be used from subclass constructors. The second usage generates the spikes and spike_shape objects, and collects some generic test results from them. This usage is only provided as an example and is not used practically.

Parameters:

`data_src`: The trace column OR the .MAT filename.

`dt`: Time resolution [s]

`dy`: y-axis resolution [ISI (V, A, etc.)]

`props`: See trace object.

Returns a structure object with the following fields:

trace, spikes, spike_shape, results, id, props.

See also: `trace` (p. 317), `spikes` (p. 227), `spike_shape` (p. 215)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/13

A.55.2 Method `trace_profile/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.56 Class `voltage_clamp`

A.56.1 Constructor `voltage_clamp/voltage_clamp`

Summary: Voltage clamp object with current and voltage traces.

Usage:

```
a_vc = voltage_clamp(data_i, data_v, dt, di, dv, id, props)
```

Description: Uses the generic trace object to store voltage clamp I, V data. Inherits the common methods defined in trace.

Parameters:

`data_i,data_v`: Column vectors of I and V data traces.

`dt`: Time resolution [s].

`di,dv`: y-axis resolution for I and V [A and V, resp]

`id`: Identification string.

`props`: A structure with any optional properties, such as:

`trace_time_start`: Samples in the beginning to discard [dt] (see trace for more)

Returns a structure object with the following fields:

v: Voltage trace object, i: Current trace object, time_steps: Times of voltage steps. v_steps: Mean voltage values before each step (including one after last step) i_steps: Mean current values of steady-state before each step. trace: A parent trace object to inherit methods from.

See also: `trace` (p. 317), `period` (p. 162)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/02/05

A.56.2 Method `voltage_clamp/calcCurPeaks`

Summary: Find current peaks during a voltage step.

Usage:

```
a_vc = calcCurPeaks(a_vc, step_num, props)
```

Parameters:

`a_vc`: A voltage clamp object.

`step_num`: 1 for prestep, 2 for the first step, 3 for next, etc (default=2).

`props`: A structure with any optional properties.

`pulseRange`: Use this range for finding peaks [dt].

`pulseStartRel`: Time to start relative to the step beginning (default=+.3) [ms]. If it has two elements, first one specifies the voltage step .

`pulseEndRel`: Time to end relative to the step end (default=-.3) [ms]. Like pulseStartRel, allows specifying voltage step.

`avgAroundMs`: If given, after finding a peak, average +/- ms around it to reduce noise.

Returns:

a_vc: Updated voltage_clamp object that contains props.iPeaks.

Example:

```
» a_vc = calcCurPeaks(a_vc, 2)
```

See also: `voltage_clamp` (p. 338), `findSteps` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/03/30

A.56.3 Method `voltage_clamp/get`

Summary: Defines generic attribute retrieval for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/14

A.56.4 Method `voltage_clamp/getResults`

Summary: Calculate measurements and observations from this object.

Usage:

```
a_prof = getResults(a_vc, props)
```

Parameters:

`a_vc`: A voltage_clamp object.

`props`: A structure with any optional properties (inherited from a_vc).

`skipStep`: Number of voltage step times to skip at the start (default=0).

Returns:

a_prof: A params_results_profile object with parameters and results structures.

Example:

```
» a_cs = params_tests_dataset(md1, md2)
» a_db = params_tests_db(a_cs)
```

See also: `params_tests_dataset/loadItemProfile` (p. 135), `params_tests_dataset` (p. 132)

Author: Cengiz Gunay <cgunay@emory.edu>, 2011/07/08

A.56.5 Method `voltage_clamp/getTimeRelStep`

Summary: Return time relative to a voltage step.

Usage:

```
rel_time = getTimeRelStep(a_vc, step_num, rel_time, props)
```

Parameters:

`a_vc`: A voltage_clamp object.

`step_num`: Relative to this voltage step. 1 for start of trace, 2 for first voltage change, and so on.

`rel_time`: One or more time values in a vector [ms].

`props`: A structure with any optional properties.

Returns:

rel_time: Time vector from start of trace [dt].

Example:

```
Select [-10, 50] ms range from step 1 into a new VC object.
» new_vc = withinPeriod(a_vc, period(getTimeRelStep(a_vc, 1, [-10 50])))
```

See also: voltage_clamp (p. 338)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/10/18

A.56.6 Method `voltage_clamp/minus`

Summary: Subtract current traces of voltage clamp object right_vc from left_vc.

Usage:

```
sub_vc = minus(left_vc, right_vc, props)
```

Description: Also returns the subtracted voltage trace in props.sub_v for visual inspection of match between the two voltage traces.

Parameters:

`left_vc, right_vc`: voltage_clamp objects.

`props`: A structure with any optional properties.

Returns:

sub_vc: voltage_clamp object with subtracted current and voltage of left_vc.

Example:

```
» sub_vc = minus(vc1, vc2)
OR
» sub_vc = vc1 - vc2;
plot the subtracted voltage clamp
» plot(sub_vc)
plot the subtracted voltage trace, too
» plot(sub_vc.props.sub_v)
```

See also: voltage_clamp (p. 338)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/03/10

A.56.7 Method `voltage_clamp/periodWhole`

Summary: Returns the boundaries of the whole period.

Usage:

```
whole_period = periodWhole(a_vc)
```

Parameters:

`a_vc`: A voltage_clamp object.

See also: `period` (p. 162), `trace` (p. 317), `voltage_clamp` (p. 338)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/04/08

A.56.8 Method `voltage_clamp/plotAllIVs`

Summary: Plot superposed I/V curves for activation, inactivation and steady-state.

Usage:

```
a_p = plotAllIVs(a_vc, title_str, props)
```

Parameters:

`a_vc`: A voltage_clamp object.

`title_str`: (Optional) Text to appear in the plot title.

`props`: A structure with any optional properties.

`quiet`: If 1, only use given title_str.

`skipStep`: Number of voltage steps to skip at the start (default=0).

Returns:

a_p: A plot_abstract object.

Example:

```
» plotFigure(plotAllIVs(data_vc, 'I/V curves'))
```

See also: `model_data_vcs` (p. 115), `voltage_clamp` (p. 338), `plot_abstract` (p. 180), `plotFigure` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2011/07/16

A.56.9 Method `voltage_clamp/plotSimCurrent`

Summary: Simulate voltage clamp current on a model channel and superpose on data.

Usage:

```
a_p = plotSimCurrent(a_vc, f_I_v, props)
```

Parameters:

`a_vc`: A voltage_clamp object.

`f_I_v`: param_func object representing the model channel.

`props`: A structure with any optional properties.

`delay`: If given, use as voltage clamp delay [ms].

`levels`: Only plot these voltage level indices.

Returns:

a_p: A plot_abstract object.

Example:

```
» plotFigure(plotSimCurrent(a_vc))
```

See also: `param_I_v` (p. **??**), `param_func` (p. **??**), `plot_abstract` (p. 180)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/03/29

A.56.10 Method `voltage_clamp/plotSteadyIV`

Summary: Plot of the I/V curve at the end of a voltage step.

Usage:

```
a_p = plotSteadyIV(a_vc, step_num, title_str, props)
```

Description: Can be superposed with other I/V plot objects (see plot_superpose).

Parameters:

`a_vc`: A voltage clamp object.

`step_num`: 1 for prestep, 2 for the first step, 3 for next, etc.

`title_str`: (Optional) Text to appear in the plot title.

`props`: A structure with any optional properties.

`quiet`: If 1, only use given title_str.

`curUnit`: Display units for current trace (default='nA').

`label`: add this as a line label to be used in superposed plots.

`plotPeaks`: If 1, use the props.iPeaks instead of steady-state.

`stepRange`: Uses the relative [start end] times in [ms] around time of step_num to calculate the current averages. If vector has 3 items, first one is the step number, which can be different than step_num.

Returns:

a_p: A plot_abstract object.

Example:

```
» a_vc = abf_voltage_clamp('data-dir/cell-A.abf')
» plotFigure(plotSteadyIV(a_vc, 2, 'I/V curve'))
```

See also: `voltage_clamp` (p. 338), `plot_abstract` (p. 180), `plotFigure` (p. **??**), `plot_superpose` (p. 196)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/03/10

A.56.11 Method `voltage_clamp/plot_abstract`

Summary: Plot of the I and V traces of voltage clamp object.

Usage:

```
a_p = plot_abstract(a_vc, title_str, props)
```

Description: Can be stacked or superposed with other plot objects.

Parameters:

`a_vc`: A voltage clamp object.

`title_str`: (Optional) Text to appear in the plot title.

`props`: A structure with any optional properties.

`quiet`: If 1, only use given title_str.

`vStep`: Index of step with varying voltages (default=2).

`label`: add this as a line label to be used in superposed plots.

`onlyPlot`: 'i' for current and 'v' for voltage plot.

`curUnit`: Display units for current trace (default='nA').

`vColors`: If 1 (default), always use same colors for same voltage levels.

`vColorsFunc`: Function to get voltage colors (default=@lines) (rest passed to plot_stack and plot_abstract)

Returns:

a_p: A plot_abstract object.

Example:

```
» a_vc = abf2voltage_clamp('data-dir/cell-A.abf')
» plotFigure(plot_abstract(a_vc, 'I/V curve'))
```

See also: plotSteadyIV (p. **??**), plot_abstract (p. 180), plotFigure (p. **??**), plot_superpose (p. 196), plot_stack (p. 194)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/03/11

A.56.12 Method voltage_clamp/saveDataTxt

Summary: Save time and current into a simple text file as columns.

Usage:

```
saveDataTxt(a_vc, props)
```

Description: File will be written to the same directory as the original vc was loaded from (using the value of props.filename).

Parameters:

a_vc: A voltage_clamp object.

props: A structure with any optional properties.

addName: String to append to file name.

saveV: Save voltage as well.

Returns:

Example:

```
» saveDataTxt(a_vc)
```

See also: voltage_clamp (p. 338)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/03/29

A.56.13 Method voltage_clamp/scale_IClCa_NaP_sub_IClCa

Summary: Scale IClCa and steady-state of INaP from voltage-step protocol data and subtract IClCa.

Usage:

```
params = scale_IClCa_NaP_sub_IClCa(a_vc, props)
```

Description: Made for Na recordings from the oocyte. While estimating IClCa, one needs to consider INaP since it counteracts IClCa.

Parameters:

a_vc: A voltage clamp object.

props: A structure with any optional properties.

`saveData`: If 1, save subtracted data into a new text file (default=0).

`plotPrepulse`: If 1, show plot of prepulse current change with voltage (default=0).

Returns:

params: Structure with tuned parameters.

Example:

```
» a_vc = abf_voltage_clamp('data-dir/cell-A.abf')
» params = scale_IClCa_NaP_sub_IClCa(a_vc)
```

See also: `voltage_clamp` (p. 338), `param_I_v` (p. **??**), `param_func` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/02/05

A.56.14 Method `voltage_clamp/scale_sub_cap_leak`

Summary: Scale capacitance and leak artifacts to subtract them.

Usage:

```
[f_capleak sub_vc] = scale_sub_cap_leak(a_vc, props)
```

Parameters:

`a_vc`: Full path to a_vc.

`props`: A structure with any optional properties.

`capLeakModel`: Model object to fit (default obtained from param_cap_leak_int_t). Can choose object obtained from another

`function such as`: param_Rs_cap_leak_int_t, param_cap_leak_2comp_int_t.

`fitRange`: Start and end times of range to apply the optimization [ms].

`fitRangeRel`: Start and end times of range relative to first voltage step [ms]. Specify any other voltage step as the first element.

`fitLevels`: Indices of voltage/current levels to use from clamp data. If empty, not fit is done.

`dispParams`: If non-zero, display params every once this many iterations.

`dispPlot`: If non-zero, update a plot of the fit at end of this many iterations.

`saveData`: If 1, save subtracted data into a new text file (default=0).

`quiet`: If 1, do not include cell name on title.

`period`: Limit the subtraction to this period of a_vc.

Returns:

f_capleak: Updated function with fitted parameters sub_vc: voltage_clamp object with passive-subtracted I trace.

Example:

```
» capleakReCe_f = ...
param_Re_Ce_cap_leak_int_t(...
struct('Re', 47, 'Ce', 12, 'gL', .56, ...
'EL', -67, 'Cm', 270, 'delay', 0.21), ...
cap, leak, Re and Ce');
» [capleakReCe_f sub_cap_leak_vc ] = ...
scale_sub_cap_leak(...
abf2voltage_clamp('calcium.abf'), '', ...
struct('capLeakModel', capleakReCe_f, ...
'fitRangeRel', [-.2 165], 'fitLevels', 1:5, ...
'optimset', struct('Display', 'iter')));
```

See also: param_I_v (p. **??**), param_func (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/01/17

A.56.15 Method voltage_clamp/set

Summary: Generic method for setting object attributes.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/08

A.56.16 Method voltage_clamp/setLevels

Summary: Choose which voltage and current step levels to keep.

Usage:

```
a_vc = setLevels(a_vc, levels, props)
```

Parameters:

a_vc: A voltage_clamp object.

levels: Only keep these voltage and current level indices.

props: A structure with any optional properties.

Returns:

a_vc: A voltage_clamp object that contains only the selected levels.

Example:

```
» a_vc = setLevels(a_vc, 1:3)
```

See also: voltage_clamp (p. 338)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/03/30

A.56.17 Method `voltage_clamp/simModel`

Summary: Simulate model channel current using voltage clamp.

Usage:

```
model_vc = simModel(a_vc, f_I_v, props)
```

Description: Often the delay is already included in the model, which is better because sub-dt precision can be achieved using interpolation.

Parameters:

`a_vc`: A voltage_clamp object.

`f_I_v`: param_func object representing the model channel.

`props`: A structure with any optional properties.

`delay`: If given, use as voltage clamp delay [ms].

`levels`: Only simulate these voltage level indices.

`period`: Limit the simulation to this period of a_vc.

`updateVm`: If 1, update v trace from simulation Vm.

Returns:

model_vc: A voltage_clamp object with simulated current data and the original voltage data.

Example:

```
» I_Ca = param_I_v([1 1 .0077 58], m_Ca, h_Ca, 'I_Ca', ...
struct('paramRanges', ...
4; 0 1; 0 1e3; 100 200'))
» model_vc = simModel(a_vc, I_Ca)
```

See also: `param_I_v` (p. **??**), `param_func` (p. **??**), `plot_abstract` (p. 180)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/03/29

A.56.18 Method `voltage_clamp/subsref`

Summary: Defines generic indexing for objects.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/04

A.56.19 Method `voltage_clamp/updateSteps`

Summary: Update voltage step time and magnitude info.

Usage:

```
a_vc = updateSteps(a_vc, props)
```

Description: Called by simModel.

Parameters:

`a_vc`: A voltage_clamp object.

`props`: A structure with any optional properties.

Returns:

a_vc: Updated object.

See also: `voltage_clamp` (p. 338)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/10/18

A.56.20 Method `voltage_clamp/withinPeriod`

Summary: Returns a voltage_clamp object valid only within the given period.

Usage:

```
[a_vc a_period] = withinPeriod(a_vc, a_period, props)
```

Parameters:

`a_vc`: A voltage_clamp object.

`a_period`: The desired period [dt].

`props`: A structure with any optional properties.

`useAvailable`: If 1, don't stop if period not available, use maximum available.

Returns:

a_vc: A voltage_clamp object a_period: The period object, updated if useAvailable is requested.

See also: `trace/withinPeriod` (p. 336), `trace` (p. 317), `period` (p. 162)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/03/29

A.57 Utility functions

A.57.1 Function `functions/abf2load`

A.57.2 Function `functions/abf2voltage_clamp`

Summary: Load I and V traces from an ABF file and make a voltage_clamp object.

Usage:

```
a_vc = abf2voltage_clamp(filename, sup_id, props)
```

Parameters:

`filename`: Full path to filename.

`sup_id`: (Optional) Concatenated to cell filename as id of voltage_clamp object.

`props`: A structure with any optional properties.

`ichan`: Current channel number or ':' for all channels when there is more than one to choose.

`actualProtocols`: Means current trace is a TTL pulse and its magnitude is meaningless.

Returns:

a_vc: A voltage_clamp object.

Example:

```
» a_vc = abf2voltage_clamp('data-dir/cell-A.abf', 'my first cell');
» plot(a_vc);
```

See also: `loadVclampAbf` (p. **??**), `abf2load` (p. **??**), `voltage_clamp` (p. 338)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/03/11

A.57.3 Function `functions/abfload`

A.57.4 Function `functions/array2str`

Example:

```
string = array2str([1 2 3 4 6 8 10 12 13])
the output is '[1:4 6:2:12 13]'
```

A.57.5 Function `functions/balanceInputProbs`

Summary: Balances samples according to prior class probabilities of the outputs.

Usage:

```
[new_inputs, new_outputs] = balanceInputProbs(a_class_inputs, a_class_outputs, balance_ratio, props)
```

Description: Uses the method in Lawrence, burns, Back, Tsoi and Lee Giles "Neural network classification and prior class probabilities" for probabilitic balancing of input and output samples when the number of samples in each class is vastly different and causes problems with classification without balancing.

Parameters:

`a_class_inputs, a_class_outputs`: Input and output vectors.

`balance_ratio`: c_s, between 0 and 1. If 1, equal samples from each class if used. If 0, prior class probabilities are followed.

`props`: A structure with any optional properties.

`repeatSamples`: If 1, repeats the smaller class samples to match with larger class. Otherwise, takes the minimal number of samples to avoid repetitions (default=1).

Returns:

new_inputs, new_outputs: New input and output vectors.

See also: `approxMappingNNet` (p. **??**), `tests_db` (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2008/01/09

A.57.6 Function `functions/boxplotp`

Description: BOXPLOTP(X,NOTCH,SYM,VERT,WHIS,PROPS) produces a box and whisker plot for each column of X. The box has lines at the lower quartile, median, and upper quartile values. The whiskers are lines extending from each end of the box to show the extent of the rest of the data. Outliers are data with values beyond the ends of the whiskers.

A.57.7 Function `functions/boxutilp`

Description: BOXUTILP(X) is a utility function for BOXPLOT, which calls BOXUTILP once for each column of its first argument. Use BOXPLOT rather than BOXUTILP.

A.57.8 Function `functions/calcGraphNormPtsRatio`

Summary: Return the ratios of normalized to point units for dimensions of axis.

Usage:

```
[ratio_x, ratio_y] = calcGraphNormPtsRatio(grfx_handle)
```

Description: Used for findind character sizes given the size of an axis. Normally if the plot is resized, the characters may take up too much space or may not fit anymore unless the spacing is corrected.

Parameters:

`grfx_handle`**:** A graphics handle to an existing axis or figure.

Returns:

ratio_x, ratio_y: Normalized to points ratio for axis width and height, respectively.

Example:

```
To find the normalized distance for a 10pt character:
» dx = 10 * calcGraphNormPtsRatio(my_figure_handle);
```

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/03/05

A.57.9 Function `functions/cell2str`

Summary: Creates a tab-delimited string from the cell array's contents.

Usage:

```
a_str = cell2str(a_cell, props)
```

Parameters:

`a_cell`**:** A cell matrix to be tabularized.

`props`**:** A structure with any optional properties.

Returns:

a_str: LaTeX formatted table string.

See also:

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/12/09

A.57.10 Function `functions/cell2TeX`

Summary: Creates LaTeX string of a formatted table with the cell array's contents.

Usage:

```
tex_string = cell2TeX(a_cell, props)
```

Parameters:

`a_cell`: A cell matrix to be tabularized.

`props`: A structure with any optional properties.

`hasTitleRow`: The first row contains titles.

`titleColWidth`: If specified, makes title cells \parbox'es with the given width.

`hasTitleCol`: The first column contains titles.

`numFormat`: Specify a sprintf-style format for displaying numbers.

Returns:

tex_string: LaTeX formatted table string.

See also:

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/12/09

A.57.11 Function `functions/chanTables2DB`

Summary: Creates a DB with channel tables exported from Genesis.

Usage:

```
a_chans_db = chanTables2DB(tables, id, props)
```

Parameters:

`tables`: Structures returned from the dump files generated by dump_chans.g.

`id`: String that identify the source of the tables structure.

`props`: A structure with any optional properties.

(rest passed to tests_db.)

Returns:

a_chans_db: A tests_db object containing channel tables.

See also: `trace` (p. 317), `trace/plot` (p. 328), `plot_abstract` (p. 180), `GP/common/dump_chans.g (Genesis)` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2007/03/07

A.57.12 Function `functions/collectspikes`

Author: <adelgado@biology.emory.edu>

A.57.13 Function `functions/colormapBlueCrossRed`

Summary: Blue to red crossing colormap, with a black-colored zero-crossing.

Usage:

```
colors = colormapBlueCrossRed(num_half_colors)
```

Description: Colormap contains (2 * num_half_colors + 1) colors, where (num_half_colors + 1) is the zero crossing.

Parameters:

`num_half_colors`**:** Number of colors to generate on one of the red or blue scales.

`props`**:** A structure with any optional properties.

Returns:

colors: RGB color matrix to be passed to colormap function.

See also: `colormap` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/06/05

A.57.14 Function `functions/defaultValue`

Summary: If variable unset (either nonexistent or empty), assign it a default value. Otherwise the variable remains unchanged.

Usage:

```
var = defaultValue(varname, a_defaultvalue)
```

Description: If the variable has already been defined, it keeps unchanged. If the variable doesn't exist or is an empty matrix, it will be assigned a default value to it.

Parameters:

`varname`**:** a string. the name of the variable.

`a_defaultvalue`**:** value for the variable.

Example:

```
SamplingRate = defaultValue('SamplingRate', 10);
```

Author: Li, Su

A.57.15 Function `functions/diff2T`

Summary: Estimate of second derivative using Taylor expansion.

Usage:

```
deriv2 = diff2T(x, dy)
```

Description: d^2 x - x(k-2) + 16 * x(k-1) - 30 * x(k) + 16 * x(k+1) - x(k+2) ——- = ———————————————————————————— dy^2 12 * dy^2

Parameters:

`x`: A vector of x = f(y).

`dy`: The resolution of the discrete points in the vector.

Returns:

deriv2: Estimate of the derivative.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/15

A.57.16 Function `functions/diff2T_h4`

Summary: Estimate of second derivative using Taylor expansion (derived with same method as diffT).

Usage:

```
deriv2 = diff2T_h4(x, dy)
```

Description: d^2 x x(k-2) - x(k-1) - x(k+1) + x(k+2) ——- = ———————————— dy^2 6 * dy^2

Parameters:

`x`: A vector of x = f(y).

`dy`: The resolution of the discrete points in the vector.

Returns:

deriv2: Estimate of the derivative.

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/04/15

A.57.17 Function `functions/diff3T`

Summary: Estimate of third derivative using Taylor expansion.

Usage:

```
deriv3 = diff3T(x, dy)
```

Description: d^3 x x(k-3) - 8 * x(k-2) + 13 * x(k-1) - 13 * x(k+1) + 8 * x(k+2) - x(k+3) ——- = ———————————————————————————— dy^3 8 * dy^3

Parameters:

`x`: A vector of x = f(y).

`dy`: The resolution of the discrete points in the vector.

Returns:

deriv3: Estimate of the derivative.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/15

A.57.18 Function `functions/diff3T_h4`

Summary: Estimate of third derivative using Taylor expansion (derived with same method as diffT and diff2T_h4).

Usage:

```
deriv2 = diff3T_h4(x, dy)
```

Description: d^3 x - x(k-2) + 2 * x(k-1) - 2 * x(k+1) + x(k+2) ——- = ———————————————— dy^3 12 * dy^3

Parameters:

`x`: A vector of x = f(y).

`dy`: The resolution of the discrete points in the vector.

Returns:

deriv2: Estimate of the derivative.

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/04/18

A.57.19 Function `functions/diffT`

Summary: Estimate of first derivative using Taylor expansion.

Usage:

```
deriv = diffT(x, dy)
```

Description: dx x(k-2) - 8 * x(k-1) + 8 * x(k+1) - x(k+2) —- = ———————————————— dy 12 * dy

Parameters:

`x`: A vector.

`dy`: The resolution of the discrete points in the vector.

Returns:

deriv: Estimate of the first derivative.

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/11/15

A.57.20 Function `functions/fillederrorbar`

Summary: Plots an errorbar with the middle points filled with the pen color.

Usage:

```
handles = fillederrorbar(...)
```

Parameters:

(see errorbar)

Returns:

handles: Handles to graphics objects.

See also:

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/13

A.57.21 Function `functions/findspikes`

Author: Li, Su based on the original of Alfonso Delagado-Reyes

---

A.57.22 Function `functions/findspikes_old`

Author: <adelgado@biology.emory.edu>, 2003-03-31

A.57.23 Function `functions/findVectorInMatrix`

Summary: Finds rows of data that match row.

Usage:

```
idx = findVectorInMatrix(data, row)
```

Description: Matlab's eq (==) command unfortunately doesn't allow this directly.

Parameters:

`data`**:** A matrix or column vector.

`row`**:** A row vector.

`Returns`**:**

`idx`**:** Indices of matching rows in the original data matrix.

See also:

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/09/1

A.57.24 Function `functions/getFieldDefault`

Summary: Get a field from a struct and return the default_value if field does not exist.

Usage:

```
value = getFieldDefault(structure, fieldname, default_value)
```

Parameters:

`fieldname`**:** Field name.

`default_value`**:** Value to return if field doesn't exist.

Author: Li, Su - 2007

A.57.25 Function `functions/getfuzzyfield`

Author: Li, Su - 2007

A.57.26 Function `functions/gettracelist2`

Summary: Gets a list of the form: '1 3 7-10' and returns a column vector with the traces numbers, and the number of traces.

Usage:

```
[traces, ntraces] = gettracelist2('list');
```

Description: Please note the space between single traces and the dash for ranges of traces.

Author: <adelgado@biology.emory.edu>

A.57.27 Function `functions/growRange`

Summary: Returns the maximal range from rows of axis limits.

Usage:

```
range = growRange(ranges)
```

Parameters:

`ranges`: A matrix where each row is return val of axis func.

Returns:

range: The maximal range obtained that includes all given axes.

See also:

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/10/13

A.57.28 Function `functions/interpValByIndex`

Summary: Finds the interpolated value by using the real valued index from the data vector.

Usage:

```
val = interpValByIndex(idx, data)
```

Description: Parameters: idx: A real-valued index. data: A data vector.

Returns:

val: the value taken from the nearest integer indices of data and interpolated.

Example:

```
» a= [1 2 3];
» interpVal ByIndex(1.5, a)
ans =
1.5000
```

See also: spike_shape (p. 215)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/08/02

A.57.29 Function functions/loadtraces

Parameters:

`file`: PCDX file.

`tracelist`: A string of trace description, such as '1-10'.

`channel`: Channel to read from.

`quiet`: (Optional) If 1, produce on print outs.

Author: <adelgado@biology.emory.edu>

A.57.30 Function functions/loadVclampAbf

Summary: Load I and V traces from an ABF file.

Usage:

```
[time, dt, data_i, data_v, cell_name] = loadVclampAbf(filename, props)
```

Description: If filename is wrong or not specified, a dialog will pop up to choose file. ABF2 files are not fully supported (see abf2load.m). Time is assumed to be in s and converted to ms.

Parameters:

`filename`: Full path to filename.

`props`: A structure with any optional properties.

`scaleI`: multiplier to correct I values to the units specified in file.

`actualProtocols`: Means current trace is a TTL pulse and its magnitude is meaningless.

Returns:

time: Time vector for measurements [ms], dt: Time step [ms], data_i: Current traces (assumed [nA]), data_v: Voltage traces (assumed [mV]), cell_name: Extracted from the file name part of the path.

Example:

```
» [time, dt, data_i, data_v, cell_name] = ...
loadVclampAbf('data-dir/cell-A.abf')
» plotVclampStack(time, data_i, data_v, cell_name);
```

See also: abf2load (p. **??**), plotVclampAbf (p. **??**), plotVclampStack (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2009/12/17

A.57.31 Function functions/logLevels

Summary: Returns a logarithmic-scaled series between min_val and max_val with num_levels elements.

Usage:

```
levels = logLevels(min_val, max_val, num_levels)
```

Parameters:

`min_val, max_val`: The low and high boundaries for the output value.

`num_levels`: Number of elements to produce, including the boundaries.

Returns:

levels: A column vector of logarithmic series between min_val and max_val.

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/04/18

A.57.32 Function functions/makeIdealClampV

Summary: Make voltage traces that mimic an ideal voltage clamp.

Usage:

```
[trace_v] = makeIdealClampV(t_vals, pre_v, pulse_v, post_v, dt, id, props)
```

Parameters:

`t_vals`: Vector with times of pulse start, end and trace end [ms].

`pre_v, post_v`: Holding and final voltage values [mV].

`pulse_v`: Vector of variable voltage steps [mV].

`dt`: Resolution of time in trace produced [ms].

`id`: An identifying string.

`props`: A structure with any optional properties.
(Rest passed to trace)

Returns:

trace_v: A trace object with the voltage traces.

Example:

```
» tr_v = makeIdealClampV([10 100 110], -90, -80:10:60, -10, 1e-1, ...
'Na chan voltage clamp protocol')
» plot(tr_v)
» vc_test = voltage_clamp(data_i, tr_v.data, tr_v.dt, 1e-9, 'sim Na data')
```

See also: trace (p. 317), voltage_clamp (p. 338)

Author: Cengiz Gunay <cgunay@emory.edu>, 2010/10/07

A.57.33 Function functions/makeIdx

Summary: Prepare the idx structure from names.

Usage:

```
idx = makeIdx(names)
```

Description: Helper function.

Parameters:

names: Cell array of names for a db dimension.

Returns:

idx: Structure associating names to array indices.

See also: tests_db (p. 256)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/17

A.57.34 Function functions/maxima

Summary: Find all local maxima.

Usage:

```
x_idx = maxima(x)
```

Description: Finds derivative sign-flipping points where the second derivative is less than zero.

Parameters:

x: A vector.

Returns:

x_idx: Indices of maxima.

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/04/18

A.57.35 Function `functions/meanSpikeFreq`

Summary: Returns the mean firing frequency in Hz according to mean \ inter-spike-interval of the given spike train and the time resolution dt.

Usage:

```
meanFreq = meanSpikeFreq( spike_train, dt, period )
```

Description: Parameters: spike_train: Spike times returned by findspikes dt: Time step size [s] period: Duration of the total time period [dt]

See also:

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/03/08

A.57.36 Function `functions/mergeStructs`

Summary: Merges all the structures given as arguments and makes a single structure.

Usage:

```
results = mergeStructs( struct1 [, struct2, ...]  )
```

Description: The fields will in earlier arguments will have priority. So, while merging two structs, if there are duplicate fields, the fields in the first will be preserved.

Parameters:

struct(n): A structure.

Returns:

results: The merged structure.

Example:

```
mergeStructs( struct('hello', 1), struct('bye', 2) );
=> struct('hello', 1, 'bye', 2)
```

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/13

A.57.37 Function `functions/mergeStructsRecursive`

Summary: Merges given structures into a single structure, merging substructures recursively.

Usage:

```
results = mergeStructsRecursive( struct1 [, struct2, ...] )
```

Description: The fields will in earlier arguments will have priority. So, while merging two structs, if there are duplicate fields, the fields in the first will be preserved. If a common field is a structure, then mergeStructsRecursive is called to merge their contents.

Parameters:

struct(n): A structure.

Returns:

results: The merged structure.

Example:

```
» mergeStructsRecursive( struct('hello', struct('a', 1),
struct('hello', struct('b', 2)) );
=> struct('hello', struct('a', 1, 'b', 2)
```

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/13

A.57.38 Function `functions/ns_CIPlist`

Author: Dawid Kurzyniec

A.57.39 Function `functions/ns_load_tracesets`

Summary: Return a set of physiol_cip_traceset objects loaded from a single NeuroSAGE HDF5 file.

Usage:

```
a_tss = ns_load_tracesets(data_src, props)
```

Description: This allows customized loading each NeuroSAGE file separately. Only loads traces that has the word 'cip' or 'spont' in the NeuroSAGE sequence name. Sample rate, channel gain and dy values are read from the acquisition data.

Parameters:

`data_src`**:** A pattern for one or more NeuroSAGE filename or structure output of ns_open_file.

`props`: A structure with any optional properties.

`VmChan`: (Optional) If a string, read voltage trace from channel having this string (e.g., 'Amp1 Vm'). If numeric, use as channel number. Added to the neuron_id to distinguish multiple neurons recorded in same file. If not specified, the first voltage channel is used.

`ImChan`: (Optional) Similar to VmChan for reading current trace. Does not affect neuron_id.

`VGain, IGain`: for HDF5 files, these two fields only works as a default value when the gains are not specified in the file.

`IncludeSeq`: A string or cell array of strings specifying keywords in sequence name to look for.

`ExcludeSeq`: A string or cell array of strings specifying keywords in sequence name to avoid searching.

`addTreats`: Structure of default treatment names and their values for this traceset to keep consistent accross tracesets. Use only lowercase in treatment names.

`fixTreats`: Override wrong treatment information with these. Same format as addTreats.

`renameTreats`: Structure with from->to rename pairs.

`trials`: A vector of trials to load from the file. All others are skipped. (All other props are passed to physiol_cip_traceset)

Returns:

a_tss: Cell array of physiol_cip_traceset objects.

See also: `physiol_cip_traceset_fileset` (p. 175), `physiol_cip_traceset` (p. 170), `params_tests_dataset` (p. 132)

Author: Li, Su; Cengiz Gunay <cgunay@emory.edu>; and Jeremy Edgerton, 2007/12/18

A.57.40 Function `functions/parseFilenameNamesVals`

Summary: Parses filename to extract names and values of parameters.

Usage:

```
names_vals = parseFilenameNamesVals(filename, props)
```

Description: Parses the given string (e.g., filename) that has names and values separated by underscores (_).

Parameters:

`filename`: file name (no need to exist)

`props`: Structure with optional properties:

`namesFirst`: If 1, names precede values (default=1).

`skipNum`: Number of words to skip (default=0). If -1, all words are skipped until a number is found. this makes sense when namesFirst=0.

Returns:

names_vals: A two-column cell array with names and values. pre_name: (Optional) Skipped prefix words in the filename.

Example:

```
Names first:
» nv = parseFilenameNamesVals('hello_boys_6_girls_4.txt', struct('skipNum', 1))
nv =
'girls' [6]
'boys' [4]
Same result with values first:
» nv = parseFilenameNamesVals('data/hello_6_girls_4_boys.txt',
struct('namesFirst', 0, 'skipNum', -1))
```

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/03/10

A.57.41 Function `functions/parseGenesisFilename`

Summary: (OBSOLETE, see parseFilenameNamesVals) Parses the GENESIS filename to get names and values of simulation parameters. Usage: names_vals = parseGenesisFilename(filename)

Description: Parameters: filename: GENESIS filename (no need to exist)

Returns:

names_vals: A two-column cell array with names and values.

See also: `parseFilenameNamesVals` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/03/10

A.57.42 Function `functions/plotColormap`

Summary: Colormap plot that requires displaying a colorbar.

Usage:

```
h = plotColormap(data, a_colormap, num_colors, props)
```

Description: Mainly serves to format the colorbar, which can only be plotted after we prepare the main axis. Thus it cannot be easily integrated into plot_stack.

Parameters:

`data`: 2D matrix with image data or cell array to be passed as arguments to arbitrary plot command (see props).

`a_colormap`: Colormap vector, function name or handle to colormap (e.g., 'jet').

`num_colors`: Parameter to be passed to the a_colormap.

`props`: A structure with any optional properties.

`command`: Plot command to interpret data (default='image').

`colorbar`: If given, show colorbar on plot.

`colorbarProps`: Set colorbar axis properties.

`colorbarLabel`: Set colorbar y-axis label.

`truncateDecDigits`: Truncate labels to this many decimal digits.

`minValue,maxValue`: Minimal and maximal values represented by 1, num_colors to annotate the colorbar, resp.

`reverseYaxis`: If 1, display y-axis values in reverse (default=0).

Returns:

h: Handle to main plot object (e.g., image).

See also: `colormap` (p. **??**), `colorbar` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/06/05

A.57.43 Function `functions/prefixStruct`

Summary: Adds the given prefix to each of the field names in the structure.

Usage:

```
new_struct = prefixStruct(a_struct, prefix_str)
```

Parameters:

`a_struct`: A structure.

`prefix_str`: A string to be prefixed to each field name.

Returns:

new_struct: The new structure.

Example:

```
prefixStruct( struct('bye', 1), 'hello');
=> struct('hellobye', 1)
```

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/12/22

A.57.44 Function `functions/properAlphaNum`

Summary: Replaces characters in string to make it only alphanumeric.

Usage:

```
a_label = properAlphaNum( a_label )
```

Description: It will only keep the character set 'A-Z a-z 0-9 _'. It will also prepend 'a_' if the label starts with a number.

Parameters:

`a_label`**:** A label string.

Returns:

a_label: The corrected proper a_label.

Example:

```
» a_label = properAlphaNum('to \this _day+1 and ^5')
ans = 'tothis_day1and5'
```

Author: Cengiz Gunay <cgunay@emory.edu>, 2011/01/19

A.57.45 Function `functions/properTeXFilename`

Summary: Replaces characters in string to make it a valid filename for inclusion in TeX documents.

Usage:

```
filename = properTeXFilename( filename )
```

Description: It will replace characters like space, '/', '.', etc.

Parameters:

`filename`**:** An input filename string (without extension!).

Returns:

filename: The corrected proper filename.

Example:

```
» fname = properTeXFilename('hello world/1')
ans = 'hello_world+1'
```

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/12/20

A.57.46 Function `functions/properTeXLabel`

Summary: Replaces characters in string or cell array of strings to make it valid in TeX documents.

Usage:

```
a_label = properTeXLabel( a_label )
```

Description: It will replace characters like space, '/', '.', etc.

Parameters:

`a_label`: A label string.

Returns:

a_label: The corrected proper a_label.

Example:

```
» a_label = properTeXLabel('this_day')
ans = 'this\_day'
```

Author: Cengiz Gunay <cgunay@emory.edu>, 2006/01/17

A.57.47 Function `functions/readgenbin`

Summary: Reads a time-range of data from a binary GENESIS file.

Usage:

```
[data, time_trace] = readgenbin(filename, start_time, end_time, endian);
```

Description: Files should be created by the disk_out method in the GENESIS neural simulator. No checking for binary type is made, so if you want reliability please ensure the file is a binary. Files written by GENESIS on big-endian machines (like old Mac and Solaris machines with PowerPC architecture) must be loaded with the endian='b' option. There are sanity checks to flag that the file may be reverse-endian, but this is not automatically corrected. Runs faster is you don't request the time_trace output.

Parameters:

`filename`: Path to GENESIS file.

`start_time, end_time`: Time in milliseconds relative to the ACTUAL time of the experiment at which data adquisition started (if you start gathering data at 200 ms and you specify 0 start time it will not work). If either is [] or NaN, defaults to beginning and end of trace, respectively. end_time is not inclusive.

`endian`: (optional) Indicates file format; 'l' for little endian and 'b' for big endian. See the "machineformat" option in fopen for more information. Defaults to the native endian of this computer.

Returns:

data: Data vector or matrix read. time_trace: (Optional) Corresponding time range vector (in ms).

Example:

```
Fully read a native-endian file:
» dat = readgenbin('mydir/myfile.bin');
Specify a time range:
» dat = readgenbin('mydir/myfile.bin', 100, 1000);
Get a time vector back:
» [dat, t] = readgenbin('mydir/myfile.bin', 100, 1000);
» figure; plot(t, dat);
Force to fully load big-endian Mac file on PC platform:
» dat = readgenbin('mydir/mymacfile.bin', NaN, NaN, 'b');
```

See also: fopen (p. **??**)

Author: Alfonso Delgado-Reyes original version based in open

A.57.48 Function `functions/readNeuronVecAscii`

Summary: Reads Neuron simulator Vector object data from ascii files.

Usage:

```
[data, label] = readNeuronVecAscii(filename)
```

Description: It's one line of code just to read the data: dlmread(filename, '\t', 2, 0)

Parameters:

`filename`: Full path to Neuron file.

Returns:

data: Row vector with two columns of data. label: String denoting Vector contents.

Example:

```
data = readNeuronVecAscii('myvec.dat');
```

Author: Cengiz Gunay <cengique@users.sf.net> 2012/03/02

A.57.49 Function `functions/readNeuronVecBin`

Author: Konstantin Miller <miller@cs.tu-berlin.de>, Aug 09, 2005.

A.57.50 Function `functions/renameIdx`

Summary: Rename one or more items in a database dimension (rows, columns, etc).

Usage:

```
new_idx = renameIdx(old_idx, old_names, new_names)
```

Description: Prefer the convenience methods in tests_db (renameColumns, renameRows) and tests_3D_db (renamePages). This is a cheap operation than modifies meta-data kept in object. For the regular expression renaming, the old_names and new_names parameters are passed to the regexprep command after removing the delimiting slashes (//). At least one grouping construct ('()') must be used in the search pattern such that it can be used in the replacement pattern (e.g., '$1'). See example above.

Parameters:

`old_idx`: An indexing structure (a_db.col_idx for columns, etc).

`old_names`: A cell array of existing names, array of numerical indices, or a regular
expression denoted between slashes (e.g., '/(.*)/').

`new_names`: New names to replace existing ones OR regular expression
replace string (no slashes, e.g, '$1_test'). See regexprep command.

Returns:

a_db: The tests_db object that includes the new names.

Example:

```
» a_db.col_idx = renameIdx(a_db.col_idx, 'PulseIni100msSpikeRateISI_D40pA', 'Firing_rate');
» a_db.col_idx = renameIdx(a_db.col_idx, 1, 'Firing_rate');
» a_db.col_idx = renameIdx(a_db.col_idx, '/(.*)/', '$1_old');
» a_db.col_idx = renameIdx(a_db.col_idx, 'a', 'b', 'c', 'd');
» a_db.col_idx = renameIdx(a_db.col_idx, [1, 2], 'c', 'd');
```

See also: `tests_db/renameColumns` (p. 302), `tests_db/renameRows` (p. 303), `tests_3D_db/renamePages` (p. 254), `regexprep` (p. **??**), `allocateRows` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2017/06/09

A.57.51 Function `functions/setAxisNonNaN`

Summary: Returns the limits of the current axis replaced with the non-NaN elements of the given vector.

Usage:

```
new_axis = setAxisNonNaN(layout_axis)
```

Parameters:

`layout_axis`**:** The axis position to layout this plot.

Returns:

new_axis: Modified axis.

Example:

```
» axis(setAxisNonNaN([0 100 NaN NaN]))
```

See also: `plot_abstract` (p. 180)

Author: Cengiz Gunay <cgunay@emory.edu>, 2007/10/29

A.57.52 Function `functions/sortedUniqueValues`

Summary: Find unique rows in an already sorted matrix (or column vector).

Usage:

```
[rows, idx] = sortedUniqueValues(data)
```

Description: Uses the derivation by Matlab diff function method. Redundant with the Matlab function UNIQUE doing the same job: [rows, idx]= unique(data, 'rows', 'first'). However, sortedUniqueValues is more efficient if the input data is already sorted for some other reason (see usage in tests_db/invarValues).

Parameters:

`data`**:** A ascending row-sorted matrix or column vector.

Returns:

rows: A matrix or column vector of unique rows. idx: Indices of the unique rows in the original data matrix.

See also: `uniqueValues` (p. **??**), `unique` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/27

A.57.53 Function `functions/string2File`

Summary: Writes string verbatim into a file.

Usage:

```
string2File(string, filename, props)
```

Parameters:

`string`: To be written into file.

`filename`: The file to be created.

`props`: A structure with any optional properties.

`append`: If 1, append to existing file.

Returns:

See also: `cell2TeX` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/12/10

A.57.54 Function `functions/struct2DB`

Summary: Converts a structure array to a tests_db object.

Usage:

```
a_tests_db = struct2DB(a_struct, props)
```

Description: Field names become column names in the DB.

Parameters:

`a_struct`: A structure to convert.

`id`: Optional database id string.

`props`: A structure with any optional properties, passed to tests_db.

Returns:

a_tests_db: A tests_db object.

See also:

Author: Cengiz Gunay <cgunay@emory.edu>, 2008/01/11

A.57.55 Function `functions/struct2str`

Summary: Converts numerical structure into a single-line string.

Usage:

```
a_str = struct2str(a_struct, props)
```

Parameters:

`a_struct`: Structure that has names pointing to numerical values.

`props`: A structure with any optional properties.

Returns:

a_str: A string that contains structure fields and value like in 'name1_val1_name2_val2_...'

See also: `cell2TeX` (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2011/05/17

A.57.56 Function `functions/subTextLabel`

Summary: Draws a text label on a plot.

Usage:

```
handle = subTextLabel(x, y, text_str, props)
```

Parameters:

`x, y`: 2D coordinates.

`text_str`: String to be drawn on plot.

`props`: A structure with any optional properties.

`Units`: position units for the coordinates (see Units in axes properties).

Returns:

handle: Text object handle.

Author: Cengiz Gunay <cgunay@emory.edu>, 2005/04/11

A.57.57 Function `functions/TeXfloat`

Summary: Places LaTeX content into a float (e.g., table, figure).

Usage:

```
tex_string = TeXfloat(contents, caption, props)
```

Description: Tabular contents can be created from cell arrays using cell2TeX. displayRowsTeX calls this function to make the float directly from database contents.

Parameters:

`contents`: Table contents in LaTeX.

`caption`: Table caption.

`props`: A structure with any optional properties.

- `rotate`: Degrees to rotate.
- `width`: Resize to this width.
- `height`: Resize to this height
- `center`: Align to center.
- `shortCaption`: Short version of caption to appear at list of tables.
- `floatType`: LaTeX float to use (default='table').
- `label`: Used for internal LaTeX references.

Returns:

tex_string: LaTeX string for float.

Example:

```
» string2File(TeXfloat(cell2TeX('a', 1; 'b', 2), 'a basic table', ...
struct('rotate', 90, 'label', 'simple-table')))
```

See also: `cell2TeX` (p. **??**), `tests_db/displayRowsTeX` (p. 269)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/12/13

A.57.58 Function `functions/trace2cc`

Summary: Converts a single-column trace vector into a current_clamp object.

Usage:

```
a_cc = trace2cc(a_tr, cip_times, cip_vals, props)
```

Parameters:

`a_tr`: A trace object.

`cip_times`: Start and end times of current injection [ms].

`cip_vals`: A vector of current injection (CIP) parameter values [nA]. The number of the elements in this vector must be capable of dividing the length of the trace evenly.

`props`: A structure with any optional properties.

`Ihold`: [nA] Specifies holding current if different than first step value.

`dt`: Simulation time step of recorded data [s]. Use dt*nout of XPP (Default=1e-3).

`paramsVary`: Structure with variable name associated with an array. If only one value is given cip_vals, use the values for this variable for the multiple trials found in file. (others are passed to current_clamp)

Returns:

a_cc: A current_clamp object.

Example:

```
The following creates a current clamp object from the cur_inj45pA_t
trace object, a current step of holding at -10 to 0 nA at times 50 and
500 ms.  The props 'threshold' and 'paramsStruct' are passed to current_clamp.
» a_cc = trace2cc(cur_inj45pA_t, [50 500], 0, ...
struct('threshold', 10, 'Ihold', -10, 'paramsStruct', ...
struct('gL_nS', 7, 'gKs_nS', 50.1, 'Cm_pA', 5)));
```

See also: plotXPPparamRanges (p. **??**), current_clamp (p. 84)

Author: Cengiz Gunay <cgunay@emory.edu>, 2011/03/04

A.57.59 Function `functions/uniqueValues`

Summary: Find unique rows in a matrix (or column vector).

Usage:

```
[rows, idx] = uniqueValues(data)
```

Description: Version which makes use of sort and diff. Maintains order of the original input.

Parameters:

`data`: A matrix or column vector

Returns:

rows: A matrix or column vector of unique rows. idx: Indices of the unique rows in the original data matrix.

See also: sortedUniqueValues (p. **??**)

Author: Cengiz Gunay <cgunay@emory.edu>, 2004/09/24

A.57.60 Function `functions/updateErrorBars`

Description: Code covered by the BSD License.

Parameters:

`h`**:** (Optional) Handle to figure or axis where to find errorbars (default=gca).

`w, xtype`**:** (Optional) Passed to errorbar_tick.

A.57.61 Function `functions/writeNeuronVecAscii`

Summary: Writes ascii file to be read by Neuron simulator as Vector object.

Usage:

```
writeNeuronVecAscii(filename, datax, datay, dx, dy, unit_x, unit_y, label)
```

Description: It's one line of code just to write the data: dlmwrite(filename, ... [(0:(num_samples - 1))'*dx*1e3, datay*1e-3], '-append', ... 'delimiter', '\t') Data converted to Neuron units of nA, mV, and ms.

Parameters:

`filename`**:** Full path to Neuron file.

`datax`**:** (Optional) X-axis points. Give empty vector to skip.

`datay`**:** Column or row vector of data.

`dx`**:** X-axis resolution in [s] or [V].

`dy`**:** y-axis resolution in [A] or [V].

`unit_x`**:** Units of x-axis; 'V' or 's'.

`unit_y`**:** Units of y-axis; 'A', 'V', or 's'.

`label`**:** Text label to export to Neuron (spaces will be replaced with '_')

Returns:

Nothing.

Example:

```
writeNeuronVecAscii('myvec.dat', [], datay, 1e-4, 1e-3, 's', 'V', 'my membrane voltage');
```

Author: Cengiz Gunay <cengique@users.sf.net> 2012/03/23
