*Response to Reviewer Comments

Database analysis of simulated and recorded electrophysiological datasets with PANDORA's Toolbox

Cengiz Günay, Jeremy R. Edgerton, Su Li,

Thomas Sangrey, Astrid A. Prinz and Dieter Jaeger

April 14, 2009

A Supplementary Materials

A.1 Supplementary Methods

A.1.1 Object-oriented software architecture

PANDORA works in Matlab, which provides a suitable environment for numerical data processing and supports object-oriented (OO) programming. PANDORA uses the object-oriented programming to separate each of its major components in a *class* that holds functions and defines an internal structure for its *objects*. An object stands out by the specific data it contains, separating it from all other objects of that class. Matlab objects make use of the *structure* data type for representing objects. Programs that are placed under directories with a name such as `@myclass/` become functions of the class called `myclass`. The function that has the same name as the class and directory name (e.g., `@myclass/myclass.m`) is called the *constructor*, and it is responsible for creating a structure variable for the class and register it as an object (Alg. 1).

Algorithm 1 Example constructor function `@myclass/myclass.m` that builds a Matlab object.

```
str = struct;                    % create a structure variable
str.name = 'my new integer object';
str.value = 5;
obj = class(obj, 'myclass');  % object belongs to 'myclass'
```

Algorithm 2 Creating a dataset object to construct and manipulate a database.

```
  >> my_dataset_obj =
2       my_dataset_class('data/*.bin', arguments...)
  >> my_database_obj =
4       param_tests_db(my_dataset_obj)
  >> sorted_obj =
6       sortrows(my_database_obj, 'AP_amplitude')
```

A.1.2 Creating a dataset object

Creating a dataset object from a set of files involves calling a dataset constructor (Alg. 2, line 1). This example demonstrates constructing the dataset object from the description of the files and other arguments specifying details of loading those files into Matlab. Together with the information from this function call and from the class definition, the newly created `my_dataset_obj` variable must have all the information that is required to create a database:

1. The location of the data files (e.g., path and file wildcard pattern),
2. The program used to load the data into Matlab (e.g., `readgenesis`),
3. The parameters required to correctly interpret the loaded data (units, scaling factors, etc.),
4. The analysis function to extract measured characteristics from a trace (e.g., `getProfileAllSpikes`),
5. The function to extract simulation or recording parameters associated with a trace (e.g., parse file name to read parameter values).

The extracted parameters and measurements must be uniform to fill a database such that the database object can be constructed automatically (Alg. 2, line 3). In this step, the `params_tests_db` function of the dataset creates the database object by processing all dataset entries (e.g., file names), and computes measured characteristics for each entry. The resulting database table has as many rows as the number of files in the original dataset, and has as many columns as the number of measurements that the analysis function generated plus the number of associated parameters (Fig. 1).

Once all parameters and measurements are loaded into a database format, the database analysis functions can be used. The database can be sorted, partitioned, queried and sifted. The example sorts the rows of the database by the action potential (AP) amplitude measure and returns the new database object, `sorted_obj` (Alg. 2, line 5).

A.1.3 List of input file formats

As of the writing of this manuscript, NeuroShare supports files acquired with devices or programs of Alpha Omega, Cambridge Electronic Design, NeuroExplorer, Plexon, R.C. Electronics Inc., Tucker-Davis Technologies, and Cyberkinetics Inc. (formerly Bionic Technologies Inc). However, NeuroShare only allows files to be loaded into Matlab in the Microsoft Windows operating system. PANDORA can also load acquisition data files from pClamp (Molecular Devices Corporation, Inc., Sunnyvale, CA, U.S.A.) via the Abf_atf_import filter (Giugliano, 2002), and Spike2 (Cambridge Elec-

Algorithm 3 Querying is accomplished by allowing symbolic names for addressing rows and columns of the database matrix. The colon operator (:) in Matlab is a wildcard indicating all elements (i.e., rows and columns respectively in the example on line 3).

```
   >> db_obj2 =
 2      db_obj(1:10, {'neuron_index', 'fire_rate'})
   >> db_obj2 =
 4      db_obj(db_obj(:, 'neuron_index') == 46, :)
   >> db_obj2 =
 6      db_obj(anyRows(db_obj(:, 'neuron_index'), [46; 56; 12]), :)
   >> db_obj2 =
 8      db_obj(db_obj(:, 'neuron_index') ~= 46 &
              (db_obj(:, 'CIP') > 100 |
10             db_obj(:, 'rate') <= 50 ), :)  ,
   >> db_obj2 =
12      model_db_obj(anyRows(model_db_obj(:, 'rate'),
                 neuron_db_obj(:, 'rate')), :)
```

tronic Design Limited, Cambridge, England) which includes a Matlab export function starting in version 6. In addition, some import filters for custom stimulation, dynamic clamp and data acquisition programs are provided (PCDX and NeuroSAGE, Jaeger Lab, Emory Univ., Atlanta, GA, U.S.A.).

A.1.4 Details of the querying capabilities

Querying is provided by re-defining (overloading) the Matlab parenthesis operator ("`()`") to accept selecting rows, columns and *pages* (the third dimension) with numerals or with text labels that match the table metadata (Alg. 3, line 1). This query chooses the first ten rows and the two labeled columns from `my_db`. The resulting selection is stored into a new variable, `new_db`, creating a new database object.

Logical queries work by finding matching row indices after filtering all rows of a database. As an example, an arbitrary "`neuron_index`" column is matched against a scalar value (Alg. 3, line 3). This was achieved by overloading the function of the equality (`==`) comparison operator. The equality operator did not allow finding a match among several values disjunctively, for which the `anyRows` function must be used (Alg. 3, line 5). Both queries return a `new_db` that contains the rows of `my_db` which match the desired neuron indices.

The querying system is similarly expressive to SQL when it comes to selecting entries of a table by a combination of multiple logical expressions. Conjunctive (`&`), disjunctive (`|`) or negative (`~`) Matlab expressions can be formed from constants or from variables obtained from other tables (Alg. 3, line 7). This query finds the database rows of any neuron other than #46 which received a current injection pulse (CIP) larger than 100 pA and responded by firing slower than 50 Hz.

Taking the query results from one table and applying them to another query is called nesting. Allowing nested queries is critical for expressiveness in forming complex queries (Alg. 3, line 11). This query selects `model_db` neurons that fire at the same rates as real neurons of `neuron_db`.

A.1.5 Performance of querying operations

The querying and measure calculation performance measurements in Table 1 were obtained on a PC computer with two 64-bit AMD Opteron 244 processors, each operating at 1800 MHz clock speed with a total of 3 GB of RAM, running MATLAB Version 7.3.0.298 (R2006b) for a 64-bit Linux. The displayed times in the table were obtained by fitting parameters of template polynomial functions (such as $t = a + bn + cm + dnm$) from the observations. For some functions, the parameters were optimized with the `fminsearch` (or `fmins` in newer versions) of Matlab. This resulted in the following equations:

- The equality operator (`==`) took $n \times 77 \times 10^{-6}$ s.
- The `anyRows` function took $7.0 \times 10^{-3} + n \times 1.16 \times 10^{-6} + m \times 9.40 \times 10^{-6} + n \times m \times 0.052 \times 10^{-6}$ s.
- Conjunctions took $m \times 1.8 \times 10^{-3} + n\,(m \times 2.80 \times 10^{-6} - 5.69 \times 10^{-6}) - 1.4 \times 10^{-3}$ s.
- Applying query results took $4.16 \times 10^{-3} + n \times 0.55 \times 10^{-6} + m \times 49.33 \times 10^{-6}$ s.

The time measurements in Table 2 for accessing, indexing and filtering operations used the PANDORA commands in Alg. 4:

- Numerically indexing a Matlab data matrix (line 2) extracted from a DB (line 1).

Algorithm 4 Accessing, indexing and filtering times were measured using these commands.

```
1 >> data = a_db.data
  >> part_data= data(1:100, 101)
3 >> part_data = a_db.data(1:100, 101)
  >> a_new_db = onlyRowsTests(a_db, 1:100, 101)
5 >> a_new_db = onlyRowsTests(a_db, 1:100, 'spike_rate')
  >> a_new_db = a_db(1:100, 101)
7 >> a_new_db = a_db(1:100, 'spike_rate')
```

- Numerically indexing the data matrix of a DB table (line 3).
- Filtering an existing DB table, `a_db`, to produce a new DB table, `a_new_db` using numeric (line 4) and symbolic (line 5) column indexing.
- Filtering using the overloaded parenthesis operators (lines 6 and 7).

A.1.6 Comparing characteristic distributions

Distributions of electrophysiological characteristics from databases can be visualized for comparison (Fig. 5A). In addition, statistics can be calculated to compare empirical distributions to each other (Fig. 5A) or to model distributions. To compare our discrete histograms, we used the Kullback-Leibler (KL) divergence measure (Kullback and Leibler, 1951), which gives the difference of two probability distributions in bits with

$$\mathcal{D}(p_1 \parallel p_2) = \sum_x p_1(x) \log \left(\frac{p_1(x)}{p_2(x)} \right) .$$

A

pAcip	0	100	200	0	100	200
PicroTx	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
KynAcid	0.001	0.001	0.001	0.001	0.001	0.001
TTX	0	0	0	0	0	0
Apamin	0	0	0	0	0	0
drug 4AP	0	0	0	0	0	0
NeuronId	107	107	107	108	108	108
TracesetIndex	109	109	109	111	111	111
NumDuplicates	2	3	1	2	3	1
RowIndex	1	2	6	7	8	12
steady rate	0	25.9982	0	0	29.9673	50.5537

B

PicroTx	0.0001	0.0001
KynAcid	0.001	0.001
TTX	0	0
Apamin	0	0
drug 4AP	0	0
NeuronId	107	108
TracesetIndex	109	111
steady rate 0pA	0	0
steady rate D100pA	25.9982	29.9673
steady rate D200pA	0	50.5537

Table S1: Characteristics extracted from the same neuron in different stimulus conditions could be combined (see Methods and Supp. Matlab Code 2). (A) Before combining, three different current injection values, 0, 100, 200 pA, existed for neurons with NeuronIds 107 and 108 in this example database. (B) After combining, each neuron is represented once and the "steady rate" characteristic extracted at different current injection values was replicated by suffixing a proper stimulation value (e.g., "_D100pA" stood for a depolarizing 100 pA current).

Because this form of the KL divergence is asymmetric, we also provided a symmetric version that is called the *resistor average* (Johnson and Sinanović, 2001; Sinanović and Johnson, 2007), defined as

$$\frac{1}{\mathcal{R}(p_1, p_2)} = \frac{1}{\mathcal{D}(p_1 \parallel p_2)} + \frac{1}{\mathcal{D}(p_2 \parallel p_1)} \, .$$

We implemented these measures under the `histogram_db` database object (See `calcKLhists` function).

A.1.7 Combining trials with different stimulus parameters

After averaging rows with redundant stimulus conditions, the database still contained multiple rows of characteristics for each neuron because of the different stimuli applied (Table S1A). However, many types of statistical analyses require that each neuron be treated as a single set of results (such as the histograms in Fig. 5A), necessitating a database where each row points to one unique neuron. In the specific experimental procedure, the stimulus parameter varied was the magnitude of a current injection pulse (CIP). To achieve a one-neuron-per-row database, we used PANDORA to find all CIP magnitudes applied to each neuron, and merged selected characteristics from each different stimulus magnitude into a single database row (Table S1B). This format increased the efficiency of representation by selecting only needed characteristics for each stimulus condition. For instance, firing rate characteristics were omitted for hyperpolarizing current stimuli since the neurons were silent, but they were kept for depolarizing stimuli because we were interested in firing rate changes with stimulus magnitude. Once the firing rate characteristics from different current injections on one neuron were combined together, it was possible to calculate each neuron's frequency-current (f-I) relationship (e.g., Fig. 11A). Although this approach was specialized for the experimental study where only a current stimulus condition was varied, the underlying generic functions (see next section) can be used for similar purposes in other experimental or simulation protocols. An important feature of these functions was to properly treat the stimulus conditions that were missing in some neurons.

A

PicroTx	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
KynAcid	0.001	0.001	0.001	0.001	0.001	0.001
TTX	0	$7e-09$	0	$7e-09$	0	$7e-09$
Apamin	0	0	0	0	0	0
drug 4AP	0	0	0	0	0	0
NeuronId	107	107	108	108	110	110
D100pA steady rate	25.9982	19.6056	29.9673	22.7628	23.8443	20.9744

B

	Page 1		Page 2		Page 3	
TTX	0	$7e-09$	0	$7e-09$	0	$7e-09$
D100pA steady rate	25.9982	19.6056	29.9673	22.7628	23.8443	20.9744
RowIndex	1	2	3	4	5	6

C

d1_2	−6.3926	−7.2045	−2.8699
PageIndex	1	2	3

D

	Page 1			Page 2		
TTX	0	0	0	$7e-09$	$7e-09$	$7e-09$
D100pA steady rate	25.9982	29.9673	23.8443	19.6056	22.7628	20.9744
RowIndex	1	3	5	2	4	6

Table S2: Steps for multivariate analysis of a sample TTX database with several parameters and one rate characteristic (see commands in Supp. Matlab Code 2). (A) After reducing the example subset DB (Table 4) to only three neurons and two TTX concentrations (see Fig. 8A). (B) Looking for background conditions with only a changing TTX parameter using the `invarValues` function resulted in a database of the rate characteristics in different pages for each separate neuron. `RowIndex` points back to rows of the original DB in panel B to enable retrieving other constant parameters. (C) From the 3D database, differential effects of TTX between its first and second value (d1_2) on the rate characteristic was mostly negative (see Fig. 8C). `PageIndex` points back to the pages of the 3D database in panel C to enable accessing absolute rate values. (D) Swapping the page and row dimensions of the 3D database groups same TTX values in each page, suitable for displaying average effects of TTX (see Fig. 8B).

A.1.8 Multivariate parameter analysis

Multivariate analysis is used to find parameter effects on measured characteristics in databases with several changing parameters (Table S2A). First, the database is sorted to identify background parameter combinations and to isolate the effect of target parameters (Table S2B and Fig. 8A; see the `invarValues` function in Supp. Mat. A.3). This is equivalent to the "group by" clause in extensions to the SQL language (`http://dev.mysql.com/doc/refman/5.1/en/select.html`), but it differs in implementation. In PANDORA, this function produces a new database with a three-dimensional matrix (Table S2B; see the `tests_3D_db` class in Supp. Mat. A.3) where each page of the matrix (a slice in its third dimension) denotes an invariant background parameter combination (e.g., other drugs applied) with varying target parameters (e.g., target drug or model parameter) and associated characteristics (e.g., firing rate). The database contains as many pages as there are invariant backgrounds. This three-dimensional database can be targeted to several types of further analyses:

1. Differential parameter effects:

   Since each page of the database contains the measured characteristics for increasing values of target parameters, subtracting successive rows gives the difference in measured characteristics for a change in parameter values (Table S2C and see Fig. 8C; see the `diff2D` function). The subtraction is done separately in each page of the database. Summing the rows of this database gives the total change in characteristics from increasing parameters from the lowest to highest values.

2. Average parameter effects:

   The statistical summary of parameter effects can be found when the page and row dimensions of the `tests_3D_db` object are swapped with the `swapRowsPages` function. This puts measured characteristics for all combinations of background parameters in the same page and different target parameter values in different pages (Table S2D). The statistics are calculated separately for each page of the database, resulting in a database table of the average characteristics for each target parameter value (see Fig. 8B; see the `statsMeanStd` function). Statistics can also be calculated to the results of the differential parameter effects (e.g., to find the average effect of changing a parameter).

3. Reducing database contents based on target parameter values:

   After swapping page and row dimensions, the pages with different target values can be combined back together to yield a two-dimensional database (Table S1; see the `mergePages` function). When inserting the characteristics from each page into a new database, a suffix is added to indicate the page they came from. Characteristics to take from each page can be chosen to keep only necessary characteristics. The `mergeMultipleCIPsinOne` function uses this method to generate a three-dimensional database by finding target values of the current stimulus parameter and then choosing desired characteristics for each of the current stimulus magnitudes (Table S1). For instance, firing rate characteristics are not needed for hyperpolarizing current stimuli.

A.1.9 Distance measures for comparing individual neuron representations

To enable comparing characteristic profiles across neurons, PANDORA offered several methods of calculating the distance between two profiles. In PANDORA, the normalized Euclidean distance ($d_{n,m}$) between the two neuron representations (x, y), is calculated as the average of absolute differences (error) of corresponding N individual characteristics, each normalized by the measure's standard deviation (σ_i) from a reference database, with

$$d_{x,y} = \sum_{i=1}^{N} |x_i - y_i| / N\sigma_i \,,$$

where x_i and y_i represent the ith characteristic of the neurons. The contribution of individual measure differences to this Euclidean distance measure are inversely proportional to that characteristic's variance in the physiology database in order to weight each characteristic's importance in accordance with its physiological variability. In addition, the error from each measure can be weighted arbitrarily to reflect other physiological reasons and assumptions.

Normalized Euclidean distance assumes that each measured characteristic

Algorithm 5 Finding closest matches of a neuron representation (row 34 in `from_db_obj`) in another database (`to_db_obj`) involves two steps. First, the source neuron representation is extracted and put in a criterion obj (`crit_row_obj` on line 1). Then, the target database is sorted according to distance to the criterion neuron representation to yield a ranked database (`a_ranked_db_obj` on line 2).

```
1 >> crit_row_obj = matchingRow(from_db_obj, 34)
  >> a_ranked_db_obj = rankMatching(to_db_obj, crit_row_obj)
```

is independent. For interdependent characteristics, the *Mahalonobis distance* method in PANDORA calculates a distance accounting for the covariance between characteristics:

$$d(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})^T P^{-1} (\vec{x} - \vec{y})}\,,$$

where P is the covariance matrix of characteristics in the database and $\vec{x}$ and $\vec{y}$ are neuron characteristic vectors (Mahalanobis, 1936). These distances can be calculated across neurons in a dataset when the measured characteristics are collected in a database.

A.1.10 Matching and ranking neuron representations according to distance

The extracted characteristics from neurons can be used to find quantitative distances between neuron representations (see above). Commands used for this comparison are reviewed here (Alg. 5). The example in the Results section compared a recorded neuron to model neurons (Table S3). The

	Criterion	Crit. STD	Rank 1	(Δ/STD)	Rank 2	(Δ/STD)	Rank 3	(Δ/STD)	Rank 4	(Δ/STD)
NaF			250		250		250		250	
NaP			0.5		0.5		1		0.5	
Kv2			1		0.1		1		10	
Kv3			50		50		50		50	
Kv4f			10		10		10		20	
KCNQ			0.08		0.08		2		0.08	
SK			2		2		8		2	
CaHVA			0.3		0.3		0.03		0.03	
HCN			0.2		0.2		0.2		0.2	
morph			1		1		1		1	
trial			101396		101387		1768		100757	
AHP depth	12.8378	2.8259	9.1446	(−1.31)	8.4588	(−1.55)	8.9246	(−1.38)	12.1371	(−0.25)
AP amplitude	68.6496	9.3287	59.784	(−0.95)	59.417	(−0.99)	59.9227	(−0.94)	60.1521	(−0.91)
AP threshold	−44.1695	4.1218	−48.1403	(−0.96)	−48.1504	(−0.97)	−47.8401	(−0.89)	−48.0135	(−0.93)
D100pA first 100ms rate	61.9785	16.5595	46.62	(−0.93)	49.7512	(−0.74)	56.3063	(−0.34)	45.5581	(−0.99)
D100pA steady rate	41.843	11.0971	43.4385	(+0.14)	44.9438	(+0.28)	44.9491	(+0.28)	48.8145	(+0.63)
H100pA potential	−84.0265	12.8929	−76.9437	(+0.55)	−76.8403	(+0.56)	−77.4378	(+0.51)	−76.9472	(+0.55)
H100pA sag	2.4265	9.5843	1.7168	(−0.07)	1.5728	(−0.09)	1.8669	(−0.06)	1.5188	(−0.09)
resting potential	−57.0681	5.9278	−59.4825	(−0.41)	−59.3612	(−0.39)	−62.1849	(−0.86)	−59.3711	(−0.39)
spont firing rate	8.8003	4.6848	9.6764	(+0.19)	9.8382	(+0.22)	6.8306	(−0.42)	8.3039	(−0.11)
Distance			0.85614		0.85662		0.86978		0.8709	

Table S3: Model neurons that best matched a selected recorded neuron (Criterion) could be found by comparing their characteristic profiles (see Supp. Methods A.1.10). The selected recorded neuron's characteristics and the standard deviation of the characteristics in the recorded neuron database (Crit. STD) were used to calculate the normalized Euclidean distance (Distance) for each of the model neurons (see Methods). The top four models shown had the lowest distances to the selected neuron, which was calculated from the average of the individual normalized characteristic differences shown in parenthesis (Δ/STD). These differences were normalized by dividing with the criterion STD such that they showed the characteristic difference in number of standard deviations. Similarly, the distance calculated indicated the average STD difference from the selected neuron.

table showed the ranked database object (`a_ranked_db_obj` on line 2 of Alg. 5) which contains both the criterion row information (selected row and the database's standard deviation) and the resulting ranked target database contents.

A.1.11 Plotting subsystem

PANDORA improves Matlab's plotting capabilities by providing an object structure for creating complex plots. A generic plot object is defined that can be superposed over other plot objects and can also be stacked horizontally or vertically. Several commonly used optional plot features, such as those for managing customized titles, axis labels and the space allocated for them, are intelligently calculated. Plots are automatically redrawn when they are resized—enabling most efficient use of the drawing area at different scales. Plot objects with customized labels and superposed components can be displayed in Matlab figures, printed to a file, or saved as a Matlab object to be used later. These plotting functions can be downloaded and used independent of PANDORA (`http://userwww.service.emory.edu/~cgunay/pandora/#cgmplot`).

PANDORA objects are visualized by functions that return generic plot objects. These objects can be plotted directly, or incorporated as a part of a more complex plot (e.g., Fig. 11A).

Algorithm 6 Functions needed to define a new datatype for creating a database from thermostat readings. The constructor function (line 1) stores the raw data, which is two arrays of thermostat readings (`thermA` and `thermB`), into the `thermostat_obj` object. The `getResults` function (line 7) returns two desired characteristics about if the average of saved thermostat readings cross specific thresholds.

```
   function thermostat_datatype(thermA, thermB)
 2 str = struct;
   str.thermA = thermA;
 4 str.thermB = thermB;
   thermostat_obj = class(str, 'thermostat_datatype')
 6 end
   function results = getResults(thermostat_obj)
 8 results = struct;
   results.isHotInside = mean(thermostat_obj.thermA) > 70;
10 results.isColdOutside = mean(thermostat_obj.thermB) < 40;
   end
```

A.1.12 Making a database from a custom datatype

If a dataset and its datatype items are not supported by existing PANDORA structures, new classes can be defined. Here, we review the steps to create a database from a hypothetical thermostat dataset to demonstrate this process.

First, a proper wrapper class for holding basic raw data traces (i.e., thermostat readings) must be defined (Alg. 6). The `getResults` function in this class defines measurements to be entered into the database (Alg. 6, line 7).

Second, a way to load raw data items must be identified to process the dataset and construct the database (Section A.1.2). This dataset requires defining a new dataset subclass, `thermostat_dataset` (Alg. 7). The characteristics and metadata are simple, so minimal effort is required to build this

Algorithm 7 Functions needed to define a new dataset class, `thermostat_dataset`, for creating a database from thermostat readings. The constructor function (line 1) stores a list of sequence numbers that represent the dataset of thermostat readings and creates a subclass of `params_tests_dataset`. The `loadItemProfile` function (line 6) gets called for each item in the dataset when addressed by its index. The function reads the raw thermostat data from an external source using the sequence number and constructs a `thermostat_datatype` object (Alg. 6). The `datatype` object is used to extract the desired characteristics and the results are encapsulated in a `results_profile` object to be entered into the database. The `getItemParams` function (line 11) returns parameter values for each dataset item (the sequence number and sensitivity of the thermostat), similar to `loadItemProfile`. Finally, the `paramNames` function (line 15) defines the names of the two parameters returned by `getItemParams` for representing them in the database.

```
 1  function thermostat_dataset(dataset_seq_numbers)
    dataset_obj = ...
 3    class(obj, 'thermostat_dataset', ...
              params_tests_dataset(dataset_seq_numbers));
 5  end
    function a_profile = loadItemProfile(dataset_obj, index)
 7  [thermA, thermB] = readTherm(dataset_obj.list{index});
    a_profile = ...
 9    results_profile(getResults(thermostat_datatype(thermA, thermB)));
    end
11  function params = getItemParams(dataset_obj, index)
    seq_num = dataset_obj.list{index};
13  params = [ seq_num, getSensitivityFromSeq(seq_num) ];
    end
15  function param_names = paramNames(dataset_obj)
    param_names = { 'Seq_No', 'Sensitivity' };
17  end
```

class. The class is composed of a constructor function that takes the new thermostat numbers, and file and parameter loader functions that read the raw data (line 6) and metadata (line 11, 15) to assign them to the proper locations in the dataset structures.

The thermostat dataset is a simple example and more complex datasets can be constructed similarly. It is possible to use PANDORA's existing data types for construct a new data structure. One of the existing classes is the `trace` class that can hold any type of time series data in a vector form (e.g., current, extracellular voltage, or EEG traces) with parameters for the x-axis and y-axis resolutions. To accommodate a new type of data that requires parameters or data not supported by the basic `trace` class, a new `trace` subclass similar to `cip_trace` can be created to take advantage of existing functions defined in `trace`. `cip_trace` provides the additional parameters for the start time and duration of the current-injection pulse (CIP) for use in measurement functions specific to this stimulus type. All other generic measurements were reused from the `trace` class. The data type is accompanied by a compatible dataset class, such as in the above thermostat example. The `params_tests_dataset` class provides an abstract mechanism to acquire the data from a dataset item (e.g., which channels and traces to load) and parameters to associate with it. This class can be used for loading simple traces from data files (see the `params_tests_dataset` class in the Supp. Mat. A.3 and the manual at Günay, 2007, 2008a,b), or it can be extended to a subclass if loading the new data items requires special operations, such as in the above thermostat dataset.

For any new subclass, including children of `params_tests_dataset` and `trace` classes, one needs to define generic functions that allow interaction with PANDORA conventions such as overloaded indexing and plotting features. These functions, such as `get` and `set` that allow reading and changing object properties, can be simply copied from any other PANDORA class provided that some keywords contained are properly renamed (see the manual for more information).

A.1.13 Custom dataset for activity sensors

As a real application of making a custom dataset, we review the details of the analysis of the sensor database from the lobster (see Methods and Results). It was suggested that homeostatic regulation of neuron activity could improve by having three sensors in each neuron (Liu et al, 1998). But this required considering 85,750 possible combinations of the original 366 sensors. To iterate across the 85,750 sensor combinations, we created a new PANDORA dataset class (see Supp. Methods A.1.12), called `triplet_dataset`, that contained items of FSD sensor triplets. We defined the function `loadItemProfile` (see Supp. Methods A.1.12) that is run for each sensor triplet, which trains a classifier to best predict functional networks based only on these sensor readings. The prediction success rate of this classifier on a 10,000-model network subset was used to indicate the performance of the chosen sensors. In addition, the function calculated other characteristics of the chosen sensors, such as how much separation of functional networks from non-functional networks they obtained in terms of center of gravity between the two classes. These calculated characteristics were entered into a database from the `triplet_dataset` class, which allowed us to compare the classification success of the 85,750 sensor triplets.

A.2 Supplementary Matlab Scripts and Data

Supp. Matlab Code 1 Example Matlab script to load a single intracellular data trace and find its spikes and to plot a few histograms from a spike database obtained from this trace (Alg. 8).

Supp. Matlab Code 2 Example Matlab script to averaging similar stimulus conditions, merging entries into single rows, and finding invariant parameter effects. After averaging, the standard deviation (STD) becomes essential for interpreting the averaged results and it is included in a second *page* of the new database, adding a third dimension to the data matrix (Alg. 9).

A.3 List of major PANDORA components and functions

See User and Programmer Manuals in Günay, 2007, 2008a,b for tutorials and usage details of components and functions.

Algorithm 8 Example Matlab script 1.

```
% Examples in this file require the supplementary Matlab data file:
% load an intracellular trace of 160 ms
load('supp_mat_1_dat.mat')

% create trace object with 10 kHz sample rate and mV units
if exist('filtfilt', 'file') == 2
    % using the signal processing toolbox
    a_trace = trace(test_data, 1e-4, 1e-3, 'test_trace');
    a_spikes = spikes(a_trace);
else
    % using the signal processing toolbox
    a_trace = trace(test_data, 1e-4, 1e-3, 'test_trace', ...
                    struct('spike_finder', 2, 'threshold', -85));
    a_spikes = spikes(a_trace);
end

% disable warnings
warning off backtrace
warning off calcInitVm:info

% extract spike characteristics
[results period_spikes a_spikes_db spikes_stats_db spikes_hists_dbs]= ...
  analyzeSpikesInPeriod(a_trace, a_spikes, periodWhole(a_trace), '');

% display info about spikes DB extracted from data
a_spikes_db %#ok<NOPTS>

% plot all spikes annotated on the trace (takes a long time for the
% 10-second trace!)
plot(a_spikes_db);

% plot some histograms
plot(histogram(a_spikes_db, 'InitVm')); % threshold voltage
plot(histogram(a_spikes_db, 'MaxAHP')); % AHP magnitude
plot(histogram(a_spikes_db, 'MinVm')); % minimum of AHP voltage

% make the example plot in the paper
plotFigure(plot_abstract(a_spikes_db, 'annotated_spike_characteristics', ...
           struct('fixedSize', [2 2], 'axisLimits', [1450 1555 -100 -60], ...
                  'quiet', 1)))
```

Algorithm 9 Example Matlab script 2.

```
% Examples in this file require the supplementary Matlab data file:
load 'supp_mat_2_dat.mat'

% Example for averaging rows
% ***************************************
% display example database contents before averaging (see Table 3A in
manuscript)
displayRows(sortrows(raw_example1_db, 'pAcip'))
% average the rows
avg_example1_db = meanDuplicateParams(raw_example1_db);
% display contents after averaging
displayRows(sortrows(avg_example1_db, 'pAcip'))
% optionally, generate a formatted LaTeX table (see Table 3B in manuscript)
string2File(displayRowsTeX(sortrows(avg_example1_db, 'pAcip'), ...
                           'Parameter␣in␣the␣raw␣cell␣database.', ...
                           struct('rotate', 0, 'width', '\textwidth', ...
                                  'label', 'tbl:ttx-cells')), 'example-table.tex')
% Example for combining (merging) rows
% ***************************************
% display example database contents before merging (see Table S1A in manuscript)
displayRowsTeX(sortrows(merge_example_db, 'NeuronId'))
% merge columns into rows
merged_db = ...
   mergeMultipleCIPsInOne(delColumns(merge_example_db(:, :, 1), ...
                                     {'NumDuplicates', 'RowIndex'}), ...
                          {'_0pA', 9, ...
                              '_D100pA', 9, '_D200pA', 9})
% display after merging
displayRowsTeX(sortrows(merged_db, 'NeuronId'))
% optionally, generate a formatted LaTeX table (see Table S1B in manuscript)
string2File(displayRowsTeX(sortrows(merged_db, 'NeuronId'), ...
                           'Parameter␣in␣the␣raw␣cell␣database.', ...
                           struct('rotate', 0, 'width', '\textwidth', ...
                                  'label', 'tbl:ttx-cells')), ['example-' ...
                     'table.tex'])
% Example for invariant parameter effects
% ***************************************
% display database contents of TTX cells (see Table 4 in manuscript)
displayRows(sortrows(ttx_example1_db, 'NeuronId'))
% select two TTX concentrations and three neurons
ttx_reduced_db = ttx_example1_db(anyRows(ttx_example1_db(:, 'TTX'), [0; ...
                     7e-9]) & anyRows(ttx_example1_db(:, 'NeuronId'), [107; ...
                     108; 110]), :)
% find invariant parameter effects on extracted characteristics
ttx_invar_reduced_db = invarParam(delColumns(ttx_reduced_db, 'TracesetIndex'), 'TTX')
% Find differential effects of TTX on rate
ttx_diffed_db = diff2D(ttx_invar_reduced_db, 'D100pA_steady_rate');
% Plot as a bar plot (Figure 8C in manuscript)
plotFigure(plotBox(ttx_diffed_db(:, 'd1_2'), '', ...
                   struct('quiet', 1, 'putLabels', 1, 'fixedSize', ...
                          [2.5 2], 'colormap', [0 0 0])))
% Find statistics of parameter effects
ttx_stats_db = statsMeanSE(swapRowsPages(ttx_invar_reduced_db));
% plot bar plot showing parameter effects (see Figure 8B in manuscript)
plotFigure(plot_bars(ttx_stats_db(:, {'TTX', 'D100pA_steady_rate'}, :), ...
                     '', ...
                     struct('pageVariable', 'TTX', 'axisLimits', [NaN NaN ...
                     20 30], 'quiet', 1, 'fixedSize', [2.5 2], 'colormap', ...
                            [0 0 0])))
```

Basic data wrapper classes that define the measurements:	
trace	A voltage/current trace.
spikes	Spike times of a trace.
spike_shape	Averaged spike shape from a trace.
period	Defines time periods to operate on trace or spikes objects.
cip_trace	A voltage trace with a CIP applied.
Profile classes that hold measurement results:	
result_profile	Base class that holds a results structure.
trace_profile	Generic example class for holding a trace profile.
cip_trace_profile	Holds cip_trace results. Template class designed only for sub-classing.
cip_trace_allspikes_profile	Created by cip_trace/getProfileAllSpikes, contains statistics of spike shape measures from individual spikes.
Dataset classes that point to or hold raw data:	
params_tests_dataset	Base class for datasets.
params_tests_fileset	Holds a list of filenames and associated information, capable of creating a params_tests_db.
params_cip_trace_fileset	Fileset from which cip_trace objects can be created.
physiol_cip_traceset	Dataset of a traceset from a single file.
physiol_cip_traceset_fileset	Dataset of a tracesets from many files.

Database classes created from datasets:	
tests_db	Base class for databases. Contains many utilities.
params_tests_db	DB extended to hold parameter values associated with results.
spikes_db	Holds measures from each individual spike in a trace.
test_3D_db	3D database of tests that vary with a third variable.
corrcoefs_db	Holds correlation coefficients.
histogram_db	Holds histogram bins.
stats_db	Holds statistical measurements.
ranked_db	Database ranked for a criterion, holds error values or distances.

Major database functions of tests_db:	
corrcoef	Calculates correlation coefficients of selected columns.
crossProd	Creates a new database by taking the set cross-product of two databases.
cov	Calculates covariance matrix.
enumerateColumns	Find unique values in selected columns and replace with enumerated numbers.
factorac	Factor analysis.
fillMissingColumns	Put this value in NaN columns.
groupBy	Equivalent to SQL GROUP BY clause, splice table by unique values of selected columns.
histogram	Calculates histogram of selected column.
invarValues	Find unique values of background columns and put the remaining columns in pages of 3D table.
princomp	Finds principal components.
sortrows	Sorts rows for chosen columns.
statsMeanStd	Calculates mean and STD of rows.
matchingRow	Extracts a selected row and information about statistics of a database for comparing to another database.
rankMatching	Uses the extracted information from matchingRow to rank a database according to distance.

Bundle classes that combine a database and original dataset:

dataset_db_bundle	Base class of bundles. Puts a processed and raw DBs with datasets.
model_ct_bundle	Bundle for model databases.
physiol_bundle	Bundle for physiology databases.

Plotting classes:

plot_abstract	Base class that holds information sufficient to generate any plot.
plot_simple	Simple extension that works for most simple plots.
plot_stack	Holds horizontal or vertical stack of plot_abstract objects.
plot_superpose	Allows superposing different plot_abstract's in the same axis.
plot_bars	Multi-axis bar plot with extended errorbars.
plot_errorbar	Errorbar plot.
plot_errorbars	Multi-axis errorbar plot.

Classes for generating formatted reports:

doc_generate	Base class for all document classes.
doc_plot	Holds a plot_abstract and captions, etc.
doc_multi	Combines multiple doc_generate objects.

Script control classes for cluster computing:

script_factory	Generates a set of scripts based on a recipe.
script_array	Designed to execute an array job serially on a computer.
script_array_for_cluster	Executes an array job on a cluster computer.

References

Giugliano M (2002) Abf_atf_import: Importing Axon Binary and Axon Text Files into MATLAB for processing, visualization and analysis. MATLAB File Exchange, URL http://www.mathworks.com/matlabcentral/fileexchange

Günay C (2007) Plotting and analysis for neural database-oriented research applications (PANDORA) toolbox. URL http://userwww.service.emory.edu/~cgunay/pandora

Günay C (2008a) PANDORA Neural Analysis Toolbox. Intenational Neuroinformatics Coordinating Facility (INCF) Software Center, URL http://software.incf.org/software/44/view/PANDORA

Günay C (2008b) PANDORA Neural Analysis Toolbox. SimToolDB, URL http://senselab.med.yale.edu/SimToolDB

Johnson DH, Sinanović S (2001) Symmetrizing the kullback-leibler distance. Tech. rep., Electrical & Computer Engineering Department, MS380 Rice

University Houston, Texas 77005-1892, URL http://www-dsp.rice.edu/~dhj/resistor.pdf

Kullback S, Leibler R (1951) On information and sufficiency. Annals of Mathematical Statistics 22(1):79–86

Liu Z, Golowasch J, Marder E, Abbott LF (1998) A model neuron with activity-dependent conductances regulated by multiple calcium sensors. J Neurosci 18(7):2309–20

Mahalanobis P (1936) On the generalized distance in statistics. Proceedings of the National Institute of Science of India 12:49–55

Sinanović S, Johnson DH (2007) Toward a theory of information processing. Signal Processing 87:1326–44